Package: phylotaR (via r-universe)

December 16, 2024

Type Package

Title Automated Phylogenetic Sequence Cluster Identification from 'GenBank'

Version 1.3.0

Description A pipeline for the identification, within taxonomic groups, of orthologous sequence clusters from 'GenBank' <https://www.ncbi.nlm.nih.gov/genbank/> as the first step in a phylogenetic analysis. The pipeline depends on a local alignment search tool and is, therefore, not dependent on differences in gene naming conventions and naming errors.

License MIT + file LICENSE

URL https://docs.ropensci.org/phylotaR/, https://github.com/ropensci/phylotaR#readme

BugReports https://github.com/ropensci/phylotaR/issues

Depends methods, R (>= 3.5.0)

Imports ape, bigmemory, ggplot2, igraph, plyr, R.utils, rentrez, restez (>= 2.1.0), RJSONIO, stringr, sys, treemapify, XML

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.3

SystemRequirements BLAST+ (>=2.0)

X-schema.org-isPartOf https://ropensci.org

Config/pak/sysreqs libglpk-dev make libicu-dev libjpeg-dev libpng-dev libxml2-dev libssl-dev xz-utils

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/phylotaR

RemoteRef master

RemoteSha 0495c37ed0a21878193ee1080a108d9ce8cbf74d

Contents

`addClade` *Add clade to tree*

Description

Returns a tree with added clade

Usage

```
addClade(tree, id, clade)
```

Arguments

`tree`	`TreeMan` object
`id`	tip/node ID in tree to which the clade will be added
`clade`	`TreeMan` object

Details

Add a `TreeMan` object to an existing `TreeMan` object by specifying an ID at which to attach. If the id specified is an internal node, then the original clade descending from that node will be replaced. Before running, ensure no IDs are shared between the `tree` and the `clade`, except for the IDs in the clade of that tree that will be replaced. Note, returned tree will not have a node matrix.

See Also

`rmClade`, `getSubtree`, `https://github.com/DomBennett/treeman/wiki/manip-methods`

Examples

```
t1 <- randTree(100)
# extract a clade
cld <- getSubtree(t1, "n2")
# remove the same clade
t2 <- rmClade(t1, "n2")
# add the clade again
t3 <- addClade(t2, "n2", cld)
# t1 and t3 should be the same
# note there is no need to remove a clade before adding
t3 <- addClade(t1, "n2", cld) # same tree
```

`addNdmtrx` *Add node matrix to a tree*

Description

Return tree with node matrix added.

Usage

```
addNdmtrx(tree, shared = FALSE, ...)
```

Arguments

`tree`	`TreeMan` object
`shared`	T/F, should the bigmatrix be shared? See bigmemory documentation.
`...`	`as.big.matrix()` additional arguments

Details

The node matrix makes 'enquiry'-type computations faster: determining node ages, number of descendants etc. But it takes up large amounts of memory and has no impact on adding or removing tips. Note, trees with the node matrix can not be written to disk using the 'serialization format' i.e. with save or `saveRDS`. The matrix is generated with bigmemory's ‘as.big.matrix()‘.

See Also

`updateSlts`, `rmNdmtrx`, `https://cran.r-project.org/package=bigmemory`

Examples

```
#
tree <- randTree(10, wndmtrx = FALSE)
summary(tree)
tree <- addNdmtrx(tree)
summary(tree)
```

`addTip` *Add tip to a tree*

Description

Returns a tree with a new tip ID added

Usage

```
addTip(
  tree,
  tid,
  sid,
  strt_age = NULL,
  end_age = 0,
  tree_age = NULL,
  pid = paste0("p_", tid)
)
```

Arguments

`tree`	`TreeMan` object
`tid`	tip ID
`sid`	ID of node that will become new tip sisters
`strt_age`	timepoint at which new tips first appear in the tree
`end_age`	timepoint at which new tips end appear in the tree, default 0.
`tree_age`	age of tree
`pid`	parent ID (default is 'p_' + tid)

Details

User must provide new tip ID, the ID of the node which will become the new tip's sister, and new branch lengths. The tip ID must only contain letters numbers and underscores. Optionally, user can specify the IDs for the new parental internal nodes. Ensure that the `strt_age` is greater than the end_age, and that the `strt_age` falls within the age span of the sister ID. Otherwise, negative spns may be produced leading to an error. Note, returned tree will not have a node matrix. Note, providing negative end ages will increase the age of the tree.

See Also

rmTips, https://github.com/DomBennett/treeman/wiki/manip-methods

Examples

```
tree <- randTree(10)
tree_age <- getAge(tree)
possible_ages <- getSpnAge(tree, "t1", tree_age)
start_age <- runif(1, possible_ages[["end"]], possible_ages[["start"]])
end_age <- possible_ages[["end"]]
tree <- addTip(tree,
  tid = "t11", sid = "t1", strt_age = start_age,
  end_age = end_age, tree_age = tree_age
)
summary(tree)
```

`aotus` *aotus*

Description

aotus

Format

A TreeMan or Phylota object

Examples

```
data("aotus")
```

`batcher` *Download in batches*

Description

Run downloader function in batches for sequences or taxonomic records

Usage

```
batcher(ids, func, ps, lvl = 0)
```

Arguments

`ids`	Vector of record ids
`func`	Downloader function
`ps`	Parameters list, generated with parameters()
`lvl`	Integer, number of message indentations indicating code depth.

Value

Vector of records

vector of rentrez function results

See Also

Other run-private: `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`birds` *birds*

Description

birds

Format

A TreeMan or Phylota object

Examples

```
data("birds")
```

`blastcache_load` *Load BLAST results from cache*

Description

Run to load cached BLAST results.

Usage

```
blastcache_load(sids, wd)
```

Arguments

`sids`	Sequence IDs
`wd`	Working dir

Value

blast_res data.frame or NULL

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`blastcache_save` *Save BLAST results to cache*

Description

Run whenever local BLAST runs are made to save results in cache in case the pipeline is run again.

Usage

```
blastcache_save(sids, wd, obj)
```

Arguments

`sids`	Sequence IDs
`wd`	Working dir
`obj`	BLAST result

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`blastdb_gen` *Generate a BLAST database*

Description

Generate BLAST database in wd for given sequences.

Usage

```
blastdb_gen(sqs, dbfl, ps)
```

Arguments

`sqs`	Sequences
`dbfl`	Outfile for database
`ps`	Parameters list, generated with parameters()

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`blastn_run` *Launch blastn*

Description

Use `blastn` to BLAST all-vs-all using a BLAST database.

Usage

```
blastn_run(dbfl, outfl, ps)
```

Arguments

`dbfl`	Database file
`outfl`	Output file
`ps`	Parameters list, generated with parameters()

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`blast_clstr` *Cluster BLAST Results*

Description

Find single-linkage clusters from BLAST results. Identifies seed sequence.

Usage

```
blast_clstr(blast_res)
```

Arguments

`blast_res`	BLAST results

Value

List of list

list of cluster descriptions

See Also

Other run-private: `batcher()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`blast_filter` *Filter BLAST results*

Description

Given a BLAST output, filters query-subject pairs such that only HSPs with a coverage greater than `mncvrg` (specified in the pipeline parameters) remain. Filters both: query-subject and subject-query pairs, if one of the coverages is insufficient. HSP coverage is obtained from the BLAST column `qcovs`.

Usage

```
blast_filter(blast_res, ps, lvl = 3)
```

Arguments

`blast_res`	BLAST results
`ps`	Parameters list, generated with parameters()
`lvl`	Integer, number of message indentations indicating code depth.

Value

data.frame blast res

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`blast_setup` *Ensures NCBI BLAST tools are installed*

Description

Ensures NCBI BLAST executables are installed on the system. Tests version number of BLAST tools.

Usage

```
blast_setup(d, v, wd, otsdr)
```

Arguments

`d`	Directory to NCBI BLAST tools
`v`	v, T/F
`wd`	Working directory
`otsdr`	Run through `outsider`?

Details

BLAST tools must be version >= 2.0

Value

list

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`blast_sqs` *BLAST All vs All*

Description

Return BLAST results from BLASTing all vs all for given sequences. Returns NULL if no BLAST results generated.

Usage

```
blast_sqs(txid, typ, sqs, ps, lvl)
```

Arguments

`txid`	Taxonomic node ID, numeric
`typ`	Cluster type, 'direct' or 'subtree'
`sqs`	Sequences
`ps`	Parameters list, generated with parameters()
`lvl`	Integer, number of message indentations indicating code depth.

Value

blast_res data.frame or NULL

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, obj_save(), outfmt_get(), parameters_load(), parameters_setup(), parent_get(), progress_init(), progress_read(), progress_reset(), progress_save(), rank_get(), rawseqrec_breakdown(), safely_connect(), search_and_cache(), searchterm_gen(), seeds_blast(), seq_download(), seqarc_gen(), seqrec_augment(), seqrec_convert(), seqrec_gen(), seqrec_get(), sids_check(), sids_get(), sids_load(), sids_save(), sqs_count(), sqs_save(), stage_args_check(), stages_run(), tax_download(), taxdict_gen(), taxtree_gen(), txids_get(), txnds_count(), warn()

`blncdTree` *Generate a balanced tree*

Description

Returns a balanced `TreeMan` tree with `n` tips.

Usage

```
blncdTree(n, wndmtrx = FALSE, parallel = FALSE)
```

Arguments

`n`	number of tips, integer, must be 3 or greater
`wndmtrx`	T/F add node matrix? Default FALSE.
`parallel`	T/F run in parallel? Default FALSE.

Details

Equivalent to ape's `stree(type='balanced')` but returns a `TreeMan` tree. Tree is always rooted and bifurcating.

See Also

TreeMan-class, randTree, unblncdTree

Examples

```
tree <- blncdTree(5)
```

`bromeliads` *bromeliads*

Description

bromeliads

Format

A TreeMan or Phylota object

Examples

```
data("bromeliads")
```

`cache_rm` *Delete a cache*

Description

Deletes a cache from a wd.

Usage

```
cache_rm(wd)
```

Arguments

`wd`	Working directory

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`cache_setup` *Set-up a cache*

Description

Creates a cache of parameters in the wd.

Usage

```
cache_setup(ps, ovrwrt = FALSE)
```

Arguments

`ps`	Parameters list, generated with parameters()
`ovrwrt`	Overwrite existing cache? Default FALSE.

Details

Warning: overwriting with this function will delete the existing cache.

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`calcDstBLD` *Calculate the BLD between two trees*

Description

Returns the branch length distance between two trees.

Usage

```
calcDstBLD(tree_1, tree_2, nrmlsd = FALSE, parallel = FALSE, progress = "none")
```

Arguments

`tree_1`	`TreeMan` object
`tree_2`	`TreeMan` object
`nrmlsd`	Boolean, should returned value be between 0 and 1? Default, FALSE.
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

BLD is the Robinson-Foulds distance weighted by branch length. Instead of summing the differences in partitions between the two trees, the metric takes the square root of the squared difference in branch lengths. Parallelizable.

References

Kuhner, M. K. and Felsenstein, J. (1994) Simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. Molecular Biology and Evolution, 11, 459-468.

See Also

`calcDstTrp`, `calcDstRF` `https://github.com/DomBennett/treeman/wiki/calc-methods`

Examples

```
tree_1 <- randTree(10)
tree_2 <- randTree(10)
calcDstBLD(tree_1, tree_2)
```

`calcDstMtrx` *Calculate the distance matrix*

Description

Returns a distance matrix for specified ids of a tree.

Usage

```
calcDstMtrx(tree, ids, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	IDs of nodes/tips
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

The distance between every id in the tree is calculated by summing the lengths of the branches that connect them. This can be useful for testing the distances between trees, checking for evoltuionary isolated tips etc. Parallelizable.

See Also

`calcDstBLD`, `calcDstRF`, `calcDstTrp` `https://github.com/DomBennett/treeman/wiki/calc-methods`

Examples

```
# checking the distance between two trees

tree_1 <- randTree(10)
tree_2 <- randTree(10)
dmat1 <- calcDstMtrx(tree_1, tree_1["tips"])
dmat2 <- calcDstMtrx(tree_2, tree_2["tips"])
mdl <- cor.test(x = dmat1, y = dmat2)
as.numeric(1 - mdl$estimate) # 1 - Pearson's r
```

`calcDstRF` *Calculate the Robinson-Foulds distance between two trees*

Description

Returns the Robinson-Foulds distance between two trees.

Usage

```
calcDstRF(tree_1, tree_2, nrmlsd = FALSE)
```

Arguments

`tree_1`	`TreeMan` object
`tree_2`	`TreeMan` object
`nrmlsd`	Boolean, should returned value be between 0 and 1? Default, FALSE.

Details

RF distance is calculated as the sum of partitions in one tree that are not shared by the other. The maximum number of split differences is the total number of nodes in both trees (excluding the roots). Trees are assumed to be bifurcating, this is not tested. The metric is calculated as if trees are unrooted. Parallelizable.

References

Robinson, D. R.; Foulds, L. R. (1981). "Comparison of phylogenetic trees". Mathematical Biosciences 53: 131-147.

See Also

calcDstBLD, calcDstTrp https://github.com/DomBennett/treeman/wiki/calc-methods

Examples

```
tree_1 <- randTree(10)
tree_2 <- randTree(10)
calcDstRF(tree_1, tree_2)
```

---

calcDstTrp *Calculate the triplet distance between two trees*

Description

Returns the triplet distance between two trees.

Usage

```
calcDstTrp(tree_1, tree_2, nrmlsd = FALSE, parallel = FALSE, progress = "none")
```

Arguments

`tree_1`	`TreeMan` object
`tree_2`	`TreeMan` object
`nrmlsd`	Boolean, should returned value be between 0 and 1? Default, FALSE.
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

The triplet distance is calculated as the sum of different outgroups among every triplet of tips between the two trees. Normalisation is performed by dividing the resulting number by the total number of triplets shared between the two trees. The triplet distance is calculated only for shared tips between the two trees. Parallelizable.

References

Critchlow DE, Pearl DK, Qian C. (1996) The Triples Distance for rooted bifurcating phylogenetic trees. Systematic Biology, 45, 323-34.

See Also

calcDstBLD, calcDstRF https://github.com/DomBennett/treeman/wiki/calc-methods

Examples

```
tree_1 <- randTree(10)
tree_2 <- randTree(10)
calcDstTrp(tree_1, tree_2)
```

`calcFrPrp` *Calculate evolutionary distinctness*

Description

Returns the evolutationary distinctness of ids using the fair proportion metric.

Usage

```
calcFrPrp(tree, tids, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`tids`	tip IDs
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

The fair proportion metric calculates the evolutionary distinctness of tips in a tree through summing the total amount of branch length each tip represents, where each branch in the tree is evenly divided between all descendants. Parallelizable.

References

Isaac, N.J.B., Turvey, S.T., Collen, B., Waterman, C. and Baillie, J.E.M. (2007). Mammals on the EDGE: conservation priorities based on threat and phylogeny. PLoS ONE, 2, e296.

See Also

`calcPhyDv`, `calcPrtFrPrp`, `https://github.com/DomBennett/treeman/wiki/calc-methods`

Examples

```
tree <- randTree(10)
calcFrPrp(tree, tree["tips"])
```

`calcNdBlnc` *Calculate the balance of a node*

Description

Returns the balance of a node.

Usage

```
calcNdBlnc(tree, id)
```

Arguments

`tree`	`TreeMan` object
`id`	node id

Details

Balance is calculated as the absolute difference between the number of descendents of the two bifurcating edges of a node and the expected value for a balanced tree. `NA` is returned if the node is polytomous or a tip.

See Also

`calcNdsBlnc`, `https://github.com/DomBennett/treeman/wiki/calc-methods`

Examples

```
tree <- randTree(10)
calcNdBlnc(tree, id = tree["root"]) # root balance
```

`calcNdsBlnc` *Calculate the balances of all nodes*

Description

Returns the absolute differences in number of descendants for bifurcating branches of every node

Usage

```
calcNdsBlnc(tree, ids, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	node ids
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Runs `calcNdBlnc()` across all node IDs. `NA` is returned if the node is polytomous. Parallelizable.

See Also

`calcNdBlnc`, `https://github.com/DomBennett/treeman/wiki/calc-methods`

Examples

```
tree <- randTree(10)
calcNdsBlnc(tree, ids = tree["nds"])
```

`calcOvrlp` *Calculate phylogenetic overlap*

Description

Returns the sum of branch lengths represented by ids_1 and ids_2 for a tree.

Usage

```
calcOvrlp(
  tree,
  ids_1,
  ids_2,
  nrmlsd = FALSE,
  parallel = FALSE,
  progress = "none"
)
```

Arguments

`tree`	`TreeMan` object
`ids_1`	tip ids of community 1
`ids_2`	tip ids of community 2
`nrmlsd`	Boolean, should returned value be between 0 and 1? Default, FALSE.
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Use this to calculate the sum of branch lengths that are represented between two communities. This measure is also known as the unique fraction. It can be used to measure concepts of phylogenetic turnover. Parallelizable.

References

Lozupone, C., & Knight, R. (2005). UniFrac: a new phylogenetic method for comparing microbial communities. Applied and Environmental Microbiology, 71(12), 8228-35.

See Also

`calcPhyDv` `https://github.com/DomBennett/treeman/wiki/calc-methods`

Examples

```
tree <- randTree(10)
ids_1 <- sample(tree["tips"], 5)
ids_2 <- sample(tree["tips"], 5)
calcOvrlp(tree, ids_1, ids_2)
```

`calcPhyDv` *Calculate phylogenetic diversity*

Description

Returns the phylogenetic diversity of a tree for the tips specified.

Usage

```
calcPhyDv(tree, tids, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`tids`	tip ids
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Faith's phylogenetic diversity is calculated as the sum of all connected branches for specified tips in a tree. It can be used to investigate how biodviersity as measured by the phylogeny changes. Parallelizable. The function uses `getCnntdNds()`.

