

# Package ‘TreeTools’

January 26, 2021

**Title** Create, Modify and Analyse Phylogenetic Trees

**Version** 1.4.2

**License** GPL (>= 3)

**Copyright** Incorporates C/C++ code from 'ape' by Emmanuel Paradis  
<doi:10.1093/bioinformatics/bty633>

**Description** Efficient implementations of functions for the creation, modification and analysis of phylogenetic trees.  
Applications include:  
generation of trees with specified shapes;  
analysis of tree shape;  
rooting of trees and extraction of subtrees;  
calculation and depiction of node support;  
calculation of ancestor-descendant relationships;  
import and export of trees from Newick, Nexus (Maddison et al. 1997)  
<doi:10.1093/sysbio/46.4.590>,  
and TNT <<http://www.lillo.org.ar/phylogeny/tnt/>> formats;  
and analysis of splits and cladistic information.

**URL** <https://ms609.github.io/TreeTools/>,  
<https://github.com/ms609/TreeTools/>

**BugReports** <https://github.com/ms609/TreeTools/issues/>

**SystemRequirements** C++14

**Depends** R (>= 3.4.0), ape (>= 5.0)

**Imports** bit64, colorspace, phangorn (>= 2.2.1), R.cache, Rdpack

**Suggests** knitr, Rcpp, rmarkdown, shiny, testthat, vdiff

**LinkingTo** Rcpp

**RdMacros** Rdpack

**LazyData** true

**ByteCompile** true

**Encoding** UTF-8

**Language** en-GB

**VignetteBuilder** knitr

**RoxygenNote** 7.1.1

**NeedsCompilation** yes

**Author** Martin R. Smith [aut, cre, cph]

(<<https://orcid.org/0000-0001-5660-1727>>),

Emmanuel Paradis [cph] (<<https://orcid.org/0000-0003-3092-2199>>)

**Maintainer** Martin R. Smith <[martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk)>

**Repository** CRAN

**Date/Publication** 2021-01-26 18:20:03 UTC

## R topics documented:

AddTip	3
ApeTime	5
ArtificialExtinction	6
as.Newick	8
as.Splits	9
brewer	10
CharacterInformation	11
CladeSizes	12
CladisticInfo	13
CollapseNode	14
ConsensusWithout	15
DescendantEdges	17
DoubleFactorial	18
doubleFactorials	19
DropTip	20
EdgeAncestry	21
EdgeDistances	22
EnforceOutgroup	23
GenerateTree	24
LabelSplits	25
LeafLabelInterchange	26
ListAncestors	27
Lobo.data	29
logDoubleFactorials	29
MakeTreeBinary	30
match	31
MRCA	32
MSTEdges	33
N1Spr	34
NDescendants	35
NewickTree	36
NJTree	37
NodeDepth	37
NodeOrder	38

NPartitionPairs . . . . .	39
NRooted . . . . .	40
nRootedShapes . . . . .	42
NSplits . . . . .	43
NTip . . . . .	44
PairwiseDistances . . . . .	45
print.TreeNumber . . . . .	46
ReadCharacters . . . . .	47
ReadTntTree . . . . .	48
ReNumber . . . . .	51
ReNumberTips . . . . .	52
RootNode . . . . .	53
RootTree . . . . .	54
sapply64 . . . . .	55
SingleTaxonTree . . . . .	56
SortTree . . . . .	57
SplitFrequency . . . . .	58
SplitInformation . . . . .	59
SplitMatchProbability . . . . .	61
SplitsInBinaryTree . . . . .	62
Stemwardness . . . . .	64
StringToPhyDat . . . . .	66
Subsplit . . . . .	68
Subtree . . . . .	69
SupportColour . . . . .	70
TipLabels . . . . .	71
TipsInSplits . . . . .	73
TotalCopheneticIndex . . . . .	74
TreeIsRooted . . . . .	76
TreeNumber . . . . .	77
TreesMatchingSplit . . . . .	79
TreesMatchingTree . . . . .	80
TrivialSplits . . . . .	81
UnrootedTreesMatchingSplit . . . . .	82
UnshiftTree . . . . .	83
WriteTntCharacters . . . . .	84
<b>Index</b>	<b>86</b>

---

AddTip	<i>Add a tip to a phylogenetic tree</i>
--------	---

---

### Description

AddTip() adds a tip to a phylogenetic tree at a specified location.

