Template to prepare preprints and manuscripts using markdown and github actions

Timothée Poisot [1,2,‡], Peregrin Took [3,4], Merriadoc Brandybuck [5,4,‡]

[1] Université de Montréal; [2] Québec Centre for Biodiversity Sciences; [3] Inn of the Prancing Pony; [4] Fellowship of the Ring; [5] Green Dragon Inn
[‡] These authors contributed equally to the work

Correspondance to:
Timothée Poisot — timothee.poisot@umontreal.ca

Purpose: This template provides a series of scripts to render a markdown document into an interactive website and a series of PDFs.
Motivation: It makes collaborating on text with GitHub easier, and means that we never need to think about the output.
Internals: GitHub actions and a series of python scritpts. The markdown is handled with pandoc.

Keywords:
pandoc
pandoc-crossref
github actions

NCBITaxonomy.jl is a package designed to facilitate the reconciliation and cleaning of taxonomic names, using a local copy of the NCBI taxonomic backbone (**Federhen2012NcbTax?**; **Schoch2020NcbTax?**); The basic search functions are coupled with quality-of-life functions including case-insensitive search and custom fuzzy string matching to facilitate the amount of information that can be extracted automatically while allowing efficient manual curation and inspection of results. NCBITaxonomy.jl works with version 1.6 of the Julia programming language (**Bezanson2017JulFre?**), and relies on the Apache Arrow format to store a local copy of the NCBI raw taxonomy files. The design of NCBITaxonomy.jl has been inspired by similar efforts, like the R package taxadb (**Norman2020TaxHig?**), which provides an offline alternative to packages like taxize (**Chamberlain2013TaxTax?**).

1 Statement of need

Unambiguously identifying species is a far more challenging task than it may appear. There are a vast number of reasons for this. Different databases keep different taxonomic "backbones," *i.e.* different data structures in which names are mapped to species, and organised in a hierarchy. Not all names are unique identifiers to groups. For example, *Io* can either refer to a genus of plants from the aster family, or to a genus of molluscs; the genus *Mus* (of which the house mouse *Mus musculus* is a species), contains a sub-genus *also* named *Mus*. Conversely, the same species can have several names, which are valid synonyms: for example, the domestic cow *Bos taurus* admits *Bos primigenius taurus* as a valid synonym. Taxonomic nomenclature also changes regularly, with groups being split, merged, or moved to a new position in the tree of life; this is, notably, a common occurrence with viral taxonomy, each subsequent version of which can differ markedly from the last; compare, *e.g* (**Lefkowitz2018VirTax?**) to (**Walker2020ChaVir?**).

To add to the complexity, one must also consider that most taxa names are at some point manually typed, which has the potential to introduce additional mistakes in raw data; it is likely to expect that such mistakes may arise when attempting to write down the (perfectly valid) names of the bacterial isolate known

as *Myxococcus llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogochensis*, or of the crowned slaty flycatcher *Griseotyrannus aurantioatrocristatus*. These mistakes are more likely when dealing with hyper-diverse samples, like plant census (**Dauncey2016ComMis?**; **Wagner2016RevSof?**; **Conti2021MatAlg?**). In addition to binomial names, the same species can be known by many vernacular (common) names, which are language or even region-specific: *Ovis aries*, for example, has valid English vernaculars including lamb, sheep, wild sheep, and domestic sheep.

All these considerations are actually important when matching species names both within and across datasets. Let us consider the following species survey of individual fishes, European chub, *Cyprinus cephalus*, *Leuciscus cephalus*, *Squalius cephalus*: all are the same species (*S. cephalus*), referred to as one of the vernacular (European chub) and two formerly accepted names now classified as synonyms. A cautious estimate of diversity based on the user-supplied names would give $n = 4$ species, when there is in fact only one.

A package with the ability to handle the sources of errors outlined above, and especially while provide an authoritative classification, can accelerate the work of consuming large volumes of biodiversity data. For example, this package was used in the process of developing the *CLOVER* database (**Gibb2021DatPro?**) of host-virus associations, by reconciling the names of viruses and mammals from four different sources, where all of the issues described above were present.

2

Overview of functionalities

An up-to-date version of the documentation for `NCBITaxonomy.jl` can be found online at https://docs.ecojulia.org/NCBITaxonomy.jl/stable/, including examples and a documentation of every method. The package is released under the MIT license. Contributions can be made in the form of issues (bug reports, questions, features suggestions) and pull requests. The package can be checked out and installed anonymously from the central Julia repository:

```
using Pkg

# This line should go in the Julia configuration file - note that the path
# will be created if it doesn't exist, and will be used to store the
# raw taxonomic table
ENV["NCBITAXONOMY_PATH"] = joinpath(homedir(), "data", "NCBITaxonomy.jl")

Pkg.add("NCBITaxonomy") # Dowloading the files may take a long time
```

The package will download the most recent version of the NCBI taxonomy database, and transform in into a set of Apache Arrow files ready for use. Note that the `NCBITAXONOMY_PATH` can specified on a per-project basis, and as long as the package is not re-built, the local set of tables downloaded from NCBI will not change; this way, users can re-run an analysis with a guarantee that the underlying taxonomic backbone has not changed.

2.1. Improved name matching Name finding is primarily done through the `taxon` function, which admits either a unique NCBI identifier (*e.g.* `taxon(36219)` for the bogue *Boops boops*), a string (`taxon("Boops boops")`), or a data frame with a restricted list of names (see the next section). The `taxon` method has additional arguments to perform fuzzy matching in order to catch possible typos (`taxon("Boops bops"; strict=false)`), to perform a lowercase search (useful when alphanumeric codes are part of the taxon name, like for some viruses), and to restrict the the search to a specific taxonomic rank.

The lowercase search can be a preferable alternative to fuzzy string matching. Consider the string `Adeno-associated virus 3b` - it has three names with equal distance (under the Levensthein string distance function):

```
julia> similarnames("Adeno-associated virus 3b"; threshold=0.95)
3-element Vector{Pair{NCBITaxon, Float64}}:
  Adeno-associated virus - 3 (ncbi:46350) => 0.96
   Adeno-associated virus 3B (ncbi:68742) => 0.96
 Adeno-associated virus 3A (ncbi:1406223) => 0.96
```

Depending on the operating system, either of these three names can be returned; compare to the output of a case insensitive name search:

```
julia> taxon("Adeno-associated virus 3b"; casesensitive=false)
Adeno-associated virus 3B (ncbi:68742)
```

This returns the correct name.

2.2. Name matching output and error handling The `taxon` function will either return a `NCBITaxon` object (made of a `name` and `id`), or throw either a `NameHasNoDirectMatch` (with instructions about how to possible solve it, using the `similarnames` function), or a `NameHasMultipleMatches` (listing the possible valid matches, and suggesting to use `alternativetaxa` to find the correct one). Therefore, the common way to work with the `taxon` function would be to wrap it in a `try`/`catch` statement:

```
try
  taxon(name)
  # Additional operations with the matched name
catch err
  if isa(err, NameHasNoDirectMatch)
    # What to do if no match is found
  elseif isa(err, NameHasMultipleMatches)
    # What to do if there are multiple matches
  else
    # What to do in case of another error that is not NCBITaxonomy specific
  end
end
```

These functions will not demand any user input in the form of key presses (though they can be wrapped in additional code to allow it), as they are intended to run on clusters without supervision. The `taxon` function has good scaling using muliple threads. For convenience in rapidly getting a taxon for demonstration purposes, we also provide a string macro, whereby *e.g.* `ncbi"Procyon lotor"` will return the taxon object for the raccoon.

2.3. Name filtering functions As the full NCBI names table has over 3 million entries at the time of writing, we have provided a number of functions to restrict the scope of names that are searched. These are driven by the NCBI *divisions*. For example `nf = mammalfilter(true)` will return a data frame containing the names of mammals, inclusive of rodents and primates, and can be used with *e.g.* `taxon(nf, "Pan")`. This has the dual advantage of making search faster, but also of avoiding matching on names that are shared by another taxonomic group (which is not an issue with *Pan*, but is an issue with *e.g. Io* as mentioned in the introduction).

Note that the use of a restricted list of names can have significant performance consequences: compare, for example, the time taken to return the taxon *Pan* (ID 9596) in the entire database, in all mammals, and in all primates:

Names list	Fuzzy matching	Time (ms)	Allocations	Memory allocated
all	no	23	34	2 KiB
	yes	105	2580	25 MiB
`mammalfilter(true)`	no	0.55	32	2 KiB
	yes	1.9	551	286 KiB
`primatefilter()`	no	0.15	33	2 KiB
	yes	0.3	92	27 KiB

Clearly, the optimal search strategy is to (i) rely on name filters to ensure that search are conducted within the appropriate NCBI division, and (ii) only rely on fuzzy matching when the strict or lowercase match fails to return a name, as fuzzy matching can result in order of magnitude more run time and memory footprint. These numbers were obtained on a single Intel i7-8665U CPU (@ (1.90GHz).

Using `"chimpanzees"` as the search string (the NCBI recognized vernacular for *Pan*) gave qualitatively similar results, suggesting that there is no performance cost associated with working with synonyms or verncular input data.

2.4. Quality of life functions In order to facilitate working with names, we provide the `authority` function (gives the full taxonomic authority for a name), `synonyms` (to get alternative valid names), `vernacular` (for English common names), and `rank` (for the taxonomic rank).

2.5. Taxonomic lineages navigation The `children` function will return all nodes that are directly descended from a taxon; the `descendants` function will recursively apply this function to all descendants of these nodes, until only terminal leaves are reached. The `parent` function is an "upwards" equivalent, giving that taxon from which a taxon descents; the `lineage` function chains calls to `parent` until either `taxon(1)` (the taxonomy root) or an arbitrary ancestor is reached.

The `taxonomicdistance` function (and its in-place equivalent, `taxonomicdistance!`, which uses memory-efficient re-allocation if the user needs to change the distance between taxonomic ranks) uses the (**Shimatani2001MeaSpe?**) approach to reconstruct a matrix of distances based on taxonomy, which can serve as a rough proxy when no phylogenies are available.

Acknowledgements: This work was supported by funding to the Viral Emergence Research Initiative (VERENA) consortium including NSF BII 2021909 and a grant from Institut de Valorisation des Données (IVADO), by the NSERC Discovery Grants and Discovery Acceleration Supplement programs, and by a donation from the Courtois Foundation. Benchmarking of this package on distributed systems was enabled by support provided by Calcul Québec (`www.calculquebec.ca`) and Compute Canada (`www.computecanada.ca`). TP wrote the initial code, TP and CJC contributed to API design, and all authors contributed to functionalities and usability testing.

3

References