References

Faith, D. (1992). Conservation evaluation and phylogenetic diversity. Biological Conservation, 61, 1-10.

See Also

`calcFrPrp`, `calcOvrlp`, `getCnnctdNds`, `https://github.com/DomBennett/treeman/wiki/calc-methods`

Examples

```
tree <- randTree(10)
calcPhyDv(tree, tree["tips"])
```

`calcPrtFrPrp` *Calculate evolutionary distinctness for part of tree*

Description

Returns the evolutationary distinctness of ids using the fair proportion metric.

Usage

```
calcPrtFrPrp(tree, tids, ignr = NULL, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`tids`	tip IDs
`ignr`	tips to ignore in calculation
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Extension of `calcFrPrp()` but with ignore argument. Use `ignr` to ignore certain tips from calculation. For example, if any of tips are extinct you may wish to ignore these.

References

Isaac, N.J.B., Turvey, S.T., Collen, B., Waterman, C. and Baillie, J.E.M. (2007). Mammals on the EDGE: conservation priorities based on threat and phylogeny. PLoS ONE, 2, e296.

See Also

`calcFrPrp` `https://github.com/DomBennett/treeman/wiki/calc-methods`

Examples

```
tree <- randTree(10)
calcPrtFrPrp(tree, c("t1", "t3"), ignr = "t2")
```

`calc_mad` *Calculate MAD score*

Description

For all sequences in a cluster(s) the MAD score.

Usage

```
calc_mad(phylota, cid)
```

Arguments

`phylota`	Phylota object
`cid`	Cluster ID(s)

Details

MAD is a measure of the deviation in sequence length of a cluster. Values range from 0 to 1. Clusters with values close to 1 have sequences with similar lengths.

Value

vector

See Also

Other tools-public: `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data("bromeliads")
random_cids <- sample(bromeliads@cids, 10)
(calc_mad(phylota = bromeliads, cid = random_cids))
```

`calc_wrdfrq` *Calculate word frequencies*

Description

For all sequences in a cluster(s) calculate the frequency of separate words in either the sequence definitions or the reported feature name.

Usage

```
calc_wrdfrq(
  phylota,
  cid,
  min_frq = 0.1,
  min_nchar = 1,
  type = c("dfln", "nm"),
  ignr_pttrn = "[^a-z0-9]"
)
```

Arguments

`phylota`	Phylota object
`cid`	Cluster ID(s)
`min_frq`	Minimum frequency
`min_nchar`	Minimum number of characters for a word
`type`	Definitions (dfln) or features (nm)
`ignr_pttrn`	Ignore pattern, REGEX for text to ignore.

Details

By default, anything that is not alphanumeric is ignored. 'dfln' and 'nm' match the slot names in a SeqRec, see list_seqrec_slots().

Value

list

See Also

Other tools-public: `calc_mad()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data('dragonflies')
# work out what gene region the cluster is likely representing with word freqs.
random_cids <- sample(dragonflies@cids, 10)
# most frequent words in definition line
(calc_wrdfrq(phylota = dragonflies, cid = random_cids, type = 'dfln'))
# most frequent words in feature name
(calc_wrdfrq(phylota = dragonflies, cid = random_cids, type = 'nm'))
```

`checkNdlst` *Check if ndlst is correct*

Description

Return T/F fpr `ndlst` consistency

Usage

```
checkNdlst(ndlst, root)
```

Arguments

`ndlst`	`ndlst`
`root`	root ID

Details

Tests whether each node in tree points to valid other node IDs. Also ensures ‘spn‘ and ‘root‘ are correct. Reports nodes that have errors.

See Also

`fastCheckTreeMan`, `checkTreeMen`

Examples

```
tree <- randTree(100)
(checkNdlst(tree@ndlst, tree@root))
```

`checkTreeMen` *Check if trees are correct*

Description

Return T/F if trees is a true `TreeMen` object

Usage

```
checkTreeMen(object)
```

Arguments

`object`	`TreeMen` object

Details

Tests whether all trees in object are `TreeMan` objects

See Also

`checkNdlst`

`clade_select` *Get all node IDs that will be processed*

Description

All nodes with less than maximum number of nodes and sequences.

Usage

```
clade_select(txdct, ps)
```

Arguments

`txdct`	TxDct
`ps`	Parameters list, generated with parameters()

Value

vector of txids

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstr2_calc` *Cluster sets of clusters identified in cluster stage*

Description

Loads cluster sets from cache. Extracts seed sequences and runs all-v-all BLAST of seeds to identify sister clusters. Sisters are then merged. An object of all sequences and clusters is then saved in cache.

Usage

```
clstr2_calc(ps)
```

Arguments

`ps`	Parameters list, generated with parameters()

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`ClstrArc-class` *Cluster record archive*

Description

Multiple cluster records.

Usage

```
## S4 method for signature 'ClstrArc'
as.character(x)

## S4 method for signature 'ClstrArc'
show(object)

## S4 method for signature 'ClstrArc'
print(x)

## S4 method for signature 'ClstrArc'
str(object, max.level = 2L, ...)

## S4 method for signature 'ClstrArc'
summary(object)

## S4 method for signature 'ClstrArc,character'
x[[i]]

## S4 method for signature 'ClstrArc,character,missing,missing'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	`ClstrArc` object
`object`	`ClstrArc` object
`max.level`	Maximum level of nesting for str()
`...`	Further arguments for str()
`i`	cid(s)
`j`	Unused
`drop`	Unused

Slots

`ids` Vector of cluster record IDs

`clstrs` List of ClstrArc named by ID

See Also

Other run-public: `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
data('aotus')
clstrarc <- aotus@clstrs
# this is a ClstrArc object
# it contains cluster records
show(clstrarc)
# you can access its different data slots with @
clstrarc@ids     # unique cluster ID
clstrarc@clstrs  # list of cluster records
# access cluster records [[
(clstrarc[[clstrarc@ids[[1]]]])  # first cluster record
# generate new cluster archives with [
(clstrarc[clstrarc@ids[1:10]])  # first 10 clusters
```

`clstrarc_gen` *Generate cluster archive container class*

Description

Takes a list of ClstrRecs, returns a ClstrArc.

Usage

```
clstrarc_gen(clstrrecs)
```

Arguments

`clstrrecs`	list of ClstrRecs

Value

ClstrArc

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstrarc_join` *Join two cluster archive*

Description

Take two ClstrArc classes and join them into a single ClstrArc.

Usage

```
clstrarc_join(clstrarc_1, clstrarc_2)
```

Arguments

`clstrarc_1`	ClstrArc
`clstrarc_2`	ClstrArc

Value

ClstrArc

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`ClstrRec-class` *Cluster record*

Description

Cluster record contains all information on a cluster.

Usage

```
## S4 method for signature 'ClstrRec'
as.character(x)

## S4 method for signature 'ClstrRec'
show(object)

## S4 method for signature 'ClstrRec'
print(x)

## S4 method for signature 'ClstrRec'
str(object, max.level = 2L, ...)

## S4 method for signature 'ClstrRec'
summary(object)
```

Arguments

`x`	`ClstrRec` object
`object`	`ClstrRec` object
`max.level`	Maximum level of nesting for str()
`...`	Further arguments for str()

Slots

`id` Cluster ID, integer

`sids` Sequence IDs

`nsqs` Number of sequences

`txids` Source txids for sequences

`ntx` Number of taxa

`typ` Cluster type: direct, subtree or merged

`seed` Seed sequence ID

`prnt` Parent taxonomic ID

See Also

Other run-public: `ClstrArc-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
data('aotus')
clstrrec <- aotus@clstrs@clstrs[[1]]
# this is a ClstrRec object
# it contains cluster information
show(clstrrec)
# you can access its different data slots with @
clstrrec@id     # cluster id
clstrrec@sids   # sequence IDs
clstrrec@nsqs   # number of sequences
clstrrec@txids  # taxonomic IDs of sequences
clstrrec@ntx    # number unique taxonomic IDs
clstrrec@typ    # cluster type: merged, subtree, direct or paraphyly
clstrrec@prnt   # MRCA of all taxa
clstrrec@seed   # most inter-connected sequence
```

`clstrrec_gen` *Generate list of clusters*

Description

Takes a list of lists of cluster descriptions and generates ClstrRecs.

Usage

```
clstrrec_gen(clstr_list, txid, sqs, typ)
```

Arguments

`clstr_list`	List of list of cluster descriptions
`txid`	Taxonomic node ID
`sqs`	Sequence records
`typ`	Cluster type

Value

list of ClstrRecs

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstrs_calc` *Calculate clusters for all sequences in wd*

Description

Loop through downloaded sequences for each clade and hierarchically find clusters using BLAST.

Usage

```
clstrs_calc(txdct, ps)
```

Arguments

`txdct`	Taxonomic dictionary
`ps`	Parameters list, generated with parameters()

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstrs_join` *Join clusters for merging*

Description

Uses seed sequence BLAST results and IDs to join clusters identified as sisters into single clusters. Resulting object is of joined clusters, merging is required to reformat the clusters for subsequent analysis.

Usage

```
clstrs_join(blast_res, seed_ids, all_clstrs, ps)
```

Arguments

`blast_res`	Seed sequence BLAST results
`seed_ids`	Seed sequence IDs
`all_clstrs`	List of all clusters
`ps`	Parameters list, generated with parameters()

Value

list of joined clusters

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstrs_merge` *Merge joined clusters*

Description

Takes a list of joined clusters and computes each data slot to create a single merged cluster. txdct is required for parent look-up.

Usage

```
clstrs_merge(jnd_clstrs, txdct)
```

Arguments

`jnd_clstrs`	List of joined clusters
`txdct`	Taxonomic dictionary

Value

list of ClstrRecs

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstrs_renumber` *Renumber cluster IDs*

Description

Returns a ClstrArc with ID determined by the number of sequences in each cluster.

Usage

```
clstrs_renumber(clstrrecs)
```

Arguments

`clstrrecs`	List of clusters

Value

ClstrArc

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstrs_save` *Save clusters to cache*

Description

Saves clusters generated by `clstr_all` to cache.

Usage

```
clstrs_save(wd, txid, clstrs)
```

Arguments

`wd`	Working directory
`txid`	Taxonomic ID, numeric
`clstrs`	cluster list

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstr_all` *Hierarchically cluster all sequences of a txid*

Description

Identifies all direct and subtree clusters for a taxonomic ID.

Usage

```
clstr_all(txid, sqs, txdct, ps, lvl = 0)
```

Arguments

`txid`	Taxonomic ID
`sqs`	Sequence object of all downloaded sequences
`txdct`	Taxonomic dictionary
`ps`	Parameters list, generated with parameters()
`lvl`	Integer, number of message indentations indicating code depth.

Value

ClstrArc

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstr_direct` *Cluster sequences directly associated with txid*

Description

In GenBank certain sequences may only be associated with a higher level taxon (e.g. genus, family ...). This function generates clusters from these sequences, alone. This function identifies such sequences in the sequence object and generates a list of clusters of cl_type ’direct’.

Usage

```
clstr_direct(txid, sqs, txdct, ps, lvl)
```

Arguments

`txid`	Taxonomic ID
`sqs`	Sequence object of all downloaded sequences
`txdct`	Taxonomic dictionary
`ps`	Parameters list, generated with parameters()
`lvl`	Integer, number of message indentations indicating code depth.

Value

ClstrArc

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstr_sqs` *Identify clusters from sequences*

Description

Given a sequence object, this function will generate a list of cluster objects using BLAST

Usage

```
clstr_sqs(txid, sqs, ps, lvl, typ = c("direct", "subtree", "paraphyly"))
```

Arguments

`txid`	Taxonomic ID
`sqs`	Sequence object of sequences to be BLASTed
`ps`	Parameters list, generated with parameters()
`lvl`	Integer, number of message indentations indicating code depth.
`typ`	Direct, subtree or paraphyly?

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clstr_subtree` *Cluster all sequences descending from a txid*

Description

Identifies clusters from sequences associated with a txid and all its descendants. Clusters returned by this function will thus be of cl_type 'subtree'.

Usage

```
clstr_subtree(txid, sqs, txdct, dds, ps, lvl)
```

Arguments

`txid`	Taxonomic ID
`sqs`	Sequence object of all downloaded sequences
`txdct`	Taxonomic dictionary
`dds`	Vector of direct descendants
`ps`	Parameters list, generated with parameters()
`lvl`	Integer, number of message indentations indicating code depth.

Value

ClstrArc

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`clusters2_run` *Run the cluster2 stage*

Description

Run the fourth stage of the phylotaR pipeline, cluster2. Identify clusters at higher taxonomic levels by merging sister clusters.

Usage

```
clusters2_run(wd)
```

Arguments

`wd`	Working directory

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
## Not run:

  # Note: this example requires BLAST and internet to run.

  # example with temp folder
  wd <- file.path(tempdir(), 'aotus')
  # setup for aotus, make sure aotus/ folder already exists
  if (!dir.exists(wd)) {
    dir.create(wd)
  }
  ncbi_dr <- '[SET BLAST+ BIN PATH HERE]'
  setup(wd = wd, txid = 9504, ncbi_dr = ncbi_dr)  # txid for Aotus primate genus
  # individually run stages
  taxise_run(wd = wd)
  download_run(wd = wd)
  clusters_run(wd = wd)
  clusters2_run(wd = wd)

## End(Not run)
```

`clusters_run` *Run the cluster stage*

Description

Run the third stage of the phylotaR pipeline, cluster. This stage hierarchically traverses the taxonomy identifying all direct and subtree clusters from downloaded sequences. Any taxonomic nodes too small for cluster identification are placed into paraphyletic clusters.

Usage

```
clusters_run(wd)
```

Arguments

`wd`	Working directory

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
## Not run:

  # Note: this example requires BLAST and internet to run.

  # example with temp folder
  wd <- file.path(tempdir(), 'aotus')
  # setup for aotus, make sure aotus/ folder already exists
  if (!dir.exists(wd)) {
    dir.create(wd)
  }
  ncbi_dr <- '[SET BLAST+ BIN PATH HERE]'
  setup(wd = wd, txid = 9504, ncbi_dr = ncbi_dr)  # txid for Aotus primate genus
  # individually run stages
  taxise_run(wd = wd)
  download_run(wd = wd)
  clusters_run(wd = wd)

## End(Not run)
```

`cmdln` *Run a command via terminal/command prompt*

Description

Provide the command and arguments as a vector. Also can take a lgfl to which all stdout and stderr is written. If lgfl is not provided, a list is returned of ’status’, ’stdout’ and ’stderr’. Else only the status is returned - 1 success, 0 failed.

Usage

```
cmdln(cmd, args, ps, lgfl = NULL)
```

Arguments

`cmd`	Command to be run
`args`	Vector of command arguments, each parameter and value must be a separate element
`ps`	Paramters
`lgfl`	File to which stdout/err will be written

Details

Note, stdout/err are returned as ’raw’. Use rawToChar() to convert to characters.

Value

status, integer or character

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`cTrees` *cTrees*

Description

Return `TreeMen` of concatenated trees.

Usage

```
cTrees(x, ...)
```

Arguments

`x`	`TreeMan` or `TreeMen` objects
`...`	more `TreeMan` or `TreeMen` objects

Details

Concatenate trees into single `TreeMen` object.

See Also

`TreeMen-class`, `TreeMan-class`, `list-to-TreeMen`

Examples

```
trees <- cTrees(randTree(10), randTree(10))
```

`cycads` *cycads*

Description

cycads

Format

A TreeMan or Phylota object

Examples

```
data("cycads")
```

`descendants_get` *Get descendants*

Description

Look-up either direct or all taxonomic descendants of a node from taxonomic dictionary.

Usage

```
descendants_get(id, txdct, direct = FALSE)
```

Arguments

`id`	txid
`txdct`	TaxDict
`direct`	T/F, return only direct descendants?

Value

vector

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, progress_read(), progress_reset(), progress_save(), rank_get(), rawseqrec_breakdown(), safely_connect(), search_and_cache(), searchterm_gen(), seeds_blast(), seq_download(), seqarc_gen(), seqrec_augment(), seqrec_convert(), seqrec_gen(), seqrec_get(), sids_check(), sids_get(), sids_load(), sids_save(), sqs_count(), sqs_save(), stage_args_check(), stages_run(), tax_download(), taxdict_gen(), taxtree_gen(), txids_get(), txnds_count(), warn()

---

download_obj_check *Check an object returned from rentrez function*

Description

Returns T/F. Checks if object returned from rentrez function is as expected.

Usage

```
download_obj_check(obj)
```

Arguments

obj	Object returned from rentrez function

Value

T/F

See Also

Other run-private: batcher(), blast_clstr(), blast_filter(), blast_setup(), blast_sqs(), blastcache_load(), blastcache_save(), blastdb_gen(), blastn_run(), cache_rm(), cache_setup(), clade_select(), clstr2_calc(), clstr_all(), clstr_direct(), clstr_sqs(), clstr_subtree(), clstrarc_gen(), clstrarc_join(), clstrrec_gen(), clstrs_calc(), clstrs_join(), clstrs_merge(), clstrs_renumber(), clstrs_save(), cmdln(), descendants_get(), error(), gb_extract(), hierarchic_download(), info(), ncbicache_load(), ncbicache_save(), obj_check(), obj_load(), obj_save(), outfmt_get(), parameters_load(), parameters_setup(), parent_get(), progress_init(), progress_read(), progress_reset(), progress_save(), rank_get(), rawseqrec_breakdown(), safely_connect(), search_and_cache(), searchterm_gen(), seeds_blast(), seq_download(), seqarc_gen(), seqrec_augment(), seqrec_convert(), seqrec_gen(), seqrec_get(), sids_check(), sids_get(), sids_load(), sids_save(), sqs_count(), sqs_save(), stage_args_check(), stages_run(), tax_download(), taxdict_gen(), taxtree_gen(), txids_get(), txnds_count(), warn()

`download_run` *Run download stage*

Description

Run the second stage of phylotaR, download. This stage downloads sequences for all nodes with sequence numbers less than mxsqs. It hierarchically traverses the taxonomy for each node and downloads direct and subtree sequences for all descendants.

Usage

```
download_run(wd)
```

Arguments

`wd`	Working directory

Examples

```
## Not run:

  # Note: this example requires BLAST and internet to run.

  # example with temp folder
  wd <- file.path(tempdir(), 'aotus')
  # setup for aotus, make sure aotus/ folder already exists
  if (!dir.exists(wd)) {
    dir.create(wd)
  }
  ncbi_dr <- '[SET BLAST+ BIN PATH HERE]'
  setup(wd = wd, txid = 9504, ncbi_dr = ncbi_dr)  # txid for Aotus primate genus
  # individually run stages
  taxise_run(wd = wd)
  download_run(wd = wd)

## End(Not run)
```

`dragonflies` *dragonflies*

Description

dragonflies

Format

A TreeMan or Phylota object

Examples

```
data("dragonflies")
```

`drop_by_rank` *Reduce clusters to specific rank*

Description

Identifies higher level taxa for each sequence in clusters for given rank. Selects representative sequences for each unique taxon using the choose_by functions. By default, the function will choose the top ten sequences by first sorting by those with fewest number of ambiguous sequences, then by youngest, then by sequence length.

Usage

```
drop_by_rank(
  phylota,
  rnk = "species",
  keep_higher = FALSE,
  n = 10,
  choose_by = c("pambgs", "age", "nncltds"),
  greatest = c(FALSE, FALSE, TRUE)
)
```

Arguments

`phylota`	Phylota object
`rnk`	Taxonomic rank
`keep_higher`	Keep higher taxonomic ranks?
`n`	Number of sequences per taxon
`choose_by`	Vector of selection functions
`greatest`	Greatest of lowest for each choose_by function

Value

phylota

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data("dragonflies")
# For faster computations, let's only work with the 5 clusters.
dragonflies <- drop_clstrs(phylota = dragonflies, cid = dragonflies@cids[10:15])


# We can use drop_by_rank() to reduce to 10 sequences per genus for each cluster
(reduced_1 <- drop_by_rank(phylota = dragonflies, rnk = 'genus', n = 10,
                           choose_by = c('pambgs', 'age', 'nncltds'),
                           greatest = c(FALSE, FALSE, TRUE)))

# We can specify what aspects of the sequences we would like to select per genus
# By default we select the sequences with fewest ambiguous nucleotides (e.g.
# we avoid Ns), the youngest age and then longest sequence.
# We can reverse the 'greatest' to get the opposite.
(reduced_2 <- drop_by_rank(phylota = dragonflies, rnk = 'genus', n = 10,
                           choose_by = c('pambgs', 'age', 'nncltds'),
                           greatest = c(TRUE, TRUE, FALSE)))


# Leading to smaller sequnces ...
r1_sqlngth <- mean(get_sq_slot(phylota = reduced_1,
                                sid = reduced_1@sids, slt_nm = 'nncltds'))
r2_sqlngth <- mean(get_sq_slot(phylota = reduced_2,
                                sid = reduced_2@sids, slt_nm = 'nncltds'))
(r1_sqlngth > r2_sqlngth)
# ... with more ambigous characters ....
r1_pambgs <- mean(get_sq_slot(phylota = reduced_1, sid = reduced_1@sids,
                              slt_nm = 'pambgs'))
r2_pambgs <- mean(get_sq_slot(phylota = reduced_2, sid = reduced_2@sids,
                              slt_nm = 'pambgs'))
(r1_pambgs < r2_pambgs)
# .... and older ages (measured in days since being added to GenBank).
r1_age <- mean(get_sq_slot(phylota = reduced_1, sid = reduced_1@sids,
                           slt_nm = 'age'))
r2_age <- mean(get_sq_slot(phylota = reduced_2, sid = reduced_2@sids,
                           slt_nm = 'age'))
(r1_age < r2_age)


# Or... we can simply reduce the clusters to just one sequence per genus
(dragonflies <- drop_by_rank(phylota = dragonflies, rnk = 'genus', n = 1))
```

`drop_clstrs` *Drop cluster records from phylota object*

Description

Drops all clusters except those identified by user.

Usage

```
drop_clstrs(phylota, cid)
```

Arguments

`phylota`	Phylota object
`cid`	Cluster ID(s) to be kept

Value

phylota

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data("dragonflies")
# specify cids to *keep*
random_cids <- sample(dragonflies@cids, 100)
# drop an entire cluster
nbefore <- length(dragonflies@cids)
dragonflies <- drop_clstrs(phylota = dragonflies, cid = random_cids)
nafter <- length(dragonflies@cids)
# now there are only 100 clusters
(nafter < nbefore)
```

`drop_sqs` *Drop sequences in a cluster*

Description

Drop all sequences in a cluster except those identified by user.

Usage

```
drop_sqs(phylota, cid, sid)
```

Arguments

`phylota`	Phylota object
`cid`	Cluster ID
`sid`	Sequence ID(s) to be kept

Value

phylota

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data("dragonflies")
# drop random sequences from cluster 0
clstr <- dragonflies[['0']]
# specify the sids to *keep*
sids <- sample(clstr@sids, 100)
(dragonflies <- drop_sqs(phylota = dragonflies, cid = '0', sid = sids))
# Note, sequences dropped may be represented in other clusters
```

`error` *Write error message to log*

Description

Inform a user if an error has occurred in log.txt, halt pipeline.

Usage

```
error(ps, ...)
```

Arguments

`ps`	Parameters list, generated with parameters()
`...`	Message elements for concatenating

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`fastCheckTreeMan` *Check if tree is correct, fast!*

Description

Return T/F if tree is a true `TreeMan` object

Usage

```
fastCheckTreeMan(object)
```

Arguments

`object`	`TreeMan` object

Details

Whenever a tree is first initiated this check is used. For more detailed checking use `checkNdlst`.

See Also

`checkNdlst`, `checkTreeMen`

`gb_extract` *Extract elements from a raw GenBank record*

Description

Returns a list of elements from a GenBank record such as ’organism’, ’sequence’ and features.

Usage

```
gb_extract(record)
```

Arguments

`record`	raw GenBank text record

Details

Uses restez extract functions. See restez package for more details.

Value

list of GenBank elements

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`getAge` *Get age of tree*

Description

Returns age, numeric, of tree

Usage

```
getAge(tree, parallel = FALSE)
```

Arguments

`tree`	`TreeMan` object
`parallel`	logical, make parallel?

Details

Calculates the age of a tree, determined as the maximum tip to root distance.

See Also

`updateSlts`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
(getAge(tree))
```

`getBiprts` *Get the sets of labels for each bipartition in tree*

Description

Returns a list of tip IDs for each branch in the tree. Options allow the user to act as if the root is not present and to use a universal code for comparing between trees.

Usage

```
getBiprts(tree, tips = tree@tips, root = TRUE, universal = FALSE)
```

Arguments

`tree`	`TreeMan` object
`tips`	vector of tips IDs to use for bipartitions
`root`	Include the root for the bipartitions? Default TRUE.
`universal`	Create a code for comparing between trees

Details

Setting `root` to FALSE will ignore the bipartitions created by the root. Setting `universal` to TRUE will return a vector of 0s and 1s, not a list of tips. These codes will always begin with 1, and will allow for the comparison of splits between trees as they do not have "chiralty", so to speak.

See Also

`calcDstRF`

Examples

```
tree <- randTree(10)
# get all of the tip IDs for each branch in the rooted tree
(getBiprts(tree))
# ignore the root and get bipartitions for unrooted tree
(getBiprts(tree, root = FALSE))
# use the universal code for comparing splits between trees
(getBiprts(tree, root = FALSE, universal = TRUE))
```

`getCnnctdNds` *Get all nodes connected by given tips*

Description

Return a vector of IDs of all nodes that are connected to tip IDs given.

Usage

```
getCnnctdNds(tree, tids)
```

Arguments

`tree`	`TreeMan` object
`tids`	vector of tip IDs

Details

Returns a vector. This function is the basis for `calcPhyDv()`, it determines the unique set of nodes connected for a set of tips.

See Also

`getUnqNds`, `calcFrPrp`, `calcPhyDv`

Examples

```
tree <- randTree(10)
cnntdnds <- getCnnctdNds(tree, c("t1", "t2"))
```

`getDcsd` *Get extinct tips from a tree*

Description

Return all extinct tip `ID`s.

Usage

```
getDcsd(tree, tol = 1e-08)
```

Arguments

`tree`	`TreeMan` object
`tol`	zero tolerance

Details

Returns a vector.

See Also

getLvng, isUltrmtrc, https://github.com/DomBennett/treeman/wiki/get-methods

Examples

```
tree <- randTree(10)
(getDcsd(tree))
```

---

getLvng *Get extant tips from a tree*

Description

Return all extant tip `IDs`.

Usage

```
getLvng(tree, tol = 1e-08)
```

Arguments

`tree`	`TreeMan` object
`tol`	zero tolerance

Details

Returns a vector.

See Also

getDcsd, isUltrmtrc, https://github.com/DomBennett/treeman/wiki/get-methods

Examples

```
tree <- randTree(10)
(getLvng(tree))
```

`getNdAge` *Get age*

Description

Return the age for `id`. Requires the known age of the tree to be provided.

Usage

```
getNdAge(tree, id, tree_age)
```

Arguments

`tree`	`TreeMan` object
`id`	node id
`tree_age`	numeric value of known age of tree

Details

Returns a numeric.

See Also

`getNdsAge`, `getSpnAge`, `getSpnsAge`, `getPrnt`, `getAge` `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
data(mammals)
# when did apes emerge?
# get parent id for all apes
prnt_id <- getPrnt(mammals, ids = c("Homo_sapiens", "Hylobates_concolor"))
# mammal_age <- getAge(mammals)  # ~166.2, needs to be performed when tree is not up-to-date
getNdAge(mammals, id = prnt_id, tree_age = 166.2)
```

`getNdKids` *Get children IDs*

Description

Return the node ids of all tips that descend from node.

Usage

```
getNdKids(tree, id)
```

Arguments

`tree`	TreeMan object
`id`	node id

Details

Returns a vector

See Also

getNdsKids, https://github.com/DomBennett/treeman/wiki/get-methods

Examples

```
tree <- randTree(10)
# everyone descends from root
getNdKids(tree, id = tree["root"])
```

---

getNdLng *Get lineage*

Description

Return unique taxonomic names for connecting `id` to root.

Usage

```
getNdLng(tree, id)
```

Arguments

`tree`	TreeMan object
`id`	node id

Details

Returns a vector.

See Also

getNdsLng, getNdsFrmTxnyms, https://github.com/DomBennett/treeman/wiki/get-methods

Examples

```
data(mammals)
# return human lineage
getNdLng(mammals, id = "Homo_sapiens")
```

`getNdPD` *Get phylogenetic diversity of node*

Description

Return summed value of all descending spns

Usage

```
getNdPD(tree, id)
```

Arguments

`tree`	`TreeMan` object
`id`	node id

Details

Sums the lengths of all descending branches from a node.

See Also

`getNdsPD`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdPD(tree, id = "n1") # return PD of n1 which in this case is for the whole tree
```

`getNdPrdst` *Get pre-distance*

Description

Return root to tip distance (prdst) for `id`

Usage

```
getNdPrdst(tree, id)
```

Arguments

`tree`	`TreeMan` object
`id`	node id

Details

Sums the lengths of all branches from `id` to root.

See Also

`getNdsPrdst`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdPrdst(tree, id = "t1") # return the distance to root from t1
```

`getNdPrids` *Get pre-nodes to root*

Description

Return node ids for connecting `id` to root.

Usage

```
getNdPrids(tree, id)
```

Arguments

`tree`	`TreeMan` object
`id`	node id

Details

Returns a vector. IDs are returned order from node ID to root.

See Also

`getNdsPrids`, `getNdPtids`, `getNdsPtids`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
# get all nodes to root
getNdPrids(tree, id = "t1")
```

`getNdPtids` *Get post-nodes to tips*

Description

Return node ids for connecting `id` to kids.

Usage

```
getNdPtids(tree, id)
```

Arguments

`tree`	`TreeMan` object
`id`	node id

Details

Returns a vector.

See Also

`getNdsPtids`, `getNdPrids`, `getNdsPrids`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
# get all nodes from root to tip
getNdPtids(tree, id = "n1")
```

`getNdsAge` *Get ages for multiple nodes*

Description

Return the age for `ids`.

Usage

```
getNdsAge(tree, ids, tree_age, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	vector of node ids
`tree_age`	numeric value of known age of tree
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Returns a vector, parallelizable.

See Also

`getNdAge`, `getSpnAge`, `getSpnsAge`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdsAge(tree, ids = tree["nds"], tree_age = getAge(tree))
```

`getNdsFrmTxnyms` *Get IDs for nodes represented txnyms*

Description

Return a list of IDs for any node that contains the given txnyms.

Usage

```
getNdsFrmTxnyms(tree, txnyms)
```

Arguments

`tree`	`TreeMan` object
`txnyms`	vector of taxonomic group names

Details

Returns a list. Txnyms must be spelt correctly.

See Also

`taxaResolve`, `setTxnyms`, `searchTxnyms`, `getNdsLng`, `getNdLng`

Examples

```
data(mammals)
# what ID represents the apes?
getNdsFrmTxnyms(mammals, "Hominoidea")
```

`getNdsKids` *Get children IDs for multiple nodes*

Description

Return the node ids of all tips that descend from each node in `ids`.

Usage

```
getNdsKids(tree, ids, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	vector of node ids
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Returns a list, parallelizable.

See Also

`getNdKids`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdsKids(tree, id = tree["nds"])
```

`getNdsLng` *Get lineage for multiple nodes*

Description

Return unique taxonyms for connecting `ids` to root.

Usage

```
getNdsLng(tree, ids, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	vector of node ids
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Returns a list, parallelizable.

See Also

`getNdLng`, `getNdsFrmTxnyms`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
data(mammals)
# return human and gorilla lineages
getNdsLng(mammals, id = c("Homo_sapiens", "Gorilla_gorilla"))
```

`getNdSlt` *Get a node slot*

Description

Returns the value of named slot.

Usage

```
getNdSlt(tree, slt_nm, id)
```

Arguments

`tree`	`TreeMan` object
`slt_nm`	slot name
`id`	node id

Details

Returned object depends on name, either character, vector or numeric. Default node slots are: id, spn, prid, ptid and txnym. If slot is empty, returns NA.

See Also

`getNdsSlt`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdSlt(tree, slt_nm = "spn", id = "t1") # return span of t1
```

---

getNdsPD *Get phylogenetic diversities of nodes*

Description

Return summed value of all descending spns

Usage

```
getNdsPD(tree, ids, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	vector of node ids
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Sums the lengths of all descending branches from a node.

See Also

`getNdPD`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdsPD(tree, ids = tree["all"]) # return PD of all ids
```

`getNdsPrdst` *Get pre-distances*

Description

Return root to tip distances (prdst) for `ids`

Usage

```
getNdsPrdst(tree, ids, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	vector of node ids
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Sums the lengths of all branches from `ids` to root.

See Also

`getNdPrdst`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdsPrdst(tree, ids = tree["tips"]) # return prdsts for all tips
```

`getNdsPrids` *Get pre-nodes for multiple nodes*

Description

Return node ids for connecting `id` to root.

Usage

```
getNdsPrids(tree, ids, ordrd = FALSE, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	vector of node ids
`ordrd`	logical, ensure returned prids are ordered ID to root
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Returns a list, parallizable. The function will work faster if `ordrd` is FALSE.

See Also

`getNdPrids`, `getNdPtids`, `getNdsPtids`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdsPrids(tree, ids = tree["tips"])
```

`getNdsPtids` *Get post-nodes to tips for multiple nodes*

Description

Return node ids for connecting `ids` to kids.

Usage

```
getNdsPtids(tree, ids, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	vector of node ids
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Returns a list, parallizable.

See Also

`getNdPtids`, `getNdPrids`, `getNdsPrids`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
# get all nodes to tip for all nodes
getNdsPtids(tree, ids = tree["nds"])
```

`getNdsSlt` *Get a node slot for multiple nodes*

Description

Returns the values of named slot as a vector for atomic values, else list.

Usage

```
getNdsSlt(tree, slt_nm, ids, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`slt_nm`	slot name
`ids`	vector of node ids
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Returned object depends on name, either character, vector or numeric. Parallelizable. Default node slots are: id, spn, prid, ptid and txnym.

See Also

`getNdSlt`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdsSlt(tree, slt_nm = "spn", ids = tree["tips"]) # return spans of all tips
```

`getNdsSstr` *Get sister id*

Description

Returns the ids of the sister(s) of nd ids given.

Usage

```
getNdsSstr(tree, ids, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	nd ids
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

An error is raised if there is no sister (e.g. for the root). There can be more than one sister if tree is polytomous. Parallelizable.

See Also

`getNdSstr`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdsSstr(tree, ids = tree["tips"])
```

`getNdSstr` *Get sister id*

Description

Returns the id of the sister(s) of node id given.

Usage

```
getNdSstr(tree, id)
```

Arguments

`tree`	`TreeMan` object
`id`	node id

Details

An error is raised if there is no sister (e.g. for the root). There can be more than one sister if tree is polytomous.

See Also

`getNdsSstr`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
getNdSstr(tree, id = "t1")
```

`getOtgrp` *Get outgroup*

Description

Return the outgroup based on a tree and a vector of IDs.

Usage

```
getOtgrp(tree, ids)
```

Arguments

`tree`	`TreeMan` object
`ids`	vector of node ids

Details

Returns a id, character. If there are multiple possible outgroups, returns NULL.

See Also

`https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
data(mammals)
# orangutan is an outgroup wrt humans and chimps
getOtgrp(mammals, ids = c("Homo_sapiens", "Pan_troglodytes", "Pongo_pygmaeus"))
```

`getPath` *Get path between nodes*

Description

Return node ids for connecting `from` to `to`.

Usage

```
getPath(tree, from, to)
```

Arguments

`tree`	`TreeMan` object
`from`	starting node id
`to`	ending node id

Details

Returns a vector, first id is `from` to `to`.

See Also

https://github.com/DomBennett/treeman/wiki/get-methods

Examples

```
data(mammals)
# what's the phylogenetic distance from humans to gorillas?
ape_id <- getPrnt(mammals, ids = c("Homo_sapiens", "Hylobates_concolor"))
pth <- getPath(mammals, from = "Homo_sapiens", to = "Gorilla_gorilla")
sum(getNdsSlt(mammals, ids = pth, slt_nm = "spn"))
```

`getPrnt` *Get parent*

Description

Return parental (most recent common ancestor) node id for `ids`.

Usage

```
getPrnt(tree, ids)
```

Arguments

`tree`	TreeMan object
`ids`	vector of node ids

Details

Returns a character.

See Also

`getSubtree`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
data(mammals)
# choosing ids from the two main branches of apes allows to find the parent for all apes
ape_id <- getPrnt(mammals, ids = c("Homo_sapiens", "Hylobates_concolor"))
```

`getSpnAge` *Get age range*

Description

Return start and end ages for `id` from when it first appears to when it splits

Usage

```
getSpnAge(tree, id, tree_age)
```

Arguments

`tree`	TreeMan object
`id`	node id
`tree_age`	numeric value of known age of tree

Details

Returns a dataframe.

See Also

`getNdAge`, `getNdsAge`, `getSpnsAge`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
data(mammals)
# mammal_age <- getAge(mammals)  # ~166.2, needs to be performed when tree is not up-to-date
getSpnAge(mammals, id = "Homo_sapiens", tree_age = 166.2)
```

`getSpnsAge` *Get age ranges for multiple nodes*

Description

Return start and end ages for `ids` from when they first appear to when they split

Usage

```
getSpnsAge(tree, ids, tree_age, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	vector of node ids
`tree_age`	numeric value of known age of tree
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Returns a dataframe, parallelizable.

See Also

`getNdAge`, `getNdsAge`, `getSpnAge`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
tree <- randTree(10)
# all nodes but root
ids <- tree["nds"][tree["nds"] != tree["root"]]
getSpnsAge(tree, ids = ids, tree_age = getAge(tree))
```

`getSubtree` *Get subtree*

Description

Return tree descending from `id`.

Usage

```
getSubtree(tree, id)
```

Arguments

`tree`	`TreeMan` object
`id`	node id

Details

Returns a `TreeMan`, parallelizable. `id` must be an internal node.

See Also

`getPrnt`, `addClade`, `https://github.com/DomBennett/treeman/wiki/get-methods`

Examples

```
data(mammals)
# get tree of apes
ape_id <- getPrnt(mammals, ids = c("Homo_sapiens", "Hylobates_concolor"))
apes <- getSubtree(mammals, id = ape_id)
summary(apes)
```

`getUnqNds` *Get unique nodes represented by tips*

Description

Return a list of IDs for any node that are represented by tip IDs given.

Usage

```
getUnqNds(tree, tids)
```

Arguments

`tree`	`TreeMan` object
`tids`	vector of tip IDs

Details

Returns a vector.

See Also

`getCnnctdNds`, `calcFrPrp`, `calcPhyDv`

Examples

```
tree <- randTree(10)
unqnds <- getUnqNds(tree, c("t1", "t2"))
```

`get_clstr_slot` *Get slot data for each cluster record*

Description

Get slot data for cluster(s)

Usage

```
get_clstr_slot(phylota, cid, slt_nm = list_clstrrec_slots())
```

Arguments

`phylota`	Phylota object
`cid`	Cluster ID
`slt_nm`	Slot name

Value

vector

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data('aotus')
random_cid <- sample(aotus@cids, 1)
(get_clstr_slot(phylota = aotus, cid = random_cid, slt_nm = 'seed'))
# see list_clstrrec_slots() for available slots
(list_clstrrec_slots())
```

`get_nsqs` *Count number of sequences*

Description

Count the number of sequences in a cluster(s).

Usage

```
get_nsqs(phylota, cid)
```

Arguments

`phylota`	Phylota object
`cid`	Cluster ID(s)

Value

vector

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data("cycads")
# count seqs for a random 10 clusters
random_cids <- sample(cycads@cids, 10)
nsqs <- get_nsqs(phylota = cycads, cid = random_cids)
```

`get_ntaxa` *Count number of unique taxa*

Description

Count the number of unique taxa represented by cluster(s) or sequences in phylota table Use rnk to specify a taxonomic level to count. If NULL counts will be made to the lowest level reported on NCBI.

Usage

```
get_ntaxa(phylota, cid = NULL, sid = NULL, rnk = NULL, keep_higher = FALSE)
```

Arguments

`phylota`	Phylota object
`cid`	Cluster ID(s)
`sid`	Sequence ID(s)
`rnk`	Taxonomic rank
`keep_higher`	Keep higher taxonomic ranks?

Value

vector

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data('bromeliads')
# how many species are there?
(get_ntaxa(phylota = bromeliads, cid = '0', rnk = 'species'))
# how many genera are there?
(get_ntaxa(phylota = bromeliads, cid = '0', rnk = 'genus'))
# how many families are there?
(get_ntaxa(phylota = bromeliads, cid = '0', rnk = 'family'))
# use list_ncbi_ranks() to see available rank names
(list_ncbi_ranks())
```

`get_sq_slot` *Get slot data for each sequence*

Description

Get slot data for either or sequences in a cluster of a vector of sequence IDs. Use list_seqrec_slots() for a list of available slots.

Usage

```
get_sq_slot(phylota, cid = NULL, sid = NULL, slt_nm = list_seqrec_slots())
```

Arguments

`phylota`	Phylota object
`cid`	Cluster ID
`sid`	Sequence ID(s)
`slt_nm`	Slot name

Value

vector

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data('aotus')
random_sid <- sample(aotus@sids, 1)
(get_sq_slot(phylota = aotus, sid = random_sid, slt_nm = 'dfln'))
# see list_seqrec_slots() for available slots
(list_seqrec_slots())
```

`get_stage_times` *Get run times for different stages*

Description

Get slot data for taxa(s)

Usage

```
get_stage_times(wd)
```

Arguments

`wd`	Working directory

Value

list of runtimes in minutes

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
## Not run:

  # Note, this example requires a wd with a completed phylotaR run
  # return a named list of the time take in minutes for each stage
  get_stage_times(wd = wd)

## End(Not run)
```

`get_txids` *Get taxonomic IDs by rank*

Description

Return taxonomic IDs for a vector of sequence IDs or all sequences in a cluster. User can specify what rank the IDs should be returned. If NULL, the lowest level is returned.

Usage

```
get_txids(
  phylota,
  cid = NULL,
  sid = NULL,
  txids = NULL,
  rnk = NULL,
  keep_higher = FALSE
)
```

Arguments

`phylota`	Phylota object
`cid`	Cluster ID
`sid`	Sequence ID(s)
`txids`	Vector of txids
`rnk`	Taxonomic rank
`keep_higher`	Keep higher taxonomic IDs?

Details

txids can either be provided by user or they can be determined for a vector of sids or for a cid. If keep_higher is TRUE, any sequence that has a identity that is higher than the given rank will be returned. If FALSE, these sequences will return ''.

Value

vector

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data('bromeliads')
# get all the genus IDs and names
genus_ids <- get_txids(phylota = bromeliads, txids = bromeliads@txids,
                       rnk = 'genus')
genus_ids <- unique(genus_ids)
# drop empty IDs -- this happens if a given lineage has no ID for specified rank
genus_ids <- genus_ids[genus_ids != '']
# get names
(get_tx_slot(phylota = bromeliads, txid = genus_ids, slt_nm = 'scnm'))
```

`get_tx_slot` *Get slot data for each taxon record*

Description

Get slot data for taxa(s)

Usage

```
get_tx_slot(phylota, txid, slt_nm = list_taxrec_slots())
```

Arguments

`phylota`	Phylota object
`txid`	Taxonomic ID
`slt_nm`	Slot name

Value

vector or list

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data('aotus')
random_txid <- sample(aotus@txids, 1)
(get_tx_slot(phylota = aotus, txid = random_txid, slt_nm = 'scnm'))
# see list_taxrec_slots() for available slots
(list_taxrec_slots())
```

`hierarchic_download` *Hierarchically get sequences for a txid*

Description

Looks up and downloads sequences for a taxonomic ID.

Usage

```
hierarchic_download(txid, txdct, ps, lvl = 0)
```

Arguments

`txid`	Taxonomic node ID, numeric
`txdct`	Taxonomic dictionary
`ps`	Parameters list, generated with parameters()
`lvl`	Integer, number of message indentations indicating code depth.

Value

Vector of SeqRecs

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`info` *Write info message to log*

Description

Inform a user via log.txt of pipeline progress.

Usage

```
info(lvl, ps, ...)
```

Arguments

`lvl`	Integer, number of message indentations indicating code depth.
`ps`	Parameters list, generated with parameters()
`...`	Message elements for concatenating

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`isUltrmtrc` *Is tree ultrametric?*

Description

Return TRUE if all tips end at 0, else FALSE.

Usage

```
isUltrmtrc(tree, tol = 1e-08)
```

Arguments

`tree`	`TreeMan` object
`tol`	zero tolerance

Details

Returns a boolean. This function works in the background for the `['ultr']` slot in a `TreeMan` object.

See Also

`getLvng`, `getDcsd`

Examples

```
tree <- randTree(10)
(isUltrmtrc(tree))
```

`is_txid_in_clstr` *Is txid in cluster?*

Description

Checks if given txid is represented by any of the sequences of a cluster by searching through all the sequence search organism lineages.

Usage

```
is_txid_in_clstr(phylota, txid, cid)
```

Arguments

`phylota`	Phylota
`txid`	Taxonomic ID
`cid`	Cluster ID

Value

boolean

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data(tinamous)
cid <- tinamous@cids[[1]]
clstr <- tinamous[[cid]]
sq <- tinamous[[clstr@sids[[1]]]]
txid <- sq@txid
# expect true
is_txid_in_clstr(phylota = tinamous, txid = txid, cid = cid)
```

`is_txid_in_sq` *Is txid in sequence?*

Description

Checks if given txid is represented by sequence by looking at sequence source organism's lineage.

Usage

```
is_txid_in_sq(phylota, txid, sid)
```

Arguments

`phylota`	Phylota
`txid`	Taxonomic ID
`sid`	Sequence ID

Value

boolean

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
data(tinamous)
sid <- tinamous@sids[[1]]
sq <- tinamous[[sid]]
txid <- sq@txid
# expect true
is_txid_in_sq(phylota = tinamous, txid = txid, sid = sid)
```

`list-to-TreeMen` *Convert list to a TreeMen*

Description

Return a `TreeMen` object from a list of `TreeMans`

See Also

`TreeMen-class`

Examples

```
trees <- list("tree_1" = randTree(10), "tree_2" = randTree(10))
trees <- as(trees, "TreeMen")
```

`list_clstrrec_slots` *List all ClstrRec slots*

Description

Returns a vector of all available ClstrRec slots of type character, integer and numeric.

Usage

```
list_clstrrec_slots()
```

Value

vector

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

`list_ncbi_ranks` *List all NCBI Ranks*

Description

Returns a vector of all NCBI taxonomic ranks in descending order.

Usage

```
list_ncbi_ranks()
```

Value

vector

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

`list_seqrec_slots` *List all SeqRec slots*

Description

Returns a vector of all available SeqRec slots of type character, integer and numeric.

Usage

```
list_seqrec_slots()
```

Value

vector

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

`list_taxrec_slots` *List all TaxRec slots*

Description

Returns a vector of all available TaxRec slots of type character, integer and numeric.

Usage

```
list_taxrec_slots()
```

Value

vector

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

`loadTreeMan` *Load a TreeMan object in serialization format*

Description

`TreeMan` equivalent to `load()` but able to handle node matrices.

Usage

```
loadTreeMan(file)
```

Arguments

`file`	file path

Details

It is not possible to use `save()` on `TreeMan` objects with node matrices. Node matrices are bigmemory matrices and are therefore outside the R environment, see bigmemory documentation for more information. Saving and loading a bigmemory matrix may cause memory issues in R and cause R to crash.

This function can safely read a `TreeMan` object with and without a node matrix. `saveTreeMan()` function stores the tree using the serialization format and the node matrix as a hidden .csv. Both parts of the tree can be reloaded to an R environment with `loadTreeMan()`. The hidden node matrix filename is based on the file argument: `file` + `_ndmtrx`

Reading and writing trees with `saveTreeMan()` and `loadTreeMan` is faster than any of the other read and write functions.

See Also

`saveTreeMan`, `readTree`,`writeTree`, `readTrmn`, `writeTrmn`

Examples

```
tree <- randTree(100, wndmtrx = TRUE)
saveTreeMan(tree, file = "test.RData")
rm(tree)
tree <- loadTreeMan(file = "test.RData")
file.remove("test.RData", "testRData_ndmtrx")
```

`mammals` *mammals*

Description

mammals

Format

A TreeMan or Phylota object

Examples

```
data("mammals")
```

`mk_txid_in_sq_mtrx` *Return matrix of txid in sequence*

Description

Searches through lineages of sequences' source organisms to determine whether each txid is represented by the sequence.

Usage

```
mk_txid_in_sq_mtrx(phylota, txids, sids = phylota@sids)
```

Arguments

`phylota`	Phylota
`txids`	Taxonomic IDs
`sids`	Sequence IDs

Value

matrix

See Also

Other tools-private: `summary_phylota()`, `update_phylota()`

`multiPhylo-class` *multiPhylo class*

Description

multiPhylo class

`multiPhylo-to-TreeMen` *Convert multiPhylo to TreeMen*

Description

Return a `TreeMen` from ape's `mutlPhylo`

See Also

`TreeMan-to-phylo`, `phylo-to-TreeMan`, `TreeMen-to-multiPhylo` `TreeMan-class`

Examples

```
library(ape)
trees <- c(rtree(10), rtree(10), rtree(10))
trees <- as(trees, "TreeMen")
```

`ncbicache_load` *Retrieve cached NCBI query*

Description

Run this function to load cached NCBI queries.

Usage

```
ncbicache_load(fnm, args, wd)
```

Arguments

`fnm`	NCBI Entrez function name
`args`	Args used for function
`wd`	Working directory

Value

rentrez result

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`ncbicache_save` *Save NCBI query result to cache*

Description

Run whenever NCBI queries are made to save results in cache in case the pipeline is run again.

Usage

```
ncbicache_save(fnm, args, wd, obj)
```

Arguments

`fnm`	NCBI Entrez function name
`args`	Args used for function
`wd`	Working directory
`obj`	NCBI query result

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`Node-class` *Node-class*

Description

The `Node` is an S4 class used for displaying node information. It is only generated when a user implements the `[[]]` on a tree. Information is only accurate if tree has been updated with `updateTree()`.

Usage

```
## S4 method for signature 'Node'
as.character(x)

## S4 method for signature 'Node'
show(object)

## S4 method for signature 'Node'
print(x)

## S4 method for signature 'Node'
summary(object)

## S4 method for signature 'Node,character,missing,missing'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	Node object
`object`	Node object
`i`	slot name
`j`	missing
`...`	missing
`drop`	missing

Slots

- `id` unique ID for node in tree['ndlst']
- `spn` length of preceding branch
- `prid` parent node ID
- `ptid` child node ID
- `kids` descending tip IDs
- `nkids` number of descending tip IDs
- `txnym` list of associated taxonyms
- `pd` total branch length represented by node
- `prdst` total branch length of connected prids
- `root` T/F root node?
- `tip` T/F tip node?

See Also

`TreeMan-class`, `TreeMen-class`

`obj_check` *Check if an object exists*

Description

Check if an object exists in the cache.

Usage

```
obj_check(wd, nm)
```

Arguments

`wd`	Working directory
`nm`	Object name

Value

T/F

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`obj_load` *Load a named object from the cache*

Description

Loads an object from the cache as stored by `obj_save`.

Usage

```
obj_load(wd, nm)
```

Arguments

`wd`	Working directory
`nm`	Object name

Value

object, multiple formats possible

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`obj_save` *Save a named object in the cache*

Description

Save an object in the cache that can be loaded by `obj_load`.

Usage

```
obj_save(wd, obj, nm)
```

Arguments

`wd`	Working directory
`obj`	Object
`nm`	Object name

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`outfmt_get` *Determine 'outformat' format*

Description

Depending on operating system, BLAST may or may not require "" around `outfmt`. This function will run a micro BLAST analysis to test. It will return the `outfmt` for use in `blastn_run`.

Usage

```
outfmt_get(ps)
```

Arguments

`ps`	Parameters list, generated with parameters()

Value

character

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`parameters` *Default parameters*

Description

Returns a parameter list with default parameter values.

Usage

```
parameters(
  wd = ".",
  txid = character(),
  mkblstdb = "",
  blstn = "",
  v = FALSE,
  ncps = 1,
  mxnds = 1e+05,
  mdlthrs = 3000,
  mnsql = 250,
  mxsql = 2000,
  mxrtry = 100,
  mxsqs = 50000,
  mxevl = 1e-10,
  mncvrg = 51,
  btchsz = 100,
  db_only = FALSE,
  outsider = FALSE,
  srch_trm = paste0("NOT predicted[TI] ", "NOT \"whole genome shotgun\"[TI] ",
    "NOT unverified[TI] ", "NOT \"synthetic construct\"[Organism] ",
    "NOT refseq[filter] NOT TSA[Keyword]"),
  date = Sys.Date()
)
```

Arguments

`wd`	The working directory where all output files are saved.
`txid`	Taxonomic group of interest, allows vectors.
`mkblstdb`	File path to makeblastdb
`blstn`	File path to blastn
`v`	Print progress statements to console? Statements will always be printed to log.txt.
`ncps`	The number of threads to use in the local-alignment search tool.
`mxnds`	The maximum number of nodes descending from a taxonomic group. If there are more than this number, nodes at the lower taxonomic level are analysed.
`mdlthrs`	'Model organism threshold'. Taxa with more sequences than this number will be considered model organisms and a random mdlthrs subset of their sequences will be downloaded.
`mnsql`	The minimum length of sequence in nucleotide base pairs to download.
`mxsql`	The maximum length of sequence in nucleotide base pairs to download. Any longer sequences will be ignored.
`mxrtry`	The maximum number of attempts to make when downloading.
`mxsqs`	The maximum number of sequences to BLAST in all-vs-all searches. If there are more sequences for a node, BLAST is performed at the lower taxonomic level.

`mxevl`	The maximum E-value for a successful BLAST.
`mncvrg`	The maximum percentile coverage defining an overlapping BLAST hit. Sequences with BLAST matches with lower values are not considered orthologous.
`btchsz`	Batch size when querying NCBI
`db_only`	Take sequences only from `restez` library? TRUE/FALSE. If TRUE, internet is still required (for taxonomic look-up) and a `restez` will need to be set up.
`outsider`	Use `om..blast`? TRUE/FALSE. If TRUE, a module for running `blastn` will be installed and all BLAST commands will be run through it. `outsider` package is required.
`srch_trm`	Sequence NCBI search term modifier. Use this parameter to change the default search term options. Default: avoid predicted, WGS, unverified, synthetic, RefSeq and Transcriptome Shotgun Assembly sequences.
`date`	Date when pipeline was initiated

Details

This function is NOT used to change the parameters in a folder. Use parameters_reset() instead. The purpose of this function is to describe the paramaters and present their default values.

Value

list

`parameters_load` *Load parameters from cache*

Description

Parameters are held in cache, use this function to load parameters set for a wd.

Usage

```
parameters_load(wd)
```

Arguments

`wd`	Working directory

Value

Parameters list

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`parameters_reset` *Change parameters in a working directory*

Description

Reset parameters after running `setup()`.

Usage

```
parameters_reset(wd, parameters, values)
```

Arguments

`wd`	Working directory
`parameters`	Parameters to be changed, vector.
`values`	New values for each parameter, vector.

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
## Not run:

  # Note: this example requires BLAST and internet to run.

  # example with temp folder
  wd <- file.path(tempdir(), 'aotus')
  # setup for aotus, make sure aotus/ folder already exists
  if (!dir.exists(wd)) {
```

```
  dir.create(wd)
}
ncbi_dr <- '[SET BLAST+ BIN PATH HERE]'
setup(wd = wd, txid = 9504, ncbi_dr = ncbi_dr)  # txid for Aotus primate genus
# run
# run(wd = wd) # not running in test
# use ctrl+c or Esc to kill
# change parameters, e.g. min and max sequence lengths
parameters_reset(wd = 'aotus', parameters = c('mnsql', 'mxsql'),
                 values = c(300, 1500))
# see ?parameters
# restart
restart(wd = wd)

## End(Not run)
```

`parameters_setup` *Set Up Parameters*

Description

Initiates cache of parameters.

Usage

```
parameters_setup(wd, ncbi_execs, overwrite = FALSE, ...)
```

Arguments

`wd`	Working directory
`ncbi_execs`	File directories for NCBI tools, see `blast_setup()`
`overwrite`	Overwrite existing cache?
`...`	Set parameters, see parameters()

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`parent_get` *Get taxonomic parent*

Description

Look-up MRCA of taxonomic id(s) from taxonomic dictionary

Usage

```
parent_get(id, txdct)
```

Arguments

`id`	txid(s)
`txdct`	TaxDict

Value

Character

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`phylo-class` *phylo class*

Description

phylo class

`phylo-to-TreeMan` *Convert phylo to TreeMan*

Description

Return a `TreeMan` from ape's `phylo`. This will preserve node labels, if they are a alphanumeric.

See Also

`TreeMan-to-phylo`, `TreeMen-to-multiPhylo` `multiPhylo-to-TreeMen` `TreeMan-class`

Examples

```
library(ape)
tree <- compute.brlen(rtree(10))
tree <- as(tree, "TreeMan")
```

`Phylota-class` *Phylota object*

Description

Phylota table contains all sequence, cluster and taxonomic information from a phylotaR pipeline run.

Usage

```
## S4 method for signature 'Phylota'
as.character(x)

## S4 method for signature 'Phylota'
show(object)

## S4 method for signature 'Phylota'
print(x)

## S4 method for signature 'Phylota'
str(object, max.level = 2L, ...)

## S4 method for signature 'Phylota'
summary(object)

## S4 method for signature 'Phylota,character'
x[[i]]
```

Arguments

`x`	`Phylota` object
`object`	`Phylota` object
`max.level`	Maximum level of nesting for str()
`...`	Further arguments for str()
`i`	Either sid or cid

Slots

`cids` IDs of all clusters

`sids` IDs of all sequences

`txids` IDs of all taxa

`sqs` All sequence records as SeqArc

`clstrs` All cluster records as ClstrArc

`txdct` Taxonomic dictionary as TaxDict

`prnt_id` Parent taxonomic ID

`prnt_nm` Parent taxonomic name

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
data('aotus')
# this is a Phylota object
# it contains cluster, sequence and taxonomic information from a phylotaR run
show(aotus)
# you can access its different data slots with @
aotus@cids   # cluster IDs
aotus@sids   # sequence IDs
aotus@txids  # taxonomic IDs
aotus@clstrs # clusters archive
aotus@sqs    # sequence archive
aotus@txdct  # taxonomic dictionary
# see all of the available slots
(slotNames(aotus))
# access different sequences and clusters with [[
(aotus[['0']])              # cluster record 0
(aotus[[aotus@sids[[1]]]])  # first sequence record
# get a summary of the whole object
(summary(aotus))
# the above generates a data.frame with information on each cluster:
# ID - unique id in the object
# Type - cluster type
```

```
# Seed - most connected sequence
# Parent - MRCA of all represented taxa
# N_taxa - number of NCBI recognised taxa
# N_seqs - number of sequences
# Med_sql - median sequence length
# MAD - Maximum alignment density, values close to 1 indicate all sequences in
#  the cluster have a similar length.
# Definition - most common words (and frequency) in sequence definitions
# Feature - most common feature name (and frequency)
```

`pinTips` *Pin tips to a tree*

Description

Returns a tree with new tips added based on given lineages and time points

Usage

```
pinTips(tree, tids, lngs, end_ages, tree_age)
```

Arguments

`tree`	`TreeMan` object
`tids`	new tip ids
`lngs`	list of vectors of the lineages of each tid (ordered high to low rank)
`end_ages`	end time points for each tid
`tree_age`	age of tree

Details

User must provide a vector of new tip IDs, a list of the ranked lineages for these IDs (in ascending order) and a vector of end time points for each new ID (0s for extant tips). The function expects the given tree to be taxonomically informed; the `txnym` slot for every node should have a taxonomic label. The function takes the lineage and tries to randomly add the new tip at the lowest point in the taxonomic rank before the end time point. Note, returned tree will not have a node matrix.

See Also

`addTip`, `rmTips`, `https://github.com/DomBennett/treeman/wiki/manip-methods`

Examples

```
# see https://github.com/DomBennett/treeman/wiki/Pinning-tips for a detailed example
```

`plants` *plants*

Description

plants

Format

A TreeMan or Phylota object

Examples

```
data("plants")
```

`plot_phylota_pa` *Plot presence/absence matrix*

Description

Plot presence/absence of taxa by each cluster in phylota object.

Usage

```
plot_phylota_pa(phylota, cids, txids, cnms = cids, txnms = txids)
```

Arguments

`phylota`	Phylota object
`cids`	Vector of cluster IDs
`txids`	Vector of taxonomic IDs
`cnms`	Cluster names
`txnms`	Taxonomic names

Details

Cluster names and taxonomic names can be given to the function, by default IDs are used.

Value

geom_object

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_treemap()`, `read_phylota()`, `write_sqs()`

Examples

```
library(phylotaR)
data(cycads)
# drop all but first ten
cycads <- drop_clstrs(cycads, cycads@cids[1:10])
# plot all
p <- plot_phylota_pa(phylota = cycads, cids = cycads@cids, txids = cycads@txids)
print(p)  # lots of information, difficult to interpret
# get genus-level taxonomic names
genus_txids <- get_txids(cycads, txids = cycads@txids, rnk = 'genus')
genus_txids <- unique(genus_txids)
# dropping missing
genus_txids <- genus_txids[genus_txids !=  '']
genus_nms <- get_tx_slot(cycads, genus_txids, slt_nm = 'scnm')
# make alphabetical for plotting
genus_nms <- sort(genus_nms, decreasing = TRUE)
# generate geom_object
p <- plot_phylota_pa(phylota = cycads, cids = cycads@cids, txids = genus_txids,
                     txnms = genus_nms)
# plot
print(p)  # easier to interpret
```

`plot_phylota_treemap` *Plot treemap of Phylota object*

Description

Treemaps show relative size with boxes. The user can explore which taxa or clusters are most represented either by sequence or cluster number. If cluster IDs are provided, the plot is made for clusters. If taxonomic IDs are provided, the plot is made for taxa.

Usage

```
plot_phylota_treemap(
  phylota,
  cids = NULL,
  txids = NULL,
  cnms = cids,
  txnms = txids,
  with_labels = TRUE,
  area = c("ntx", "nsq", "ncl"),
  fill = c("NULL", "typ", "ntx", "nsq", "ncl")
)
```

Arguments

`phylota`	Phylota object
`cids`	Cluster IDs
`txids`	Taxonomic IDs
`cnms`	Cluster names
`txnms`	Taxonomic names
`with_labels`	Show names per box?
`area`	What determines the size per box?
`fill`	What determines the coloured fill per box?

Details

The function can take a long time to run for large Phylota objects over many taxonomic IDs because searches are made across lineages. The idea of the function is to assess the data dominance of specific clusters and taxa.

Value

geom_object

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `read_phylota()`, `write_sqs()`

Examples

```
data("tinamous")
# Plot clusters, size by n. sq, fill by n. tx
p <- plot_phylota_treemap(phylota = tinamous, cids = tinamous@cids,
                          area = 'nsq', fill = 'ntx')
print(p)
# Plot taxa, size by n. sq, fill by ncl
txids <- get_txids(tinamous, txids = tinamous@txids, rnk = 'genus')
txids <- txids[txids !=  '']
txids <- unique(txids)
txnms <- get_tx_slot(tinamous, txids, slt_nm = 'scnm')
p <- plot_phylota_treemap(phylota = tinamous, txids = txids, txnms = txnms,
                          area = 'nsq', fill = 'ncl')
print(p)
```

`progress_init` *Initialise progress list in cache*

Description

Creates a progress list recording each stage run in cache.

Usage

```
progress_init(wd)
```

Arguments

`wd`	Working directory

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`progress_read` *Read the progress from cache*

Description

Return the last completed stage using the cache.

Usage

```
progress_read(wd)
```

Arguments

`wd`	Working directory

Value

stage name, character, or NA is complete

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`progress_reset` *Reset progress*

Description

Reset progress to an earlier completed stage.

Usage

```
progress_reset(wd, stg)
```

Arguments

`wd`	Working directory
`stg`	Stage to which the pipeline will be reset

Details

For example, resetting the progress to ’download’ mark stages ’download’, ’cluster’ and ’cluster2’ as un-run.

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`progress_save` *Save current progress*

Description

Stores the pipeline progress in the cache.

Usage

```
progress_save(wd, stg)
```

Arguments

`wd`	Working directory
`stg`	Stage

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`pstMnp` *Update prinds and tinds*

Description

Return tree with updated slots.

Usage

```
pstMnp(tree)
```

Arguments

`tree`	`TreeMan` object

Details

This function is automatically run. Only run, if you are creating yor own functions to add and remove elements of the `ndlst`.

See Also

`updateSlts`, `addNdmtrx`, `getAge`

`randTree` *Generate a random tree*

Description

Returns a random `TreeMan` tree with `n` tips.

Usage

```
randTree(n, wndmtrx = FALSE, parallel = FALSE)
```

Arguments

`n`	number of tips, integer, must be 3 or greater
`wndmtrx`	T/F add node matrix? Default FALSE.
`parallel`	T/F run in parallel? Default FALSE.

Details

Equivalent to `ape`'s `rtree()` but returns a `TreeMan` tree. Tree is always rooted and bifurcating.

See Also

TreeMan-class, blncdTree, unblncdTree

Examples

```
tree <- randTree(5)
```

`rank_get` *Get rank*

Description

Look-up taxonomic rank from dictionary.

Usage

```
rank_get(txid, txdct)
```

Arguments

`txid`	txid
`txdct`	TaxDict

Value

character

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`rawseqrec_breakdown` *Breakdown a sequence record into its features*

Description

Takes GenBank record's elements and returns a SeqRec. For sequences with lots of features, the sequence is broken down into these features provided they are of the right size. Sequences are either returned as features or whole sequence records, never both.

Usage

```
rawseqrec_breakdown(record_parts, ps)
```

Arguments

`record_parts`	list of record elements from a GenBank record
`ps`	Parameters list, generated with parameters()

Value

list of SeqRecs

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`readTree` *Read a Newick tree*

Description

Return a `TreeMan` or `TreeMen` object from a Newick treefile

Usage

```
readTree(
  file = NULL,
  text = NULL,
  spcl_slt_nm = "Unknown",
  wndmtrx = FALSE,
  parallel = FALSE,
  progress = "none"
)
```

Arguments

`file`	file path
`text`	Newick character string
`spcl_slt_nm`	name of special slot for internal node labels, default 'Unknown'.
`wndmtrx`	T/F add node matrix? Default FALSE.
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Read a single or multiple trees from a file, or a text string. Parallelizable when reading multiple trees. The function will add any internal node labels in the Newick tree as a user-defined data slots. The name of this slot is defined with the `spcl_slt_nm`. These data can be accessed/manipulated with the `` `getNdsSlt()` `` function. Trees are always read as rooted. (Unrooted trees have polytomous root nodes.)

See Also

`https://en.wikipedia.org/wiki/Newick_format`, `addNdmtrx`, `writeTree`, `randTree`, `readTrmn`, `writeTrmn`, `saveTreeMan`, `loadTreeMan`

Examples

```
# tree string with internal node labels as bootstrap results
tree <- readTree(
  text = "((A:1.0,B:1.0)0.9:1.0,(C:1.0,D:1.0)0.8:1.0)0.7:1.0;",
  spcl_slt_nm = "bootstrap"
)
```

```
# retrieve bootstrap values by node
tree["bootstrap"]
```

`readTrmn` *Read a .trmn tree*

Description

Return a `TreeMan` or `TreeMen` object from a .trmn treefile

Usage

```
readTrmn(file, wndmtrx = FALSE, parallel = FALSE, progress = "none")
```

Arguments

`file`	file path
`wndmtrx`	T/F add node matrix? Default FALSE.
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Read a tree(s) from a file using the .trmn format. It is faster to read and write tree files using treeman with the .trmn file format. In addition it is possible to encode more information than possible with the Newick, e.g. any taxonomic information and additional slot names added to the tree are recorded in the file.

See Also

`writeTrmn`, `readTree`,`writeTree`, `randTree`, `saveTreeMan`, `loadTreeMan`

Examples

```
tree <- randTree(10)
writeTrmn(tree, file = "test.trmn")
tree <- readTrmn("test.trmn")
file.remove("test.trmn")
```

`read_phylota` *Generate a Phylota object in R*

Description

Creates a Phylota object containing information on clusters, sequences and taxonomy from the working directory of a completed pipeline.

Usage

```
read_phylota(wd)
```

Arguments

`wd`	Working directory

Value

Phylota

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `write_sqs()`

Examples

```
## Not run:

  # Note, this example requires a wd with a completed phylotaR run
  phylota <- read_phylota(wd)

## End(Not run)
```

`reset` *Reset a phylotaR pipeline run*

Description

Resets the pipeline to a specified stage.

Usage

```
reset(wd, stage, hard = FALSE)
```

Arguments

`wd`	Working directory
`stage`	Name of stage to which the pipeline will be reset
`hard`	T/F, delete all cached data?

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
## Not run:

  # Note: this example requires BLAST and internet to run.

  # example with temp folder
  wd <- file.path(tempdir(), 'aotus')
  # setup for aotus, make sure aotus/ folder already exists
  if (!dir.exists(wd)) {
    dir.create(wd)
  }
  ncbi_dr <- '[SET BLAST+ BIN PATH HERE]'
  setup(wd = wd, txid = 9504, ncbi_dr = ncbi_dr)  # txid for Aotus primate genus
  # individually run taxise
  taxise_run(wd = wd)
  # reset back to taxise as if it has not been run
  reset(wd = 'aotus', stage = 'taxise')
  # run taxise again ....
  taxise_run(wd = wd)

## End(Not run)
```

`restart` *Restart a phylotaR pipeline run*

Description

Restarts the running of a pipeline as started with `run`.

Usage

```
restart(wd, nstages = 4)
```

Arguments

`wd`	Working directory
`nstages`	Number of total stages to run, max 4.

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `run()`, `setup()`, `taxise_run()`

Examples

```
## Not run:

  # Note: this example requires BLAST and internet to run.

  # example with temp folder
  wd <- file.path(tempdir(), 'aotus')
  # setup for aotus, make sure aotus/ folder already exists
  if (!dir.exists(wd)) {
    dir.create(wd)
  }
  ncbi_dr <- '[SET BLAST+ BIN PATH HERE]'
  setup(wd = wd, txid = 9504, ncbi_dr = ncbi_dr)  # txid for Aotus primate genus
  # run and stop after 10 seconds
  R.utils::withTimeout(expr = {
    run(wd = wd)
  }, timeout = 10)
  # use ctrl+c or Esc to kill without a timelimit
  # and restart with ....
  restart(wd = wd)

## End(Not run)
```

`rmClade` *Remove a clade from a tree*

Description

Returns a tree with a clade removed

Usage

```
rmClade(tree, id)
```

Arguments

`tree`	TreeMan object
`id`	node ID parent of clade to be removed

Details

Inverse function of `getSubtree()`. Takes a tree and removes a clade based on an internal node specified. Node is specified with `id`, all descending nodes and tips are removed. The resulting tree will replace the missing clade with a tip of `id`.

See Also

addClade, getSubtree, rmTips https://github.com/DomBennett/treeman/wiki/manip-methods

Examples

```
t1 <- randTree(100)
# remove a clade
t2 <- rmClade(t1, "n2")
summary(t1)
summary(t2)
```

---

rmNdmtrx *Remove node matrix*

Description

Return tree with memory heavy node matrix removed.

Usage

```
rmNdmtrx(tree)
```

Arguments

`tree`	TreeMan object

Details

Potential uses: reduce memory load of a tree, save tree using serialization methods.

See Also

addNdmtrx

Examples

```
#
tree <- randTree(10)
summary(tree)
tree <- rmNdmtrx(tree)
summary(tree)
```

`rmNodes` *Remove nodes from a tree*

Description

Returns a tree with a node ID(s) removed

Usage

```
rmNodes(tree, nids, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`nids`	internal node IDs
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Removes nodes in a tree. Joins the nodes following to the nodes preceding the node to be removed. Creates polytomies. Warning: do not use this function to remove tip nodes, this create a corrupted tree.

See Also

`addTip`, `rmTips`, `https://github.com/DomBennett/treeman/wiki/manip-methods`

Examples

```
tree <- randTree(10)
tree <- rmNodes(tree, "n3")
summary(tree) # tree is now polytmous
```

`rmOtherSlt` *Remove a user-defined slot*

Description

Returns a tree with a user-defined tree slot removed.

Usage

```
rmOtherSlt(tree, slt_nm)
```

Arguments

`tree`	`TreeMan` object
`slt_nm`	name of slot to be removed

Details

A user can specify a new slot using the `setNdSlt()` function or upon reading a tree. This can be removed using this function by specifying the name of the slot to be removed.

See Also

`setNdOther`, `setNdsOther`, `https://github.com/DomBennett/treeman/wiki/set-methods`

Examples

```
tree <- randTree(10)
vals <- runif(min = 0, max = 1, n = tree["nall"])
tree <- setNdsOther(tree, tree["all"], vals, "confidence")
tree <- updateSlts(tree)
summary(tree)
tree <- rmOtherSlt(tree, "confidence")
tree <- updateSlts(tree)
summary(tree)
```

`rmTips` *Remove tips from a tree*

Description

Returns a tree with a tip ID(s) removed

Usage

```
rmTips(tree, tids, drp_intrnl = TRUE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`tids`	tip IDs
`drp_intrnl`	Boolean, drop internal branches, default FALSE
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Removes tips in a tree. Set drp_intrnl to FALSE to convert internal nodes into new tips. Warning: do not use this function to remove internal nodes, this create a corrupted tree.

See Also

`addTip`, `rmNodes`, `https://github.com/DomBennett/treeman/wiki/manip-methods`

Examples

```
tree <- randTree(10)
tree <- rmTips(tree, "t1")
summary(tree)
# running the function using an internal
# node will create a corrupted tree
tree <- rmTips(tree, "n3")
# run summary() to make sure a change has
# not created a corruption
# summary(tree)
```

`run` *Run phylotaR pipeline*

Description

Run the entire phylotaR pipeline. All generated files will be stored in the wd. The process can be stopped at anytime and restarted with `restart`. `nstages` must be a numeric value representing the number of stages that will be run. Stages are run in the following order: 1 - taxise, 2 - download, 3 - cluster and 4 - cluster2.

For example, specifying `nstages` = 3, will run taxise, download and cluster. Stages can also be run individually, see linked functions below.

Usage

```
run(wd, nstages = 4)
```

Arguments

`wd`	Working directory
`nstages`	Number of total stages to run, max 4.

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `setup()`, `taxise_run()`

Examples

```
## Not run:

  # Note: this example requires BLAST and internet to run.

  # example with temp folder
  wd <- file.path(tempdir(), 'aotus')
  # setup for aotus, make sure aotus/ folder already exists
  if (!dir.exists(wd)) {
    dir.create(wd)
  }
  ncbi_dr <- '[SET BLAST+ BIN PATH HERE]'
  # e.g. "/usr/local/ncbi/blast/bin/"
  setup(wd = wd, txid = 9504, ncbi_dr = ncbi_dr)  # txid for Aotus primate genus
  run(wd = wd)

## End(Not run)
```

`safely_connect` *Safely run rentrez function*

Description

Safely run a rentrez function. If the query fails, the function will retry.

Usage

```
safely_connect(func, args, fnm, ps)
```

Arguments

`func`	rentrez function
`args`	rentrez function arguments, list
`fnm`	rentrez function name
`ps`	Parameters list, generated with parameters()

Value

rentrez function results

See Also

Other run-private: `batcher`(), `blast_clstr`(), `blast_filter`(), `blast_setup`(), `blast_sqs`(), `blastcache_load`(), `blastcache_save`(), `blastdb_gen`(), `blastn_run`(), `cache_rm`(), `cache_setup`(), `clade_select`(), `clstr2_calc`(), `clstr_all`(), `clstr_direct`(), `clstr_sqs`(), `clstr_subtree`(), `clstrarc_gen`(), `clstrarc_join`(), `clstrrec_gen`(), `clstrs_calc`(), `clstrs_join`(), `clstrs_merge`(), `clstrs_renumber`(), `clstrs_save`(), `cmdln`(), `descendants_get`(), `download_obj_check`(), error(), gb_extract(), hierarchic_download(), info(), ncbicache_load(), ncbicache_save(), obj_check(), obj_load(), obj_save(), outfmt_get(), parameters_load(), parameters_setup(), parent_get(), progress_init(), progress_read(), progress_reset(), progress_save(), rank_get(), rawseqrec_breakdown(), search_and_cache(), searchterm_gen(), seeds_blast(), seq_download(), seqarc_gen(), seqrec_augment(), seqrec_convert(), seqrec_gen(), seqrec_get(), sids_check(), sids_get(), sids_load(), sids_save(), sqs_count(), sqs_save(), stage_args_check(), stages_run(), tax_download(), taxdict_gen(), taxtree_gen(), txids_get(), txnds_count(), warn()

`saveTreeMan` *Save a TreeMan object in serialization format*

Description

`TreeMan` equivalent to `save()` but able to handle node matrices.

Usage

```
saveTreeMan(tree, file)
```

Arguments

`tree`	`TreeMan` object
`file`	file path

Details

It is not possible to use `save()` on `TreeMan` objects with node matrices. Node matrices are bigmemory matrices and are therefore outside the R environment, see bigmemory documentation for more information. Saving and loading a bigmemory matrix may cause memory issues in R and cause R to crash.

This function can safely store a `TreeMan` object with and without a node matrix. This function stores the tree using the serialization format and the node matrix as a hidden .csv. Both parts of the tree can be reloaded to an R environment with `loadTreeMan()`. The hidden node matrix filename is based on the file argument: `file` + `_ndmtrx`

Reading and writing trees with `saveTreeMan()` and `loadTreeMan` is faster than any of the other read and write functions.

See Also

loadTreeMan, readTree,writeTree, readTrmn, writeTrmn

Examples

```
tree <- randTree(100, wndmtrx = TRUE)
saveTreeMan(tree, file = "test.RData")
rm(tree)
tree <- loadTreeMan(file = "test.RData")
file.remove("test.RData", "testRData_ndmtrx")
```

`searchterm_gen` *Construct GenBank Search Term*

Description

Construct search term for searching GenBank's nucleotide database. Limits the maximum size of sequences, avoids whole genome shotguns, predicted, unverified and synthetic sequences.

Usage

```
searchterm_gen(txid, ps, direct = FALSE)
```

Arguments

`txid`	Taxonomic ID
`ps`	Parameters list, generated with parameters()
`direct`	Node-level only or subtree as well? Default FALSE.

Value

character, search term

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`searchTxnyms` *Get node labels based on online taxonomic database*

Description

Return names of each node in tree based on searching tip labels through Global Names Resolver `http://resolver.globalnames.org/` in NCBI.

Usage

```
searchTxnyms(tree, cache = FALSE, parent = NULL, clean = TRUE, infer = TRUE)
```

Arguments

`tree`	TreeMan object
`cache`	T/F, create a local cache of downloaded names?
`parent`	specify parent of all names to prevent false names
`clean`	T/F, ensure returned names contain no special characters?
`infer`	T/F, infer taxonyms for unfound nodes?

Details

For each node, all the descendants are searched, the taxonomic lineages returned and then searched to find the lowest shared name. All the tip labels are searched against a specified taxonomic database through the GNR and NCBI. (So far only tested with NCBI database.) Use the infer argument to ensure a taxonym is returned for all nodes. If infer is true, all nodes without an identifed taxonym adopt the taxonym of their parent. Will raise a warning if connection fails and will return NULL.

See Also

`taxaResolve`, `setTxnyms`, `getNdsFrmTxnyms`

Examples

```
tree <- randTree(8)
new_tids <- c(
  "Gallus_gallus", "Aileuropoda_melanoleucha", "Ailurus_fulgens",
  "Rattus_rattus", "Mus_musculus", "Gorilla_gorilla", "Pan_trogoldytes", "Homo_sapiens"
)
tree <- setNdsID(tree, tree["tips"], new_tids)
nd_labels <- searchTxnyms(tree)
print(nd_labels)
```

`search_and_cache` *Run rentrez function and cache results*

Description

Safely run a rentrez function. If the query fails, the function will retry. All query results are cached. To remove cached data use hard reset.

Usage

```
search_and_cache(func, args, fnm, ps)
```

Arguments

`func`	rentrez function
`args`	rentrez function arguments, list
`fnm`	rentrez function name
`ps`	Parameters list, generated with parameters()

Value

rentrez function results

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`seeds_blast` *BLAST seed sequences*

Description

Runs all-v-all blast for seed sequences.

Usage

```
seeds_blast(sqs, ps)
```

Arguments

`sqs`	All seed sequences to be BLASTed
`ps`	Parameters list, generated with parameters()

Value

blast res data.frame

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`SeqArc-class` *Sequence record archive*

Description

Multiple sequence records containing sequence data.

Usage

```
## S4 method for signature 'SeqArc'
as.character(x)

## S4 method for signature 'SeqArc'
show(object)

## S4 method for signature 'SeqArc'
print(x)

## S4 method for signature 'SeqArc'
str(object, max.level = 2L, ...)

## S4 method for signature 'SeqArc'
summary(object)

## S4 method for signature 'SeqArc,character'
x[[i]]

## S4 method for signature 'SeqArc,character,missing,missing'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	`SeqArc` object
`object`	`SeqArc` object
`max.level`	Maximum level of nesting for str()
`...`	Further arguments for str()
`i`	sid(s)
`j`	Unused
`drop`	Unused

Details

Sequences are stored as raw. Use rawToChar().

Slots

`ids`	Vector of Sequence Record IDs
`nncltds`	Vector of sequence lengths
`nambgs`	Vector of number of ambiguous nucleotides
`txids`	Vector source txid associated with each sequence
`sqs`	List of SeqRecs named by ID

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
data('aotus')
seqarc <- aotus@sqs
# this is a SeqArc object
# it contains sequence records
show(seqarc)
# you can access its different data slots with @
seqarc@ids     # sequence IDs defined as accession + feature position
seqarc@nncltds # number of nucleotides of all sequences
seqarc@nambgs  # number of ambiguous nucleotides of all sequences
seqarc@txids   # all the taxonomic IDs for all sequences
seqarc@sqs     # list of all SeqRecs
# access sequence records [[
(seqarc[[seqarc@ids[[1]]]])  # first sequence record
# generate new sequence archives with [
(seqarc[seqarc@ids[1:10]])  # first 10 sequences
```

`seqarc_gen` *Generate sequence archive*

Description

Creates an S4 SeqArc from list of SeqRecs

Usage

```
seqarc_gen(seqrecs)
```

Arguments

`seqrecs`	List of SeqRecs

Value

SeqArc

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`SeqRec-class` *Sequence record*

Description

Sequence record contains sequence data.

Usage

```
## S4 method for signature 'SeqRec'
as.character(x)

## S4 method for signature 'SeqRec'
show(object)

## S4 method for signature 'SeqRec'
print(x)

## S4 method for signature 'SeqRec'
str(object, max.level = 2L, ...)

## S4 method for signature 'SeqRec'
summary(object)
```

Arguments

`x`	SeqRec object
`object`	SeqRec object
`max.level`	Maximum level of nesting for str()
`...`	Further arguments for str()

Details

Sequence is stored as raw. Use rawToChar().

Slots

- `id` Unique ID
- `nm` Best-guess sequence name
- `accssn` Accession
- `vrsn` Accession version
- `url` URL
- `txid` Taxonomic ID of source taxon
- `orgnsm` Scientific name of source taxon
- `sq` Sequence
- `dfln` Definition line
- `ml_typ` Molecule type, e.g. DNA
- `rec_typ` Record type: Whole or feature
- `nncltds` Number of nucleotides
- `nambgs` Number of ambiguous nucleotides
- `pambgs` Proportion of ambiguous nucleotides
- `gcr` GC ratio
- `age` Number of days between sequence upload and running pipeline

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
data('aotus')
seqrec <- aotus@sqs@sqs[[1]]
# this is a SeqRec object
# it contains sequence records
show(seqrec)
# you can access its different data slots with @
seqrec@id       # sequence ID, accession + feature location
seqrec@nm       # feature name, '' if none
seqrec@accssn   # accession
seqrec@vrsn     # accession version
seqrec@url      # NCBI GenBank URL
seqrec@txid     # Taxonomic ID
seqrec@orgnsm   # free-text organism name
seqrec@sq       # sequence, in raw format
seqrec@dfln     # sequence definition
seqrec@ml_typ   # molecule type
seqrec@rec_typ  # whole record or feature
seqrec@nncltds  # sequence length
seqrec@nambgs   # number of non-ATCGs
```

```
seqrec@pambgs   # proportion of non-ATCGs
seqrec@gcr      # GC-ratio
seqrec@age      # days since being added to GenBank
# get the sequence like so....
(rawToChar(seqrec@sq))
```

`seqrec_augment` *Augment sequence records list*

Description

Add taxids to records and convert to archive.

Usage

```
seqrec_augment(sqs, txdct)
```

Arguments

`sqs`	List of SeqRecs
`txdct`	Taxonomic Dictionary

Value

SeqArc

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`seqrec_convert` *Convert raw Entrez gb text record to SeqRecs*

Description

Parses returned sequences features with Entrez, returns one or more SeqRec objects for each raw record.

Usage

```
seqrec_convert(raw_recs, ps)
```

Arguments

`raw_recs`	Raw text records returned from Entrez fetch
`ps`	Parameters list, generated with parameters()

Value

SeqRecs

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`seqrec_gen` *Generate sequence record*

Description

Creates an S4 SeqRec

Usage

```
seqrec_gen(
  accssn,
  nm,
  txid,
  sq,
  dfln,
  orgnsm,
  ml_typ,
  rec_typ,
  vrsn,
  age,
  lctn = NULL
)
```

Arguments

`accssn`	Accession ID
`nm`	Sequence name
`txid`	Taxonomic ID of source organism
`sq`	Sequence
`dfln`	Definition line
`orgnsm`	Source organism name
`ml_typ`	Molecule type
`rec_typ`	Sequence record type
`vrsn`	Accession version
`age`	Number of days since upload
`lctn`	Location numbers for features, e.g. '1..200'

Value

SeqRec

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`seqrec_get` *seqrec_get*

Description

Downloads sequences from GenBank in batches.

Usage

```
seqrec_get(txid, ps, direct = FALSE, lvl = 0)
```

Arguments

`txid`	NCBI taxonomic ID
`ps`	Parameters list, generated with parameters()
`direct`	Node-level only or subtree as well? Default FALSE.
`lvl`	Integer, number of message indentations indicating code depth.

Details

If a restez database is available and the number of sequences to retrieve is less than 'btchsz', the function will look the sequences up from the database rather than download.

Value

Vector of sequence records

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`seq_download` *Download sequences for txids*

Description

Look up and download all sequences for given taxonomic IDs.

Usage

```
seq_download(txids, txdct, ps)
```

Arguments

`txids`	Taxonomic node IDs, numeric vector
`txdct`	Taxonomic dictionary
`ps`	Parameters list, generated with parameters()

Details

Sequence downloads are cached.

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`setAge` *Set the age of a tree*

Description

Return a tree with the age altered.

Usage

```
setAge(tree, val)
```

Arguments

`tree`	`TreeMan` object
`val`	new age

Details

Use this function to change the age of a tree. For example, you might want to convert the tree so that its age equals 1. This function will achieve that by modiyfing every branch, while maintaining their relative lengths.

See Also

`setPD` `https://github.com/DomBennett/treeman/wiki/set-methods`

Examples

```
tree <- randTree(10)
tree <- setAge(tree, val = 1)
summary(tree)
```

`setNdID` *Set the ID of a node*

Description

Return a tree with the ID of a node altered.

Usage

```
setNdID(tree, id, val)
```

Arguments

`tree`	`TreeMan` object
`id`	id to be changed
`val`	new id

Details

IDs cannot be changed directly for the `TreeMan` class. To change an ID use this function. Warning: all IDs must be unique, avoid spaces in IDs and only use letters, numbers and underscores. Use `updateSlts` after running.

See Also

`setNdsID` `https://github.com/DomBennett/treeman/wiki/set-methods`

Examples

```
tree <- randTree(10)
tree <- setNdID(tree, "t1", "heffalump")
tree <- updateSlts(tree)
```

`setNdOther` *Set a user defined slot*

Description

Return a tree with a user defined slot for node ID.

Usage

```
setNdOther(tree, id, val, slt_nm)
```

Arguments

`tree`	`TreeMan` object
`id`	id of the node
`val`	data for slot
`slt_nm`	slot name

Details

A user can specify new slots in a tree. Add a new slot with this function by providing a node ID, a value for the new slot and a unique new slot name. Slot names must not be default `TreeMan` names. The new value can be any data type.

See Also

setNdsOther https://github.com/DomBennett/treeman/wiki/set-methods

Examples

```
tree <- randTree(10)
tree <- setNdOther(tree, "t1", 1, "binary_val")
tree <- updateSlts(tree)
(getNdSlt(tree, id = "t1", slt_nm = "binary_val"))
```

---

setNdsID *Set the IDs of multiple nodes*

Description

Return a tree with the IDs of nodes altered.

Usage

```
setNdsID(tree, ids, vals, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	ids to be changed
`vals`	new ids
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Runs `setNdID()` over multiple nodes. Warning: all IDs must be unique, avoid spaces in IDs, only use numbers, letters and underscores. Parellizable.

See Also

setNdID https://github.com/DomBennett/treeman/wiki/set-methods

Examples

```
tree <- randTree(10)
new_ids <- paste0("heffalump_", 1:tree["ntips"])
tree <- setNdsID(tree, tree["tips"], new_ids)
summary(tree)
```

`setNdsOther` *Set a user defined slot for multiple nodes*

Description

Return a tree with a user defined slot for node IDs.

Usage

```
setNdsOther(tree, ids, vals, slt_nm, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	id sof the nodes
`vals`	data for slot
`slt_nm`	slot name
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Runs `setNdOther()` over multiple nodes. Parellizable.

See Also

`setNdOther` `https://github.com/DomBennett/treeman/wiki/set-methods`

Examples

```
tree <- randTree(10)
# e.g. confidences for nodes
vals <- runif(min = 0, max = 1, n = tree["nall"])
tree <- setNdsOther(tree, tree["all"], vals, "confidence")
tree <- updateSlts(tree)
summary(tree)
(getNdsSlt(tree, ids = tree["all"], slt_nm = "confidence"))
```

`setNdSpn` *Set the branch length of a specific node*

Description

Return a tree with the span of a node altered.

Usage

```
setNdSpn(tree, id, val)
```

Arguments

`tree`	`TreeMan` object
`id`	id of node whose preceding edge is to be changed
`val`	new span

Details

Takes a tree, a node ID and a new value for the node's preceding branch length (span).

See Also

`setNdsSpn` `https://github.com/DomBennett/treeman/wiki/set-methods`

Examples

```
tree <- randTree(10)
tree <- setNdSpn(tree, id = "t1", val = 100)
tree <- updateSlts(tree)
summary(tree)
```

`setNdsSpn` *Set the branch lengths of specific nodes*

Description

Return a tree with the spans of nodes altered.

Usage

```
setNdsSpn(tree, ids, vals, parallel = FALSE, progress = "none")
```

Arguments

`tree`	`TreeMan` object
`ids`	ids of nodes whose preceding edges are to be changed
`vals`	new spans
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

Runs `setNdSpn` over multiple nodes. Parallelizable.

See Also

`setNdSpn` `https://github.com/DomBennett/treeman/wiki/set-methods`

Examples

```
tree <- randTree(10)
# make tree taxonomic
tree <- setNdsSpn(tree, ids = tree["all"], vals = 1)
summary(tree)
# remove spns by setting all to 0
tree <- setNdsSpn(tree, ids = tree["all"], vals = 0)
summary(tree)
```

`setPD` *Set the phylogenetic diversity*

Description

Return a tree with the phylogenetic diversity altered.

Usage

```
setPD(tree, val)
```

Arguments

`tree`	`TreeMan` object
`val`	new phylogenetic diversity

Details

Use this function to convert the phylogenetic diversity of a tree. For example, you might want to convert the tree so the sum of all branches is 1. This function will achieve that by modiyfing every branch, while maintaining their relative lengths.

See Also

`setAge` `https://github.com/DomBennett/treeman/wiki/set-methods`

Examples

```
tree <- randTree(10)
tree <- setPD(tree, val = 1)
summary(tree)
```

`setTxnyms` *Set the txnym slots in a tree*

Description

Return a tree with txnyms added to specified nodes

Usage

```
setTxnyms(tree, txnyms)
```

Arguments

`tree`	`TreeMan` object
`txnyms`	named vector or list

Details

Returns a tree. Specify the taxonomic groups for nodes in a tree by providing a vector or list named by node IDs. Takes output from `searchTxnyms`. Only letters, numbers and underscores allowed. To remove special characters use regular expressions, e.g. `gsub(['a-zA-Z0-9_'], '', txnym)`

See Also

`taxaResolve`, `searchTxnyms`, `getNdsLng`, `getNdLng`, `https://github.com/DomBennett/treeman/wiki/set-methods`

Examples

```
data(mammals)
# let's change the txnym for humans
# what's its summary before we change anything?
summary(mammals[["Homo_sapiens"]])
# now let's add Hominini
new_txnym <- list("Homo_sapiens" = c("Hominini", "Homo"))
mammals <- setTxnyms(mammals, new_txnym)
summary(mammals[["Homo_sapiens"]])
```

`setup` *Set-up parameters*

Description

Set up working directory with parameters.

Usage

```
setup(
  wd,
  txid,
  ncbi_dr = ".",
  v = FALSE,
  overwrite = FALSE,
  outsider = FALSE,
  ...
)
```

Arguments

`wd`	Working directory
`txid`	Root taxonomic ID(s), vector or numeric
`ncbi_dr`	Directory to NCBI BLAST tools, default '.'
`v`	Verbose, T/F
`overwrite`	Overwrite existing cache?
`outsider`	Run through `outsider`? T/F
`...`	Additional parameters

Details

See `parameters`() for a description of all parameters and their defaults. You can change parameters after a folder has been set up with `parameters_reset`().

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `taxise_run()`

Examples

```
## Not run:

  # Note: this example requires BLAST to run.

  # example with temp folder
  wd <- file.path(tempdir(), 'aotus')
  # setup for aotus, make sure aotus/ folder already exists
  if (!dir.exists(wd)) {
    dir.create(wd)
  }
  ncbi_dr <- '[SET BLAST+ BIN PATH HERE]'
  # e.g. "/usr/local/ncbi/blast/bin/"
  setup(wd = wd, txid = 9504, ncbi_dr = ncbi_dr)  # txid for Aotus primate genus
  # see ?parameters for all available parameter options

## End(Not run)
```

`sids_check` *Check if sids exist*

Description

Check if sids are already downloaded for a txid.

Usage

```
sids_check(wd, txid)
```

Arguments

`wd`	Working directory
`txid`	Taxonomic ID, numeric

Details

#' @name sqs_load #' @title Load sequences from cache #' @description Load sequences downloaded by `dwnld` function. #' @param wd Working directory #' @param txid Taxonomic ID, numeric #' @family run-private #' @return SeqArc sqs_load <- function(wd, txid) d <- file.path(wd, 'cache') if (!file.exists(d)) stop('Cache does not exist.')

d <- file.path(d, 'sqs') if (!file.exists(d)) stop('‘sqs‘ not in cache. Have you run the download stage?')

fl <- file.path(d, paste0(txid, '.RData')) if (!file.exists(fl)) stop(paste0('[', txid, '] not in ‘sqs‘ of cache.'))

sqs <- try(readRDS(file = fl), silent = TRUE) if (inherits(sqs, 'try-error')) file.remove(fl)

sqs

Value

T/F

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`sids_get` *Return random set of sequence IDs*

Description

For a given txid return a random set of sequences associated.

Usage

```
sids_get(txid, direct, ps, retmax = 100, hrdmx = 1e+05)
```

Arguments

`txid`	NCBI taxon identifier
`direct`	Node-level only or subtree as well? Default FALSE.
`ps`	Parameters list, generated with parameters()
`retmax`	Maximum number of sequences when querying model organisms. The smaller the more random, the larger the faster.
`hrdmx`	Absolute maximum number of sequence IDs to download in a single query.

Details

For model organisms downloading all IDs can a take long time or even cause an xml parsing error. For any search with more than hrdmx sequences, this function we will run multiple small searches downloading retmax seq IDs at a time with different retstart values to generate a semi-random vector of sequence IDs. For all other searches, all IDs will be retrieved. Note, it makes no sense for mdlthrs in parameters to be greater than hrdmx in this function.

Value

vector of IDs

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`sids_load` *Load sids from cache*

Description

Load sids downloaded by `sids_get` function.

Usage

```
sids_load(wd, txid)
```

Arguments

`wd`	Working directory
`txid`	Taxonomic ID, numeric

Value

vector of sids

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`sids_save` *Save sids to cache*

Description

Saves sids downloaded

Usage

```
sids_save(wd, txid, sids)
```

Arguments

`wd`	Working directory
`txid`	Taxonomic ID, numeric
`sids`	sids

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`sqs_count` *Count number of sequences for txid*

Description

Return the number of sequences associated with a taxonomic ID on NCBI GenBank.

Usage

```
sqs_count(txid, ps, direct = FALSE)
```

Arguments

`txid`	Taxonomic ID
`ps`	Parameters list, generated with parameters()
`direct`	Node-level only or subtree as well? Default FALSE.

Value

integer

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`sqs_save` *Save sequences to cache*

Description

Saves sequences downloaded

Usage

```
sqs_save(wd, txid, sqs)
```

Arguments

`wd`	Working directory
`txid`	Taxonomic ID, numeric
`sqs`	Sequences

Details

Used within the `dwnld` function. Saves sequence data by txid in cache.

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`stages_run` *Sequentially run each stage*

Description

Runs stages from `frm` to `to`. Records stage progress in cache.

Usage

```
stages_run(wd, to, frm, stgs_msg, rstrt = FALSE)
```

Arguments

`wd`	Working directory
`to`	Total number of stages to run
`frm`	Starting stage to run from
`stgs_msg`	Printout stage message for log
`rstrt`	Restarting, T/F

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`stage_args_check` *Check stage arguments*

Description

Ensures stage arguments are valid, raises an error if not.

Usage

```
stage_args_check(to, frm)
```

Arguments

`to`	ending stage
`frm`	starting stage

Value

character, stage message

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`sturgeons` *sturgeons*

Description

sturgeons

Format

A TreeMan or Phylota object

Examples

```
data("sturgeons")
```

`summary_phylota` *Summarise clusters in Phylota Table*

Description

Generates a summary data.frame from all clusters in Phylota object.

Usage

```
summary_phylota(phylota)
```

Arguments

`phylota`	Phylota object

See Also

Other tools-private: `mk_txid_in_sq_mtrx()`, `update_phylota()`

`tardigrades` *tardigrades*

Description

tardigrades

Format

A TreeMan or Phylota object

Examples

```
data("tardigrades")
```

`taxaResolve` *Resolve taxonmic names online*

Description

Resolve taxonomic names via the Global Names Resolver.

Usage

```
taxaResolve(
  nms,
  batch = 100,
  datasource = 4,
  genus = TRUE,
  cache = FALSE,
  parent = NULL
)
```

Arguments

`nms`	vector of names
`batch`	size of the batches to be queried
`datasource`	ID number of the datasource
`genus`	boolean, if true will search against GNR with just the genus name for names that failed to resolve using the full species name
`cache`	T/F, create a local cache of downloaded names?
`parent`	specify parent of all names to prevent false names

Details

Returns dataframe containing GNR metadata for each name wames that cannot be resolved are returned as NA. Various datasources are available, see `http://resolver.globalnames.org/data_sources` for a list and IDs. Default is 4 for NCBI. Will raise a warning if connection fails and will return NULL.

See Also

`searchTxnyms`, `setTxnyms`, `getNdsFrmTxnyms`

Examples

```
my_lovely_names <- c(
  "Gallus gallus", "Pongo pingu", "Homo sapiens",
  "Arabidopsis thaliana", "Macaca thibetana", "Bacillus subtilis"
)
res <- taxaResolve(nms = my_lovely_names)
length(colnames(res)) # 10 different metadata for returned names including original search name
```

```
# let's look at the lineages
lineages <- strsplit(as.vector(res$lineage), "\\|")
print(lineages[[6]]) # the bacteria has far fewer taxonomic levels
```

`TaxDict-class` *Taxonomic record dictionary*

Description

Taxonomic dictionary contains a taxonomic tree and NCBI taxonomy data for all taxonomic IDs.

Usage

```
## S4 method for signature 'TaxDict'
as.character(x)

## S4 method for signature 'TaxDict'
show(object)

## S4 method for signature 'TaxDict'
print(x)

## S4 method for signature 'TaxDict'
str(object, max.level = 2L, ...)

## S4 method for signature 'TaxDict'
summary(object)
```

Arguments

`x`	`TaxDict` object
`object`	`TaxDict` object
`max.level`	Maximum level of nesting for str()
`...`	Further arguments for str()

Slots

`txids` Taxonomic IDs of taxon records

`recs` Environment of records

`prnt` Parent taxonomic ID

`txtr` Taxonomic tree

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
data('aotus')
txdct <- aotus@txdct
# this is a TaxDict object
# it contains taxonomic information, including records and tree
show(txdct)
# you can access its different data slots with @
txdct@txids  # taxonomic IDs
txdct@recs   # taxonomic records environment
txdct@txtr   # taxonomic tree
txdct@prnt   # MRCA
# access any record through the records environment
txdct@recs[[txdct@txids[[1]]]]
# for interacting with the taxonomic tree, see the treeman package
summary(txdct@txtr)
```

`taxdict_gen` *Generate taxonomic dictionary*

Description

Takes a vector of txids and a list of taxonomic records and returns a taxonomic dictionary.

Usage

```
taxdict_gen(txids, recs, ps)
```

Arguments

`txids`	Vector of taxonomic IDs
`recs`	List of taxonomic records
`ps`	Parameters list, generated with parameters()

Value

TaxDict

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`, `warn()`

`taxise_run` *Run taxise stage*

Description

Run the first stage of phylotaR, taxise. This looks up all descendant taxonomic nodes for a given taxonomic ID. It then looks up relevant taxonomic information and generates a taxonomic dictionary for user interaction after phylotaR has completed.

Usage

```
taxise_run(wd)
```

Arguments

`wd`	Working directory

Details

Objects will be cached.

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `TaxRec-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `setup()`

Examples

```
## Not run:

  # Note: this example requires BLAST and internet to run.

  # example with temp folder
  wd <- file.path(tempdir(), 'aotus')
  # setup for aotus, make sure aotus/ folder already exists
  if (!dir.exists(wd)) {
    dir.create(wd)
  }
  ncbi_dr <- '[SET BLAST+ BIN PATH HERE]'
  setup(wd = wd, txid = 9504, ncbi_dr = ncbi_dr)  # txid for Aotus primate genus
  # individually run stages
  taxise_run(wd = wd)

## End(Not run)
```

`TaxRec-class` *Taxonomic record*

Description

Taxonomic dictionary contains a taxonomic tree and NCBI taxonomy data for all taxonomic IDs.

Usage

```
## S4 method for signature 'TaxRec'
as.character(x)

## S4 method for signature 'TaxRec'
show(object)

## S4 method for signature 'TaxRec'
print(x)

## S4 method for signature 'TaxRec'
str(object, max.level = 2L, ...)

## S4 method for signature 'TaxRec'
summary(object)
```

Arguments

`x`	`TaxRec` object
`object`	`TaxRec` object
`max.level`	Maximum level of nesting for str()
`...`	Further arguments for str()

Slots

`id` Taxonomic ID

`scnm` Scientific name

`cmnm` Common name

`rnk` Rank

`lng` Lineage

`prnt` Parent

See Also

Other run-public: `ClstrArc-class`, `ClstrRec-class`, `Phylota-class`, `SeqArc-class`, `SeqRec-class`, `TaxDict-class`, `clusters2_run()`, `clusters_run()`, `parameters_reset()`, `reset()`, `restart()`, `run()`, `setup()`, `taxise_run()`

Examples

```
data('aotus')
taxrec <- aotus@txdct@recs[[aotus@txdct@txids[[1]]]]
# this is a TaxRec object
# it contains NCBI's taxonomic information for a single node
show(taxrec)
# you can access its different data slots with @
taxrec@id    # taxonomic ID
taxrec@scnm  # scientific name
taxrec@cmnm  # common name, '' if none
taxrec@rnk   # rank
taxrec@lng   # lineage information: list of IDs and ranks
taxrec@prnt  # parent ID
```

`taxtree_gen` *Generate taxonomic tree*

Description

Generate a taxonomic tree for easy look up of taxonomic parents and descendants.

Usage

```
taxtree_gen(prinds, ids, root, ps)
```

Arguments

`prinds`	Vector of integers indicating preceding node.
`ids`	Vector of taxonomic IDs
`root`	ID of root taxon
`ps`	Parameters list, generated with parameters()

Value

TreeMan

TreeMan class

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, rank_get(), rawseqrec_breakdown(), safely_connect(), search_and_cache(), searchterm_gen(), seeds_blast(), seq_download(), seqarc_gen(), seqrec_augment(), seqrec_convert(), seqrec_gen(), seqrec_get(), sids_check(), sids_get(), sids_load(), sids_save(), sqs_count(), sqs_save(), stage_args_check(), stages_run(), tax_download(), taxdict_gen(), txids_get(), txnds_count(), warn()

`tax_download` *Download taxonomic records*

Description

Downloads one batch of taxonomic records.

Usage

```
tax_download(ids, ps)
```

Arguments

`ids`	Vector of taxonomic IDs
`ps`	Parameters list, generated with parameters()

Value

list of list

See Also

Other run-private: batcher(), blast_clstr(), blast_filter(), blast_setup(), blast_sqs(), blastcache_load(), blastcache_save(), blastdb_gen(), blastn_run(), cache_rm(), cache_setup(), clade_select(), clstr2_calc(), clstr_all(), clstr_direct(), clstr_sqs(), clstr_subtree(), clstrarc_gen(), clstrarc_join(), clstrrec_gen(), clstrs_calc(), clstrs_join(), clstrs_merge(), clstrs_renumber(), clstrs_save(), cmdln(), descendants_get(), download_obj_check(), error(), gb_extract(), hierarchic_download(), info(), ncbicache_load(), ncbicache_save(), obj_check(), obj_load(), obj_save(), outfmt_get(), parameters_load(), parameters_setup(), parent_get(), progress_init(), progress_read(), progress_reset(), progress_save(), rank_get(), rawseqrec_breakdown(), safely_connect(), search_and_cache(), searchterm_gen(), seeds_blast(), seq_download(), seqarc_gen(), seqrec_augment(), seqrec_convert(), seqrec_gen(), seqrec_get(), sids_check(), sids_get(), sids_load(), sids_save(), sqs_count(), sqs_save(), stage_args_check(), stages_run(), taxdict_gen(), taxtree_gen(), txids_get(), txnds_count(), warn()

`tinamous` *tinamous*

Description

tinamous

Format

A TreeMan or Phylota object

Examples

```
data("tinamous")
```

`TreeMan-class` *TreeMan-class*

Description

S4 class for representing phylogenetic trees as a list of nodes.

Usage

```
## S4 method for signature 'TreeMan,character'
x[[i]]

## S4 method for signature 'TreeMan,character,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'TreeMan'
as.character(x)

## S4 method for signature 'TreeMan'
show(object)

## S4 method for signature 'TreeMan'
print(x)

## S4 method for signature 'TreeMan'
str(object, max.level = 2L, ...)

## S4 method for signature 'TreeMan'
summary(object)

## S4 method for signature 'TreeMan'
cTrees(x, ...)
```

Arguments

`x`	`TreeMan` object
`i`	node ID or slot name
`j`	missing
`...`	additional tree objects
`drop`	missing
`object`	`TreeMan` object
`max.level`	`str()` maximum number of levels to show

Details

A `TreeMan` object holds a list of nodes. The idea of the `TreeMan` class is to make adding and removing nodes as similar as possible to adding and removing elements in a list. Note that internal nodes and tips are both considered nodes. Trees can be polytomous but not unrooted.

Each node within the `TreeMan` `ndlst` contains the following data slots:

- `id`: character string for the node ID
- `txnym`: name of taxonomic clade (optional)
- `spn`: length of the preceding branch
- `prid`: ID of the immediately preceding node, NULL if root
- `ptid`: IDs of the immediately connecting nodes

See below in 'Examples' for these methods in use.

Slots

`ndlst` list of nodes

`nds` vector of node ids that are internal nodes

`nnds` numeric of number of internal nodes in tree

`tips` vector of node ids that are tips

`ntips` numeric of number of internal nodes in tree

`all` vector of all node ids

`nall` numeric of number of all nodes in tree

`pd` numeric of total branch length of tree

`tinds` indexes of all tip nodes in tree

`prinds` indexes of all pre-nodes in tree

`wspn` logical, do nodes have spans

`wtxnyms` logical, do nodes have txnyms

`ply` logical, is tree bifurcating

`root` character of node id of root, if no root then empty character

`updtd` logical, if tree slots have been updated since initiation or change

`othr_slt_nms` vector, character list of additional data slots added to nodes

`ndmtrx` matrix, T/Fs representing tree structure

See Also

`randTree`, `Node-class`, `phylo-to-TreeMan`, `TreeMan-to-phylo`

Examples

```
# Generate random tree
tree <- randTree(10)
# Print to get basic stats
summary(tree)
# Slots....
tree["tips"] # return all tips IDs
tree["nds"] # return all internal node IDs
tree["ntips"] # count all tips
tree["nnds"] # count all internal nodes
tree["root"] # identify root node
tree[["t1"]] # return t1 node object
tree["pd"] # return phylogenetic diversity
tree["ply"] # is polytomous?
# Additional special slots (calculated upon call)
tree["age"] # get tree's age
tree["ultr"] # determine if tree is ultrametric
tree["spns"] # get all the spans of the tree IDs
tree["prids"] # get all the IDs of preceding nodes
tree["ptids"] # get all the IDs of following nodes
tree["txnyms"] # get all the taxonyms of all nodes
# In addition [] can be used for any user-defined slot
# Because all nodes are lists with metadata we can readily
#  get specific information on nodes of interest
nd <- tree[["n2"]]
summary(nd)
# And then use the same syntax for the tree
nd["nkids"] # .... nkids, pd, etc.

# Convert to phylo and plot
library(ape)
tree <- as(tree, "phylo")
plot(tree)
```

`TreeMan-to-phylo` *Convert TreeMan to phylo*

Description

Return ape's `phylo` from a `TreeMan`. This will preserve node labels if they are different from the default labels (n#).

See Also

`phylo-to-TreeMan`, `TreeMen-to-multiPhylo` `multiPhylo-to-TreeMen` `TreeMan-class`

Examples

```
library(ape)
tree <- randTree(10)
tree <- as(tree, "phylo")
```

`TreeMen-class` *TreeMen-class*

Description

S4 class for multiple phylogenetic trees

Usage

```
## S4 method for signature 'TreeMen'
cTrees(x, ...)

## S4 method for signature 'TreeMen,ANY'
x[[i]]

## S4 method for signature 'TreeMen,character,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'TreeMen'
as.character(x)

## S4 method for signature 'TreeMen'
show(object)

## S4 method for signature 'TreeMen'
str(object, max.level = 2L, ...)

## S4 method for signature 'TreeMen'
print(x)

## S4 method for signature 'TreeMen'
summary(object)
```

Arguments

`x`	`TreeMen` object
`...`	additional tree objects
`i`	tree index (integer or character)
`j`	missing
`drop`	missing
`object`	`TreeMen` object
`max.level`	`str()` maximum level

Slots

`treelst`	list of `TreeMan` objects
`ntips`	sum of tips per tree
`ntrees`	total number of trees

See Also

`cTrees`

`TreeMen-to-multiPhylo` *Convert TreeMen to multiPhylo*

Description

Return ape's `multiPhylo` from a `TreeMen`

See Also

`TreeMan-to-phylo`, `phylo-to-TreeMan`, `multiPhylo-to-TreeMen` `TreeMan-class`

Examples

```
library(ape)
trees <- cTrees(randTree(10), randTree(10), randTree(10))
trees <- as(trees, "multiPhylo")
```

`twoer` *Generate a tree of two tips*

Description

Returns a `TreeMan` tree with two tips and a root.

Usage

```
twoer(tids = c("t1", "t2"), spns = c(1, 1), rid = "root", root_spn = 0)
```

Arguments

`tids`	tip IDs
`spns`	tip spans
`rid`	root ID
`root_spn`	root span

Details

Useful for building larger trees with `addClade()`. Note, a node matrix cannot be added to a tree of two tips.

See Also

`TreeMan-class`, `randTree`

Examples

```
tree <- twoer()
```

`txids_get` *Searches for descendant taxonomic IDs*

Description

Searches NCBI taxonomy for all descendant taxonomic nodes.

Usage

```
txids_get(ps, retmax = 10000)
```

Arguments

`ps`	Parameters list, generated with parameters()
`retmax`	integer, maximum number of IDs to return per query

Value

Vector of txids

vector of ids

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txnds_count()`, `warn()`

`txnds_count` *Count number of descending taxonomic nodes*

Description

Searches NCBI taxonomy and returns number of descendants taxonomic nodes (species, genera ...) of ID.

Usage

```
txnds_count(txid, ps)
```

Arguments

`txid`	Taxonomic ID
`ps`	Parameters list, generated with parameters()

Value

integer

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `warn()`

`ultrTree` *Make tree ultrametric*

Description

Returns a tree with all tips ending at time 0

Usage

```
ultrTree(tree)
```

Arguments

`tree`	`TreeMan` object

Details

Re-calculates the branch lengths in the tree so that all tips are brought to the same time point: all species are extant.

See Also

https://github.com/DomBennett/treeman/wiki/manip-methods

Examples

```
tree <- randTree(10)
(getDcsd(tree)) # list all extinct tips
tree <- ultrTree(tree)
(getDcsd(tree)) # list all extinct tips
```

`unblncdTree` *Generate an unbalanced tree*

Description

Returns an unbalanced `TreeMan` tree with `n` tips.

Usage

```
unblncdTree(n, wndmtrx = FALSE, parallel = FALSE)
```

Arguments

`n`	number of tips, integer, must be 3 or greater
`wndmtrx`	T/F add node matrix? Default FALSE.
`parallel`	T/F run in parallel? Default FALSE.

Details

Equivalent to ape's `stree(type='left')` but returns a `TreeMan` tree. Tree is always rooted and bifurcating.

See Also

`TreeMan-class`, `randTree`, `blncdTree`

Examples

```
tree <- unblncdTree(5)
```

`updateSlts` *Update tree slots after manipulation*

Description

Return tree with updated slots.

Usage

```
updateSlts(tree)
```

Arguments

`tree`	`TreeMan` object

Details

Tree slots in the `TreeMan` object are usually automatically updated. For certain single node manipulations they are not. Run this function to update the slots.

See Also

`addNdmtrx`, `getAge`

`update_phylota` *Update slots*

Description

After change, run to update slots.

Usage

```
update_phylota(phylota)
```

Arguments

`phylota`	Phylota

Value

Phylota

See Also

Other tools-private: `mk_txid_in_sq_mtrx()`, `summary_phylota()`

`warn` *Write warning message to log*

Description

Inform a user if a potential error has occurred in log.txt.

Usage

```
warn(ps, ...)
```

Arguments

`ps`	Parameters list, generated with parameters()
`...`	Message elements for concatenating

See Also

Other run-private: `batcher()`, `blast_clstr()`, `blast_filter()`, `blast_setup()`, `blast_sqs()`, `blastcache_load()`, `blastcache_save()`, `blastdb_gen()`, `blastn_run()`, `cache_rm()`, `cache_setup()`, `clade_select()`, `clstr2_calc()`, `clstr_all()`, `clstr_direct()`, `clstr_sqs()`, `clstr_subtree()`, `clstrarc_gen()`, `clstrarc_join()`, `clstrrec_gen()`, `clstrs_calc()`, `clstrs_join()`, `clstrs_merge()`, `clstrs_renumber()`, `clstrs_save()`, `cmdln()`, `descendants_get()`, `download_obj_check()`, `error()`, `gb_extract()`, `hierarchic_download()`, `info()`, `ncbicache_load()`, `ncbicache_save()`, `obj_check()`, `obj_load()`, `obj_save()`, `outfmt_get()`, `parameters_load()`, `parameters_setup()`, `parent_get()`, `progress_init()`, `progress_read()`, `progress_reset()`, `progress_save()`, `rank_get()`, `rawseqrec_breakdown()`, `safely_connect()`, `search_and_cache()`, `searchterm_gen()`, `seeds_blast()`, `seq_download()`, `seqarc_gen()`, `seqrec_augment()`, `seqrec_convert()`, `seqrec_gen()`, `seqrec_get()`, `sids_check()`, `sids_get()`, `sids_load()`, `sids_save()`, `sqs_count()`, `sqs_save()`, `stage_args_check()`, `stages_run()`, `tax_download()`, `taxdict_gen()`, `taxtree_gen()`, `txids_get()`, `txnds_count()`

`writeTree` *Write a Newick tree*

Description

Creates a Newick tree from a `TreeMan` object.

Usage

```
writeTree(
  tree,
  file,
  append = FALSE,
  ndLabels = function(nd) {
     return(NULL)
 },
  parallel = FALSE,
  progress = "none"
)
```

Arguments

`tree`	`TreeMan` object
`file`	file path
`append`	T/F append tree to already existing file
`ndLabels`	node label function
`parallel`	logical, make parallel?
`progress`	name of the progress bar to use, see `create_progress_bar`

Details

The `ndLabels` argument can be used to add a user defined node label in the Newick tree. It should take only 1 argument, `nd`, the node represented as a list. It should only return a single character value that can be added to a newick string.

See Also

`https://en.wikipedia.org/wiki/Newick_format`, `readTree`, `randTree`, `readTrmn`, `writeTrmn`, `saveTreeMan`, `loadTreeMan`

Examples

```
tree <- randTree(10)
# write out the tree with node labels as IDs
ndLabels <- function(n) {
  n[["id"]]
}
writeTree(tree, file = "example.tre", ndLabels = ndLabels)
file.remove("example.tre")
```

`writeTrmn` *Write a .trmn tree*

Description

Write to disk a `TreeMan` or `TreeMan` object using the .trmn treefile

Usage

```
writeTrmn(tree, file)
```

Arguments

`tree`	TreeMan object or TreeMen object
`file`	file path

Details

Write a tree(s) to file using the .trmn format. It is faster to read and write tree files using treeman with the .trmn file format. In addition it is possible to encode more information than possible with the Newick, e.g. any taxonomic information and additional slot names added to the tree are recorded in the file.

See Also

`readTrmn`, `readTree`,`writeTree`, `randTree`, `saveTreeMan`, `loadTreeMan`

Examples

```
tree <- randTree(10)
writeTrmn(tree, file = "test.trmn")
tree <- readTrmn("test.trmn")
file.remove("test.trmn")
```

`write_sqs` *Write out sequences*

Description

Write out sequences, as .fasta, for a given vector of IDs.

Usage

```
write_sqs(phylota, outfile, sid, sq_nm = sid, width = 80)
```

Arguments

`phylota`	Phylota
`outfile`	Output file
`sid`	Sequence ID(s)
`sq_nm`	Sequence name(s)
`width`	Maximum number of characters in a line, integer

Details

The user can control the output definition lines of the sequences using the sq_nm. By default sequences IDs are used. Note, ensure the sq_nm are in the same order as sid.

See Also

Other tools-public: `calc_mad()`, `calc_wrdfrq()`, `drop_by_rank()`, `drop_clstrs()`, `drop_sqs()`, `get_clstr_slot()`, `get_nsqs()`, `get_ntaxa()`, `get_sq_slot()`, `get_stage_times()`, `get_tx_slot()`, `get_txids()`, `is_txid_in_clstr()`, `is_txid_in_sq()`, `list_clstrrec_slots()`, `list_ncbi_ranks()`, `list_seqrec_slots()`, `list_taxrec_slots()`, `plot_phylota_pa()`, `plot_phylota_treemap()`, `read_phylota()`

Examples

```
data('aotus')
# get sequences for a cluster and write out
random_cid <- sample(aotus@cids, 1)
sids <- aotus[[random_cid]]@sids
write_sqs(phylota = aotus, outfile = file.path(tempdir(), 'test.fasta'),
          sq_nm = 'my_gene', sid = sids)
```

`yeasts` *yeasts*

Description

yeasts

Format

A TreeMan or Phylota object

Examples

```
data("yeasts")
```

Index