**Usage**

```
AddTip(
  tree,
  where = sample.int(tree$Nnode * 2 + 2L, size = 1) - 1L,
  label = "New tip",
  nTip = NTip(tree),
  nNode = tree$Nnode,
  rootNode = RootNode(tree)
)
```

```
AddTipEverywhere(tree, label = "New tip", includeRoot = FALSE)
```

**Arguments**

tree	A tree of class <a href="#">phylo</a> .
where	The node or tip that should form the sister taxon to the new node. To add a new tip at the root, use where = 0. By default, the new tip is added to a random edge.
label	Character string providing the label to apply to the new tip.
nTip, nNode, rootNode	Optional integer vectors specifying number of tips and nodes in tree, and index of root node. Not checked for correctness: specifying values here trades code safety for a nominal speed increase.
includeRoot	Logical; if TRUE, each position adjacent to the root edge is considered to represent distinct edges; if FALSE, they are treated as a single edge.

**Details**

AddTip() extends [bind.tree](#), which cannot handle single-taxon trees.

AddTipEverywhere() adds a tip to each edge in turn.

**Value**

AddTip() returns a tree of class [phylo](#) with an additional tip at the desired location.

AddTipEverywhere() returns a list of class [multiPhylo](#) containing the trees produced by adding label to each edge of tree in turn.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Add one tree to another: [bind.tree\(\)](#)

Other tree manipulation: [CollapseNode\(\)](#), [ConsensusWithout\(\)](#), [DropTip\(\)](#), [EnforceOutgroup\(\)](#), [LeafLabelInterchange\(\)](#), [MakeTreeBinary\(\)](#), [RenumberTips\(\)](#), [RenumberTree\(\)](#), [Renumber\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [SortTree\(\)](#), [Subtree\(\)](#)

**Examples**

```
plot(tree <- BalancedTree(10))
ape::nodeLabels()
ape::nodeLabels(15, 15, bg='green')

plot(AddTip(tree, 15, 'NEW_TIP'))

oldPar <- par(mfrow=c(2, 4), mar=rep(0.3, 4), cex=0.9)

backbone <- BalancedTree(4)
# Treating the position of the root as instructive:
additions <- AddTipEverywhere(backbone, includeRoot = TRUE)
xx <- lapply(additions, plot)

par(mfrow=c(2, 3))
# Don't treat root edges as distinct:
additions <- AddTipEverywhere(backbone, includeRoot = FALSE)
xx <- lapply(additions, plot)

par(oldPar)
```

---

ApeTime

*Read modification time from 'ape' Nexus file*

---

**Description**

ApeTime() reads the time that a tree written with 'ape' was modified, based on the comment in the Nexus file.

**Usage**

```
ApeTime(filename, format = "double")
```

**Arguments**

filename	Character string specifying path to the file.
format	Format in which to return the time: 'double' as a sortable numeric; any other value to return a string in the format YYYY-MM-DD hh:mm:ss.

**Value**

ApeTime() returns the time that the specified file was created by ape, in the format specified by format.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

---

ArtificialExtinction *Artificial Extinction*

---

### Description

Remove tokens that do not occur in a fossil 'template' taxon from a living taxon, to simulate the process of fossilization in removing data from a phylogenetic dataset.

### Usage

```
ArtificialExtinction(  
  dataset,  
  subject,  
  template,  
  replaceAmbiguous = "ambig",  
  replaceCoded = "original",  
  replaceAll = TRUE,  
  sampleFrom = NULL  
)  
  
## S3 method for class 'matrix'  
ArtificialExtinction(  
  dataset,  
  subject,  
  template,  
  replaceAmbiguous = "ambig",  
  replaceCoded = "original",  
  replaceAll = TRUE,  
  sampleFrom = NULL  
)  
  
## S3 method for class 'phyDat'  
ArtificialExtinction(  
  dataset,  
  subject,  
  template,  
  replaceAmbiguous = "ambig",  
  replaceCoded = "original",  
  replaceAll = TRUE,  
  sampleFrom = NULL  
)  
  
ArtEx(  
  dataset,  
  subject,  
  template,  
  replaceAmbiguous = "ambig",
```

```

    replaceCoded = "original",
    replaceAll = TRUE,
    sampleFrom = NULL
  )

```

### Arguments

dataset	Phylogenetic dataset of class <code>phyDat</code> or <code>matrix</code> .
subject	Vector identifying subject taxa, by name or index.
template	Character or integer identifying taxon to use as a template.
replaceAmbiguous, replaceCoded	Character specifying whether tokens that are ambiguous (?) or coded (not ?) in the fossil template should be replaced with: <ul style="list-style-type: none"> <li>• original: Their original value; i.e. no change;</li> <li>• ambiguous: The ambiguous token, ?;</li> <li>• binary: The tokens 0 or 1, with equal probability;</li> <li>• uniform: One of the tokens present in <code>sampleFrom</code>, with equal probability;</li> <li>• sample: One of the tokens present in <code>sampleFrom</code>, sampled according to their frequency.</li> </ul>
replaceAll	Logical: if TRUE, replace all tokens in a subject; if FALSE, leave any ambiguous tokens (?) ambiguous.
sampleFrom	Vector identifying a subset of characters from which to sample replacement tokens. If NULL, replacement tokens will be sampled from the initial states of all taxa not used as a template (including the subjects).

### Details

Note: this simple implementation does not account for character contingency, e.g. characters whose absence imposes inapplicable or absent tokens on dependent characters.

### Value

A dataset with the same class as `dataset` in which entries that are ambiguous in `template` are made ambiguous in `subject`.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### Examples

```

set.seed(1)
dataset <- matrix(c(sample(0:2, 4 * 8, TRUE),
                    '0', '0', rep('?', 6)), nrow = 5,
                 dimnames = list(c(LETTERS[1:4], 'FOSSIL'),
                                 paste('char', 1:8)), byrow = TRUE)
artex <- ArtificialExtinction(dataset, c('A', 'C'), 'FOSSIL')

```

---

`as.Newick`*Write a phylogenetic tree in Newick format*

---

### Description

`as.Newick()` creates a character string representation of a phylogenetic tree, in the Newick format, using R's internal tip numbering. Use [RenumberTips\(\)](#) to ensure that the internal numbering follows the order you expect.

### Usage

```
as.Newick(x)

## S3 method for class 'phylo'
as.Newick(x)

## S3 method for class 'list'
as.Newick(x)

## S3 method for class 'multiPhylo'
as.Newick(x)
```

### Arguments

`x` Object to convert to Newick format. See Usage section for supported classes.

### Value

`as.Newick()` returns a character string representing tree in Newick format.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

- Retain leaf labels: [NewickTree\(\)](#)
- Change R's internal numbering of leaves: [RenumberTips\(\)](#)
- Write tree to text or file: [ape::write.tree\(\)](#)

### Examples

```
trees <- list(BalancedTree(1:8), PectinateTree(8:1))
trees <- lapply(trees, RenumberTips, 1:8)
as.Newick(trees)
```



---

as.Splits                      *Convert object to Splits*

---

### Description

as.Splits() converts a phylogenetic tree to a Splits object representing its constituent bipartition splits.

### Usage

```
as.Splits(x, tipLabels = NULL, ...)

## S3 method for class 'phylo'
as.Splits(x, tipLabels = NULL, asSplits = TRUE, ...)

## S3 method for class 'multiPhylo'
as.Splits(x, tipLabels = x[[1]]$tip.label, asSplits = TRUE, ...)

## S3 method for class 'Splits'
as.Splits(x, tipLabels = NULL, ...)

## S3 method for class 'list'
as.Splits(x, tipLabels = NULL, asSplits = TRUE, ...)

## S3 method for class 'matrix'
as.Splits(x, tipLabels = NULL, ...)

## S3 method for class 'logical'
as.Splits(x, tipLabels = NULL, ...)

## S3 method for class 'Splits'
as.logical(x, tipLabels = NULL, ...)
```

### Arguments

x	Object to convert into splits: perhaps a tree of class <a href="#">phylo</a> . If a logical matrix is provided, each row will be considered as a separate split.
tipLabels	Character vector specifying sequence in which to order tip labels. Label order must (currently) match to combine or compare separate Splits objects.
...	Presently unused.
asSplits	Logical specifying whether to return a Splits object, or an unannotated two-dimensional array (useful where performance is paramount).

**Value**

`as.Splits()` returns an object of class `Splits`, or (if `asSplits = FALSE`) a two-dimensional array of raw objects, with each bit specifying whether or not the leaf corresponding to the respective bit position is a member of the split. Splits are named according to the node at the non-root end of the edge that defines them. In rooted trees, the child of the rightmost root edge names the split.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other Splits operations: [LabelSplits\(\)](#), [NSplits\(\)](#), [NTip\(\)](#), [SplitFrequency\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#), [TipsInSplits\(\)](#), [match\(\)](#)

**Examples**

```
splits <- as.Splits(BalancedTree(letters[1:6]))
summary(splits)
TipsInSplits(splits)
summary(!splits)
TipsInSplits(!splits)

length(splits + !splits)
length(unique(splits + !splits))

summary(c(splits[[2:3]], !splits[[1:2]]))

moreSplits <- as.Splits(PectinateTree(letters[6:1]), tipLabel = splits)
print(moreSplits, details = TRUE)
match(splits, moreSplits)
moreSplits %in% splits
```

---

brewer

*Brewer palettes*


---

**Description**

A list of eleven Brewer palettes containing one to eleven colours that are readily distinguished by colourblind viewers, followed by a twelfth 12-colour palette adapted for colour blindness.

**Usage**

```
brewer
```

**Format**

An object of class `list` of length 12.

**Source**

- [ColourBrewer2.org](http://ColourBrewer2.org)
- [Martin Krzywinski](#)

**Examples**

```
data("brewer", package="TreeTools")
plot(0, type='n', xlim=c(1, 12), ylim=c(12, 1),
     xlab = 'Colour', ylab='Palette')
for (i in seq_along(brewer)) text(seq_len(i), i, col=brewer[[i]])
```

---

CharacterInformation *Character information content*

---

**Description**

CharacterInformation() calculates the phylogenetic information content of a given character.

**Usage**

```
CharacterInformation(tokens)
```

**Arguments**

tokens            Character vector specifying the tokens assigned to each taxon for a character. Example: `c(0,0,0,1,1,1,'?','-')`. Note that ambiguous tokens such as `(01)` are not supported, and should be replaced with `?`.

**Value**

CharacterInformation() returns a numeric specifying the phylogenetic information content of the character (*sensu* Steel & Penny 2006), in bits.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**References**

- Steel MA, Penny D (2006). "Maximum parsimony and the phylogenetic information in multi-state characters." In Albert VA (ed.), *Parsimony, Phylogeny, and Genomics*, 163–178. Oxford University Press, Oxford.

**See Also**

Other split information functions: [SplitInformation\(\)](#), [SplitMatchProbability\(\)](#), [TreesMatchingSplit\(\)](#), [UnrootedTreesMatchingSplit\(\)](#)

---

CladeSizes

*Clade sizes*

---

### Description

CladeSizes() reports the number of nodes in each clade in a tree.

### Usage

```
CladeSizes(tree, internal = FALSE, nodes = NULL)
```

### Arguments

tree	A tree of class <a href="#">phylo</a> .
internal	Logical specifying whether internal nodes should be counted towards the size of each clade.
nodes	Integer specifying indices of nodes at the base of clades whose sizes should be returned. If unspecified, counts will be provided for all nodes (including leaves).

### Value

CladeSizes() returns the number of nodes (including leaves) that are descended from each node, not including the node itself.

### See Also

Other tree navigation: [AncestorEdge\(\)](#), [DescendantEdges\(\)](#), [EdgeAncestry\(\)](#), [EdgeDistances\(\)](#), [ListAncestors\(\)](#), [MRCA\(\)](#), [NDescendants\(\)](#), [NodeDepth\(\)](#), [NodeOrder\(\)](#), [NonDuplicateRoot\(\)](#), [RootNode\(\)](#)

### Examples

```
tree <- BalancedTree(6)
plot(tree)
ape::nodeLabels()
CladeSizes(tree, nodes = c(1, 8, 9))
```

---

CladisticInfo	<i>Cladistic information content of a tree</i>
---------------	--

---

**Description**

CladisticInfo() calculates the cladistic (phylogenetic) information content of a phylogenetic object, *sensu* Thorley *et al.* (1998).

**Usage**

```
CladisticInfo(x)

PhylogeneticInfo(x)

## S3 method for class 'phylo'
CladisticInfo(x)

## S3 method for class 'list'
CladisticInfo(x)

## S3 method for class 'multiPhylo'
CladisticInfo(x)

PhylogeneticInformation(x)

CladisticInformation(x)
```

**Arguments**

x                      Tree of class phylo, or a list thereof.

**Details**

The CIC is the logarithm of the number of binary trees that include the specified topology. A base two logarithm gives an information content in bits.

The CIC was originally proposed by Rohlf (1982), and formalised, with an information-theoretic justification, by Thorley *et al.* (1998). Steel and Penny (2006) term the equivalent quantity 'phylogenetic information content' in the context of individual characters.

The number of binary trees consistent with a cladogram provides a more satisfactory measure of the resolution of a tree than simply counting the number of edges resolved (Page, 1992).

**Value**

CladisticInfo() returns a numeric giving the cladistic information content of the input tree(s), in bits.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**References**

- Page RD (1992). “Comments on the information content of classifications.” *Cladistics*, **8**(1), 87–95. doi: [10.1111/j.10960031.1992.tb00054.x](https://doi.org/10.1111/j.10960031.1992.tb00054.x), <https://doi.org/10.1111/j.1096-0031.1992.tb00054.x>.
- Rohlf FJ (1982). “Consensus indices for comparing classifications.” *Mathematical Biosciences*, **59**(1), 131–144. doi: [10.1016/00255564\(82\)901122](https://doi.org/10.1016/00255564(82)901122), [https://doi.org/10.1016/0025-5564\(82\)90112-2](https://doi.org/10.1016/0025-5564(82)90112-2).
- Steel MA, Penny D (2006). “Maximum parsimony and the phylogenetic information in multistate characters.” In Albert VA (ed.), *Parsimony, Phylogeny, and Genomics*, 163–178. Oxford University Press, Oxford.
- Thorley JL, Wilkinson M, Charleston M (1998). “The information content of consensus trees.” In Rizzi A, Vichi M, Bock H (eds.), *Advances in Data Science and Classification*, 91–98. Springer, Berlin. ISBN 978-3-540-64641-9, doi: [10.1007/9783642722530](https://dx.doi.org/10.1007/978-3-642-72253-0), <https://dx.doi.org/10.1007/978-3-642-72253-0>.

**See Also**

Other tree information functions: [NRooted\(\)](#), [TreesMatchingTree\(\)](#)

Other tree characterization functions: [Stemwardness](#), [TotalCopheneticIndex\(\)](#)

---

CollapseNode

*Collapse nodes on a phylogenetic tree*

---

**Description**

Collapses specified nodes or edges on a phylogenetic tree, resulting in polytomies.

**Usage**

```
CollapseNode(tree, nodes)
```

```
## S3 method for class 'phylo'
CollapseNode(tree, nodes)
```

```
CollapseEdge(tree, edges)
```

**Arguments**

tree            A tree of class [phylo](#).

nodes, edges   Integer vector specifying the nodes or edges in the tree to be dropped. (Use [nodelabels\(\)](#) or [edgelabels\(\)](#) to view numbers on a plotted tree.)

**Value**

`CollapseNode()` and `CollapseEdge()` return a tree of class `phylo`, corresponding to tree with the specified nodes or edges collapsed. The length of each dropped edge will (naively) be added to each descendant edge.

**Author(s)**

Martin R. Smith

**See Also**

Other tree manipulation: [AddTip\(\)](#), [ConsensusWithout\(\)](#), [DropTip\(\)](#), [EnforceOutgroup\(\)](#), [LeafLabelInterchange\(\)](#), [MakeTreeBinary\(\)](#), [Renumertips\(\)](#), [Renumertree\(\)](#), [Renumber\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [SortTree\(\)](#), [Subtree\(\)](#)

**Examples**

```
oldPar <- par(mfrow=c(3, 1), mar=rep(0.5, 4))

tree <- as.phylo(898, 7)
tree$edge.length <- 11:22
plot(tree)
nodelabels()
edgelabels()
edgelabels(round(tree$edge.length, 2),
            cex = 0.6, frame = 'n', adj = c(1, -1))

# Collapse by node number
newTree <- CollapseNode(tree, c(12, 13))
plot(newTree)
nodelabels()
edgelabels(round(newTree$edge.length, 2),
            cex = 0.6, frame = 'n', adj = c(1, -1))

# Collapse by edge number
newTree <- CollapseEdge(tree, c(2, 4))
plot(newTree)

par(oldPar)
```

---

ConsensusWithout

*Consensus without taxa*

---

**Description**

`ConsensusWithout()` displays a consensus plot with specified taxa excluded, which can be a useful way to increase the resolution of a consensus tree when a few wildcard taxa obscure a consistent set of relationships. `MarkMissing()` adds missing taxa as loose leaves on the plot.

**Usage**

```

ConsensusWithout(trees, tip = character(0), ...)

## S3 method for class 'phylo'
ConsensusWithout(trees, tip = character(0), ...)

## S3 method for class 'multiPhylo'
ConsensusWithout(trees, tip = character(0), ...)

## S3 method for class 'list'
ConsensusWithout(trees, tip = character(0), ...)

MarkMissing(tip, position = "bottomleft", ...)

```

**Arguments**

trees	A list of phylogenetic trees, of class <code>multiPhylo</code> or <code>list</code> .
tip	A character vector specifying the names (or numbers) of tips to drop (using <code>ape::drop.tip()</code> ).
...	Additional parameters to pass on to <code>ape::consensus()</code> or <code>legend()</code> .
position	Where to plot the missing taxa. See <code>legend()</code> for options.

**Value**

`ConsensusWithout()` returns a consensus tree (of class `phylo`) without the excluded taxa.  
`MarkMissing()` provides a null return, after plotting the specified tips as a legend.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other tree manipulation: [AddTip\(\)](#), [CollapseNode\(\)](#), [DropTip\(\)](#), [EnforceOutgroup\(\)](#), [LeafLabelInterchange\(\)](#), [MakeTreeBinary\(\)](#), [RenumberTips\(\)](#), [RenumberTree\(\)](#), [Renumber\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [SortTree\(\)](#), [Subtree\(\)](#)

Other tree properties: [NSplits\(\)](#), [NTip\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#), [TreeIsRooted\(\)](#)

**Examples**

```

oldPar <- par(mfrow=c(1, 2), mar=rep(0.5, 4))

# Two trees differing only in placement of tip 2:
trees <- as.phylo(c(0, 53), 6)
plot(trees[[1]])
plot(trees[[2]])

# Strict consensus (left panel) lacks resolution:

```



```

plot(ape::consensus(trees))

# But omitting tip two (right panel) reveals shared structure in common:
plot(ConsensusWithout(trees, 't2'))
MarkMissing('t2')

par(oldPar)

```

---

DescendantEdges      *Identify descendant edges*

---

### Description

Quickly identify edges that are 'descended' from edges in a tree.

### Usage

```
DescendantEdges(edge = NULL, parent, child, nEdge = length(parent))
```

```
AllDescendantEdges(parent, child, nEdge = length(parent))
```

### Arguments

edge	Integer specifying the number of the edge whose child edges are required (see <a href="#">edgelabels()</a> ).
parent	Integer vector corresponding to the first column of the edge matrix of a tree of class <a href="#">phylo</a> , i.e. <code>tree\$edge[,1]</code>
child	Integer vector corresponding to the second column of the edge matrix of a tree of class <a href="#">phylo</a> , i.e. <code>tree\$edge[,2]</code> .
nEdge	number of edges (calculated from <code>length(parent)</code> if not supplied).

### Details

The order of parameters in `DescendantEdges()` will change in the future, to allow `AllDescendantEdges()` to be merged into this function ([#31](#)). Please explicitly name the edge parameter in `DescendantEdges()`, and replace `AllDescendantEdges()` with `DescendantEdges(edge = NULL)`, to future-proof your code.

### Value

`DescendantEdges()` returns a logical vector stating whether each edge in turn is a descendant of the specified edge (or the edge itself).

`AllDescendantEdges()` returns a matrix of class logical, with row *N* specifying whether each edge is a descendant of edge *N* (or the edge itself).

**See Also**

Other tree navigation: [AncestorEdge\(\)](#), [CladeSizes\(\)](#), [EdgeAncestry\(\)](#), [EdgeDistances\(\)](#), [ListAncestors\(\)](#), [MRCA\(\)](#), [NDescendants\(\)](#), [NodeDepth\(\)](#), [NodeOrder\(\)](#), [NonDuplicateRoot\(\)](#), [RootNode\(\)](#)

---

DoubleFactorial	<i>Double factorial</i>
-----------------	-------------------------

---

**Description**

Calculate the double factorial of a number, or its logarithm.

**Usage**

```
DoubleFactorial(n)
DoubleFactorial64(n)
LnDoubleFactorial(n)
Log2DoubleFactorial(n)
LogDoubleFactorial(n)
LnDoubleFactorial.int(n)
LogDoubleFactorial.int(n)
```

**Arguments**

n                      Vector of integers.

**Value**

Returns the double factorial,  $n * (n - 2) * (n - 4) * (n - 6) * \dots$

**Functions**

- `DoubleFactorial64`: Returns the exact double factorial as a 64-bit integer64, for  $n < 34$ .
- `LnDoubleFactorial`: Returns the logarithm of the double factorial.
- `Log2DoubleFactorial`: Returns the logarithm of the double factorial.
- `LnDoubleFactorial.int`: Slightly faster, when  $x$  is known to be length one and below 50001

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other double factorials: [doubleFactorials](#), [logDoubleFactorials](#)

**Examples**

```
DoubleFactorial (-4:0) # Return 1 if n < 2
DoubleFactorial (2) # 2
DoubleFactorial (5) # 1 * 3 * 5
exp(LnDoubleFactorial.int (8)) # log(2 * 4 * 6 * 8)
DoubleFactorial64(31)
```

---

doubleFactorials	<i>Double factorials</i>
------------------	--------------------------

---

**Description**

A vector with pre-calculated values of double factorials up to 300!!, and the logarithms of double factorials up to 50 000!!.

**Usage**

```
doubleFactorials
```

**Format**

An object of class numeric of length 300.

**Details**

301!! is too large to store as an integer; use [logDoubleFactorials](#) instead.

**See Also**

Other double factorials: [DoubleFactorial\(\)](#), [logDoubleFactorials](#)

---

DropTip

*Drop tips from tree*

---

## Description

DropTip() removes specified tips from a phylogenetic tree, collapsing incident branches.

## Usage

```
DropTip(tree, tip)
```

```
## S3 method for class 'phylo'
```

```
DropTip(tree, tip)
```

```
## S3 method for class 'multiPhylo'
```

```
DropTip(tree, tip)
```

## Arguments

tree	A tree of class <a href="#">phylo</a> .
tip	Character vector specifying labels of leaves in tree to be dropped, or integer vector specifying the indices of leaves to be dropped. