

RNeXML: a package for reading and writing richly annotated phylogenetic, character, and trait data in R

Carl Boettiger^{*,a}, Scott Chamberlain^c, Rutger Vos^d, Hilmar Lapp^b

^a*Center for Stock Assessment Research, Department of Applied Math and Statistics, University of California, Mail Stop SOE-2, Santa Cruz, CA 95064, USA*

^b*Center for Genomic and Computational Biology, Duke University, Durham, NC, USA and National Evolutionary Synthesis Center and Duke University, Durham, NC, USA*

^c*University of California, Berkeley, CA, USA*

^d*Naturalis Biodiversity Center, Leiden, the Netherlands*

Abstract

1. NeXML is a powerful and extensible exchange standard recently proposed to better meet the expanding needs for phylogenetic data and metadata sharing. Here we present the RNeXML package, which provides users of the R programming language with easy-to-use tools for reading and writing NeXML documents, including rich metadata, in a way that interfaces seamlessly with the extensive library of phylogenetic tools already available in the R ecosystem.
2. Wherever possible, we designed RNeXML to map NeXML document contents, whose arrangement is influenced by the format's XML Schema definition, to their most intuitive or useful representation in R. To make NeXML's powerful facility for recording semantically rich machine-readable metadata accessible to R users, we designed a functional programming interface to it that hides the semantic web standards leveraged by NeXML from R users who are unfamiliar with them.
3. RNeXML can read any NeXML document that validates, and it generates valid NeXML documents from phylogeny and character data in various R representations in use. The metadata programming interface at a basic level aids fulfilling data documentation best practices, and at an advanced level preserves NeXML's nearly limitless extensibility, for which we provide a fully working demonstration. Furthermore, to lower the barriers to sharing well-documented phylogenetic data, RNeXML has started to integrate with taxonomic metadata augmentation services on the web, and with online repositories for data archiving.
4. RNeXML allows R's rich ecosystem to read and write data in the NeXML format through an interface that is no more involved than reading or writing data from other, less powerful data formats. It also provides an interface designed to feel familiar to R programmers and to be consistent with recommended practices for R package development, yet that retains the full power

*Corresponding author
Email address: `cboettig(at)gmail.com` (Carl Boettiger)

for users to add their own custom data and metadata to the phylogenies they work with, without introducing potentially incompatible changes to the exchange standard.

Introduction

Users of the popular statistical and mathematical computing platform R (R Core Team 2014) enjoy a wealth of readily installable comparative phylogenetic methods and tools (O’Meara 2014). Exploiting the opportunities arising from this wealth for complex and integrative comparative research questions relies on the ability to reuse and integrate previously generated or published data and metadata. The expanding data exchange needs of the evolutionary research community are rapidly outpacing the capabilities of most current and widely used data exchange standards (Vos *et al.* 2012), which were all developed a decade or more ago. This has resulted in a radiation of different data representations and exchange standard “flavors” that are no longer interoperable at the very time when the growth of available data and methods has made that interoperability most valuable. In response to the unmet needs for standardized data exchange in phylogenetics, a modern XML-based exchange standard, called NeXML, has recently been developed (Vos *et al.* 2012). NeXML comprehensively supports current data exchange needs, is predictably machine-readable, and is forward compatible.

The exchange problem for phylogenetic data is particularly acute in light of the challenges in finding and sharing phylogenetic data without the otherwise common loss of most data and metadata semantics (Stoltzfus *et al.* 2012; Drew *et al.* 2013; Cranston *et al.* 2014). For example, the still popular NEXUS file format (Maddison *et al.* 1997) cannot consistently represent horizontal gene transfer or ambiguity in reading a character (such as a DNA sequence base pair). This and other limitations have led to modifications of NEXUS in different ways for different needs, with the unfortunate result that NEXUS files generated by one program can be incompatible with another (Vos *et al.* 2012). Without a formal grammar, software based on NEXUS files may also make inconsistent assumptions about tokens, quoting, or element lengths. Vos *et al.* (2012) estimates that as many as 15% of the NEXUS files in the CIPRES portal contain unrecoverable but hard to diagnose errors.

A detailed account of how the NeXML standard addresses these and other relevant challenges can be found in Vos *et al.* (2012). In brief, NeXML was designed with the following important properties. First, NeXML is defined by a precise grammar that can be programmatically **validated**; i.e., it can be verified whether a file precisely follows this grammar, and therefore whether it can be read (parsed) without

errors by software that uses the NeXML grammar (e.g. RNeXML) is predictable. Second, NeXML is **extensible**: a user can define representations of new, previously unanticipated information (as we will illustrate) without violating its defining grammar. Third and most importantly, NeXML is rich in **computable semantics**: it is designed for expressing metadata such that machines can understand their meaning and make inferences from it. For example, OTUs in a tree or character matrix for frog species can be linked to concepts in a formally defined hierarchy of taxonomic concepts such as the Vertebrate Taxonomy Ontology (Midford *et al.* 2013), which enables a machine to infer that a query for amphibia is to include the frog data in what is returned. (For a more broader discussion of the value of such capabilities for evolutionary and biodiversity science we refer the reader to Parr et al. (2011).) To make the capabilities of NeXML available to R users in an easy-to-use form, and to lower the hurdles to adoption of the standard, we present RNeXML, an R package that aims to provide easy programmatic access to reading and writing NeXML documents, tailored for the kinds of use-cases that will be common for users and developers of the wealth of evolutionary analysis methods within the R ecosystem.

The RNeXML package

The RNeXML package is written entirely in R and available under a Creative Commons Zero public domain waiver. The current development version can be found on Github at <https://github.com/ropensci/RNeXML>, and the stable version can be installed from the CRAN repository. RNeXML is part of the rOpenSci project. Users of RNeXML are encouraged to submit bug reports or feature requests in the issues log on Github, or the phylogenetics R users group list at r-sig-phylo@r-project.org for help. Vignettes with more detailed examples of specific features of RNeXML are distributed with the R package and serve as a supplement to this manuscript. Each of the vignettes can be found at <http://ropensci.github.io/RNeXML/>.

Representation of NeXML documents in R

Conceptually, a NeXML document has the following components: (1) phylogeny topology and branch length data, (2) character or trait data in matrix form, (3) operational taxonomic units (OTUs), and (4) metadata. To represent the contents of a NeXML document (currently in memory), RNeXML defines the `nexml` object type. This type therefore holds phylogenetic trees as well as character or trait matrices, and all metadata, which is similar to the phylogenetic data object types defined in the `phylobase`

90 package (NESCENT R Hackathon Team 2014), but contrasts with the more widely used ones defined
91 in the **ape** package (Paradis *et al.* 2004), which represents trees alone.

92 When reading and writing NeXML documents, **RNeXML** aims to map their components to and from,
93 respectively, their most widely used representations in R. As a result, the types of objects accepted
94 or returned by the package's methods are the **phylo** and **multiPhylo** objects from the **ape** package
95 (Paradis *et al.* 2004) for phylogenies, and R's native **data.frame** list structure for data matrices.

96 *Reading phylogenies and character data*

97 The method **nexml_read()** reads NeXML files, either from a local file, or from a remote location via
98 its URL, and returns an object of type **nexml**:

```
nex <- nexml_read("components/trees.xml")
```

99 The method **get_trees_list()** can be used to extract the phylogenies as an **ape::multiPhylo** object,
100 which can be treated as a list of **ape::phylo** objects:

```
phy <- get_trees_list(nex)
```

101 The **get_trees_list()** method is designed for use in scripts, providing a consistent and predictable
102 return type regardless of the number of phylogenies a NeXML document contains. For greater convenience
103 in interactive use, the method **get_trees()** returns the R object most intuitive given the arrangement
104 of phylogeny data in the source NeXML document. For example, the method returns an **ape::phylo**
105 object if the NeXML document contains a single phylogeny, an **ape::multiPhylo** object if it contains
106 multiple phylogenies arranged in a single **trees** block, and a list of **ape::multiPhylo** objects if it
107 contains multiple **trees** blocks (the capability for which NeXML inherits from NEXUS).

108 If the location parameter with which the **nexml_read()** method is invoked is recognized as a URL,
109 the method will automatically download the document to the local working directory and read it from
110 there. This gives convenient and rapid access to phylogenetic data published in NeXML format on the
111 web, such as the content of the phylogenetic data repository TreeBASE (Piel *et al.* 2002, 2009). For
112 example, the following plots a tree in TreeBASE (using **ape**'s **plot** function):

```

tb_nex <- nexml_read(
  "https://raw.githubusercontent.com/TreeBASE/supertreebase/master/data/treebase/S100.xml")
tb_phy <- get_trees_list(tb_nex)
plot(tb_phy[[1]])

```

113 The method `get_characters()` obtains character data matrices from a `nexml` object, and returns them
 114 as a standard `data.frame` R object with columns as characters and rows as taxa:

```

nex <- nexml_read("components/comp_analysis.xml")
get_characters(nex)

```

	log snout-vent length	reef-dwelling
115 taxon_8	-3.2777799	0
116 taxon_9	2.0959433	1
117 taxon_10	3.1373971	0
118 taxon_1	4.7532824	1
119 taxon_2	-2.7624146	0
120 taxon_3	2.1049413	0
121 taxon_4	-4.9504770	0
122 taxon_5	1.2714718	1
123 taxon_6	6.2593966	1
124 taxon_7	0.9099634	1

126 A NeXML data matrix can be of molecular (for molecular sequence alignments), discrete (for most
 127 morphological character data), or continuous type (for many trait data). To enable strict validation
 128 of data types NeXML allows, and if their data types differ requires multiple data matrices to be
 129 separated into different “blocks”. Since the `data.frame` data structure in R has no such constraints, the
 130 `get_characters()` method combines such blocks as separate columns into a single `data.frame` object,
 131 provided they correspond to the same taxa. Otherwise, a list of `data.frames` is returned, with list
 132 elements corresponding to characters blocks. Similar to the methods for obtaining trees, there is also a
 133 method `get_characters_list()`, which always returns a list of `data.frames`, one for each character
 134 block.

135 *Writing phylogenies and character data*

136 The method `nexml_write()` generates a NeXML file from its input parameters. In its simplest
137 invocation, the method writes a tree to a file:

```
data(bird.orders)
nexml_write(bird.orders, file = "birds.xml")
```

138 The first argument to `nexml_write()` is either an object of type `nexml`, or any object that can be coerced
139 to it, such as in the above example an `ape::phylo` phylogeny. Alternatively, passing a `multiPhylo`
140 object would write a list of phylogenies to the file.

141 In addition to trees, the `nexml_write()` method also allows to specify character data as another
142 parameter. The following example uses data from the comparative phylogenetics R package `geiger`
143 (Pennell *et al.* 2014).

```
library("geiger")
data(geospiza)
nexml_write(trees = geospiza$phy,
            characters = geospiza$dat,
            file="geospiza.xml")
```

144 Note that the NeXML format is well-suited for incomplete data: for instance, here it does not assume
145 the character matrix has data for every tip in the tree.

146 *Validating NeXML*

147 File validation is a central feature of the NeXML format which ensures that any properly implemented
148 NeXML parser will always be able to read the NeXML file. The function takes the path to any NeXML
149 file and returns `TRUE` to indicate a valid file, or `FALSE` otherwise, along with a display of any error
150 messages generated by the validator.

```
nexml_validate("geospiza.xml")
```

151 [1] TRUE

152 The `nexml_validate()` function performs this validation using the online NeXML validator (when a
153 network connection is available), which performs additional checks not expressed in the NeXML schema
154 itself (Vos *et al.* 2012). If a network connection is not available, the function falls back on the schema
155 validation method from the XML package (Lang 2013).

156 *Creating and populating `nexml` objects*

157 Instead of packaging the various components for a NeXML file at the time of writing the file, RNeXML
158 also allows users to create and iteratively populate in-memory `nexml` objects. The methods to do this
159 are `add_characters()`, `add_trees()`, and `add_meta()`, for adding characters, trees, and metadata,
160 respectively. Each of these functions will automatically create a new `nexml` object if not supplied with
161 an existing one as the last (optional) argument.

162 For example, here we use `add_trees()` to first create a `nexml` object with the phylogeny data, and
163 then add the character data to it:

```
nexObj <- add_trees(geospiza$phy)
nexObj <- add_characters(geospiza$dat, nexObj)
```

164 The data with which a `nexml` object is populated need not share the same OTUs. RNeXML automatically
165 adds new, separate OTU blocks into the NeXML file for each data matrix and tree that uses a different
166 set of OTUs.

167 Other than storage size, there is no limit to the number of phylogenies and character matrices that
168 can be included in a single NeXML document. This allows, for example, to capture samples from a
169 posterior probability distribution of inferred or simulated phylogenies and character states in a single
170 NeXML file.

171 *Data documentation and annotation with built-in metadata*

172 NeXML allows attaching (“*annotating*”) metadata to any data element, and even to metadata themselves.
173 Whether at the level of the document as a whole or an individual data matrix or phylogeny, metadata
174 can provide bibliographic and provenance information, for example about the study as part of which
175 the phylogeny was generated or applied, which data matrix and which methods were used to generate it.
176 Metadata can also be attached to very specific elements of the data, such as specific traits, individual
177 OTUs, nodes, or even edges of the phylogeny.

178 As described in Vos et al. (2012), to encode metadata annotations NeXML uses the “Resource
179 Description Framework in Annotations” (RDFa) (Prud’hommeaux 2014). This standard provides for
180 a strict machine-readable format yet enables future backwards compatibility with compliant NeXML
181 parsers (and thus RNeXML), because the capacity of a tool to *parse* annotations is not predicated on
182 *understanding* the meaning of annotations it has not seen before.

183 To lower the barriers to sharing well-documented phylogenetic data, RNeXML aims to make recording
184 useful and machine-readable metadata easier at several levels.

185 First, when writing a NeXML file the package adds certain basic metadata automatically if they
186 are absent, using default values consistent with recommended best practices (Cranston *et al.* 2014).
187 Currently, this includes naming the software generating the NeXML, a time-stamp of when a tree was
188 produced, and an open data license. These are merely default arguments to `add_basic_meta()` and
189 can be configured.

190 Second, RNeXML provides a simple method, called `add_basic_metadata()`, to set metadata attributes
191 commonly recommended for inclusion with data to be publicly archived or shared (Cranston *et al.*
192 2014). The currently accepted parameters include `title`, `description`, `creator`, `pubdate`, `rights`,
193 `publisher`, and `citation`. Behind the scenes the method automatically anchors these attributes in
194 common vocabularies (such as Dublin Core).

195 Third, RNeXML integrates with the R package `taxize` (Chamberlain & Szöcs 2013) to mitigate one of
196 the most common obstacles to reuse of phylogenetic data, namely the misspellings and inconsistent
197 taxonomic naming with which OTU labels are often fraught. The `taxize_nexml()` method in RNeXML
198 uses `taxize` to match OTU labels against the NCBI database, and, where a unique match is found, it
199 annotates the respective OTU with the matching NCBI identifier.

200 *Data annotation with custom metadata*

201 The RNeXML interface described above for built-in metadata allows users to create precise and semantically
202 rich annotations without confronting any of the complexity of namespaces and ontologies. Nevertheless,
203 advanced users may desire the explicit control over these semantic tools that takes full advantage of
204 the flexibility and extensibility of the NeXML specification (Parr *et al.* 2011; Vos *et al.* 2012). In this
205 section we detail how to accomplish these more complex uses in RNeXML.

206 Using a vocabulary or ontology terms rather than simple text strings to describe data is crucial for
207 allowing machines to not only parse but also interpret and potentially reason over their semantics.

208 To achieve this benefit for custom metadata extensions, the user necessarily needs to handle certain
209 technical details from which the RNeXML interface shields her otherwise, in particular the globally unique
210 identifiers (normally HTTP URIs) of metadata terms and vocabularies. To be consistent with XML
211 terminology, RNeXML calls vocabulary URIs *namespaces*, and their abbreviations *prefixes*. For example,
212 the namespace for the Dublin Core Metadata Terms vocabulary is “<http://purl.org/dc/elements/1.1/>”.
213 Using its common abbreviation “dc”, a metadata property “dc:title” expands to the identifier “<http://purl.org/dc/elements/1.1/title>”. This URI resolves to a human and machine-readable (depending on
215 access) definition of precisely what the term `title` in Dublin Core means. In contrast, just using the
216 text string “title” could also mean the title of a person, a legal title, the verb title, etc. URI identifiers
217 of metadata vocabularies and terms are not mandated to resolve, but if machines are to derive the
218 maximum benefit from them, they should resolve to a definition of their semantics in RDF.

219 RNeXML includes methods to obtain and manipulate metadata properties, values, identifiers, and
220 namespaces. The `get_namespaces()` method accepts a `nexml` object and returns a named list of
221 namespace prefixes and their corresponding identifiers known to the object:

```
birds <- nexml_read("birds.xml")
prefixes <- get_namespaces(birds)
prefixes["dc"]
```

```
222                                     dc
223 "http://purl.org/dc/elements/1.1/"
```

224 The `get_metadata()` method returns, as a named list, the metadata annotations for a given `nexml`
225 object at a given level, with the whole NeXML document being the default level (“all” extracts all
226 metadata objects):

```
meta <- get_metadata(birds)
otu_meta <- get_metadata(birds, level="otu")
```

227 The returned list does not include the data elements to which the metadata are attached. Therefore, a
228 different approach, documented in the metadata vignette, is recommended for accessing the metadata
229 attached to data elements.

230 The `meta()` method creates a new metadata object from a property name and content (value). For
231 example, the following creates a modification date metadata object, using a property in the PRISM
232 vocabulary:

```
modified <- meta(property = "prism:modificationDate", content = "2013-10-04")
```

233 Metadata annotations in NeXML can be nested within another annotation, which the `meta()` method
234 accommodates by accepting a parameter `children`, with the list of nested metadata objects (which can
235 themselves be nested) as value.

236 The `add_meta()` function adds metadata objects as annotations to a `nexml` object at a specified level,
237 with the default level being the NeXML document as a whole:

```
birds <- add_meta(modified, birds)
```

238 If the prefix used by the metadata property is not among the built-in ones (which can be obtained
239 using `get_namespaces()`), it has to be provided along with its URI as the `namespaces` parameter. For
240 example, the following uses the “[Simple Knowledge Organization System](http://www.w3.org/2004/02/skos/core#)” (SKOS) vocabulary to add a
241 note to the trees in the `nexml` object:

```
history <- meta(property = "skos:historyNote",  
  content = "Mapped from the bird.orders data in the ape package using RNeXML")  
birds <- add_meta(history,  
  birds,  
  level = "trees",  
  namespaces = c(skos = "http://www.w3.org/2004/02/skos/core#"))
```

242 Alternatively, additional namespaces can also be added in batch using the `add_namespaces()` method.
243 By virtue of subsetting the S4 `nexml` object, RNeXML also offers fine control of where a `meta` element is
244 added, for which the package vignette on S4 subsetting of `nexml` contains examples.

245 Because NeXML expresses all metadata using the RDF standard, and stores them compliant with
246 RDFa, they can be extracted as an RDF graph, queried, analyzed, and mashed up with other RDF data,
247 local or on the web, using a wealth of off-the-shelf tools for working with RDF (see Prud’hommeaux
248 (2014) or Hartig (2012)). Examples for these possibilities are included in the RNeXML SPARQL vignette

(a recursive acronym for SPARQL Protocol and RDF Query Language, see <http://www.w3.org/TR/rdf-sparql-query/>), and the package also comes with a demonstration that can be run from R using the following command: `demo("sparql", "RNeXML")`).

Using metadata to extend the NeXML standard

NeXML was designed to prevent the need for future non-interoperable “flavors” of the standard in response to new research directions. Its solution to this inevitable problem is a highly flexible metadata system without sacrificing strict validation of syntax and structure.

Here we illustrate how RNeXML’s interface to NeXML’s metadata system can be used to record and share a type of phylogenetic data not taken into account when NeXML was designed, in this case stochastic character maps (Huelsenbeck *et al.* 2003). Such data assign certain parts (corresponding to time) of each branch in a time-calibrated phylogeny to a particular “state” (typically of a morphological characteristic). The current de-facto format for sharing stochastic character maps, created by `simmap` (Bollback 2006), a widely used tool for creating such maps, is a non-interoperable modification of the standard Newick tree format. This means that computer programs designed to read Newick or NEXUS formats may fail when trying to read in a phylogeny that includes `simmap` annotations.

In contrast, by allowing new data types to be added as — sometimes complex — metadata annotations NeXML can accommodate data extensions without compromise to its grammar and thus syntax. In NeXML. To illustrate how RNeXML facilitates extending the NeXML standard in this way, we have implemented two functions in the package, `nexml_to_simmap` and `simmap_to_nexml`. These functions show how `simmap` data can be represented as `meta` annotations on the branch length elements of a NeXML tree, and provide routines to convert between this NeXML representation and the extended `ape::phylo` representation of a `simmap` tree in R that was introduced by Revell (2012). We encourage readers interested in this capability to consult the example code in `simmap_to_nexml` to see how this is implemented.

Extensions to NeXML must also be defined in the file’s namespace in order to valid. This provides a way to ensure that a URI providing documentation of the extension is always included. Our examples here use the prefix, `simmap`, to group the newly introduced metadata properties in a vocabulary, for which the `add_namespace()` method can be used to give a URI as an identifier:

```
nex <- add_namespaces(c(simmap =  
  "https://github.com/ropensci/RNeXML/tree/master/inst/simmap.md"))
```

Here the URI does not resolve to a fully machine-readable definition of the terms and their semantics, but it can nonetheless be used to provide at least a human-readable informal definition of the terms.

Publishing NeXML files from R

Data archiving is increasingly required by scientific journals, including in evolutionary biology, ecology, and biodiversity (e.g. Rausher et al. (2010)). The effort involved with preparing and submitting properly annotated data to archives remains a notable barrier to the broad adoption of data archiving and sharing as a normal part of the scholarly publication workflow (Tenopir *et al.* 2011; Stodden 2014). In particular, the majority of phylogenetic trees published in the scholarly record are inaccessible or lost to the research community (Drew *et al.* 2013).

One of RNeXML's aims is to promote the archival of well-documented phylogenetic data in scientific data repositories, in the form of NeXML files. To this end, the method `nexml_publish()` provides an API directly from within R that allows data archival to become a step programmed into data management scripts. Initially, the method supports the data repository Figshare (<http://figshare.com>):

```
doi <- nexml_publish(birds, repository="figshare")
```

This method reserves a permanent identifier (DOI) on the figshare repository that can later be made public through the figshare web interface. This also acts as a secure backup of the data to a repository and a way to share with collaborators prior to public release.

Conclusions and future directions

RNeXML allows R's ecosystem to read and write data in the NeXML format through an interface that is no more involved than reading or writing data from other phylogenetic data formats. It also carries immediate benefits for its users compared to other formats. For example, comparative analysis R packages and users frequently add their own metadata annotations to the phylogenies they work with, such as annotations of species, stochastic character maps, trait values, model estimates and parameter values. RNeXML affords R the capability to harness machine-readable semantics and an extensible

300 metadata schema to capture, preserve, and share these and other kinds of information, all through an
301 API instead of having to understand in detail the schema underlying the NeXML standard. To assist
302 users in meeting the rising bar for best practices in data sharing in phylogenetic research (Cranston *et*
303 *al.* 2014), RNeXML captures metadata information from the R environment to the extent possible, and
304 applies reasonable defaults.

305 The goals for continued development of RNeXML revolve primarily around better interoperability with
306 other existing phylogenetic data representations in R, such as those found in the **phylobase** package
307 (NESCENT R Hackathon Team 2014); and better integration of the rich metadata semantics found in
308 ontologies defined in the Web Ontology Language (OWL), including programmatic access to machine
309 reasoning with such metadata.

310 *Acknowledgements*

311 This project was supported in part by the National Evolutionary Synthesis Center (NESCent) (NSF
312 #EF-0905606), and grants from the National Science Foundation (DBI-1306697) and the Alfred P Sloan
313 Foundation (Grant 2013-6-22). RNeXML started as a project idea for the Google Summer of Code(TM),
314 and we thank Kseniia Shumelchuk for taking the first steps to implement it. We are grateful to F.
315 Michonneau for helpful comments on an earlier version of this manuscript.

316 **References**

- 317 Bollback, J. (2006). *BMC Bioinformatics*, **7**, 88. Retrieved from [http://dx.doi.org/10.1186/1471-2105-7-](http://dx.doi.org/10.1186/1471-2105-7-88)
318 [88](http://dx.doi.org/10.1186/1471-2105-7-88)
- 319 Chamberlain, S.A. & Szöcs, E. (2013). Taxize: Taxonomic search and retrieval in r. *F1000Research*.
320 Retrieved from <http://dx.doi.org/10.12688/f1000research.2-191.v2>
- 321 Cranston, K., Harmon, L.J., O’Leary, M.A. & Lisle, C. (2014). Best practices for data shar-
322 ing in phylogenetic research. *PLoS Curr.* Retrieved from [http://dx.doi.org/10.1371/currents.tol.](http://dx.doi.org/10.1371/currents.tol.bf01eff4a6b60ca4825c69293dc59645)
323 [bf01eff4a6b60ca4825c69293dc59645](http://dx.doi.org/10.1371/currents.tol.bf01eff4a6b60ca4825c69293dc59645)
- 324 Drew, B.T., Gazis, R., Cabezas, P., Swithers, K.S., Deng, J., Rodriguez, R., Katz, L.A., Crandall,
325 K.A., Hibbett, D.S. & Soltis, D.E. (2013). Lost branches on the tree of life. *PLoS Biol*, **11**, e1001636.
326 Retrieved from <http://dx.doi.org/10.1371/journal.pbio.1001636>

327 Hartig, O. (2012). An introduction to sPARQL and queries over linked data. *Web engineering* pp.
 328 506–507. Springer Science + Business Media. Retrieved from [http://dx.doi.org/10.1007/978-3-642-](http://dx.doi.org/10.1007/978-3-642-31753-8_56)
 329 [31753-8_56](http://dx.doi.org/10.1007/978-3-642-31753-8_56)

330 Huelsenbeck, J.P., Nielsen, R. & Bollback, J.P. (2003). Stochastic mapping of morphological characters.
 331 *Systematic Biology*, **52**, 131–158. Retrieved from <http://dx.doi.org/10.1080/10635150390192780>

332 Lang, D.T. (2013). *XML: Tools for parsing and generating xML within r and s-plus*. Retrieved from
 333 <http://CRAN.R-project.org/package=XML>

334 Maddison, D., Swofford, D. & Maddison, W. (1997). NEXUS: An extensible file format for systematic
 335 information. *Syst. Biol.*, **46**, 590–621. Retrieved from <http://www.ncbi.nlm.nih.gov/pubmed/11975335>

336 Midford, P., Dececchi, T., Balhoff, J., Dahdul, W., Ibrahim, N., Lapp, H., Lundberg, J., Mabee, P.,
 337 Sereno, P., Westerfield, M., Vision, T. & Blackburn, D. (2013). The vertebrate taxonomy ontology: A
 338 framework for reasoning across model organism and species phenotypes. *J. Biomed. Semantics*, **4**, 34.
 339 Retrieved from <http://dx.doi.org/10.1186/2041-1480-4-34>

340 NESCENT R Hackathon Team. (2014). *Phylobase: Base package for phylogenetic structures and*
 341 *comparative data*. Retrieved from <http://CRAN.R-project.org/package=phylobase>

342 O’Meara, B. (2014). CRAN task view: Phylogenetics, especially comparative methods. Retrieved from
 343 <http://cran.r-project.org/web/views/Phylogenetics.html>

344 Paradis, E., Claude, J. & Strimmer, K. (2004). APE: Analyses of phylogenetics and evolution in R
 345 language. *Bioinformatics*, **20**, 289–290.

346 Parr, C.S., Guralnick, R., Cellinese, N. & Page, R.D.M. (2011). Evolutionary informatics: unifying
 347 knowledge about the diversity of life. *Trends in ecology & evolution*, **27**, 94–103. Retrieved from
 348 <http://www.ncbi.nlm.nih.gov/pubmed/22154516>

349 Pennell, M.W., Eastman, J.M., Slater, G.J., Brown, J.W., Uyeda, J.C., Fitzjohn, R.G., Alfaro, M.E. &
 350 Harmon, L.J. (2014). Geiger v2.0: An expanded suite of methods for fitting macroevolutionary models
 351 to phylogenetic trees. *Bioinformatics*, **30**, 2216–2218.

352 Piel, W.H., Chan, L., Dominus, M.J., Ruan, J., Vos, R.A. & Tannen, V. (2009). TreeBASE v. 2: A
 353 database of phylogenetic knowledge. Retrieved from <http://www.e-biosphere09.org>

354 Piel, W.H., Donoghue, M.J. & Sanderson, M.J. (2002). TreeBASE: A database of phylogenetic
 355 information. *The interoperable ‘catalog of life’* (eds J. Shimura, K.L. Wilson & D. Gordon), pp. 41–47.

356 Research report. National Institute for Environmental Studies, Tsukuba, Japan. Retrieved from
 357 http://donoghuelab.yale.edu/sites/default/files/124_piel_shimura02.pdf
 358 Prud'hommeaux, E. (2014). SPARQL query language for rDF. *W3C*. Retrieved from <http://www.w3.org/TR/rdf-sparql-query/>
 359
 360 R Core Team. (2014). *R: A language and environment for statistical computing*. R Foundation for
 361 Statistical Computing, Vienna, Austria. Retrieved from <http://www.R-project.org/>
 362 Rausher, M.D., McPeck, M.A., Moore, A.J., Rieseberg, L. & Whitlock, M.C. (2010). Data archiving.
 363 *Evolution*, **64**, 603–604. Retrieved from <http://dx.doi.org/10.1111/j.1558-5646.2009.00940.x>
 364 Revell, L.J. (2012). Phytools: An r package for phylogenetic comparative biology (and other things).
 365 *Methods in Ecology and Evolution*, **3**, 217–223.
 366 Stodden, V. (2014). The scientific method in practice: Reproducibility in the computational sciences.
 367 *SSRN Journal*. Retrieved from <http://dx.doi.org/10.2139/ssrn.1550193>
 368 Stoltzfus, A., O'Meara, B., Whitacre, J., Mounce, R., Gillespie, E.L., Kumar, S., Rosauer, D.F. & Vos,
 369 R.A. (2012). Sharing and re-use of phylogenetic trees (and associated data) to facilitate synthesis. *BMC*
 370 *Research Notes*, **5**, 574. Retrieved from <http://dx.doi.org/10.1186/1756-0500-5-574>
 371 Tenopir, C., Allard, S., Douglass, K., Aydinoglu, A.U., Wu, L., Read, E., Manoff, M. & Frame, M.
 372 (2011). Data sharing by scientists: Practices and perceptions (C. Neylon, Ed.). *PLoS ONE*, **6**, e21101.
 373 Retrieved from <http://dx.doi.org/10.1371/journal.pone.0021101>
 374 Vos, R.A., Balhoff, J.P., Caravas, J.A., Holder, M.T., Lapp, H., Maddison, W.P., Midford, P.E.,
 375 Priyam, A., Sukumaran, J., Xia, X. & Stoltzfus, A. (2012). NeXML: Rich, extensible, and verifiable
 376 representation of comparative data and metadata. *Systematic Biology*, **61**, 675–689. Retrieved from
 377 <http://dx.doi.org/10.1093/sysbio/sys025>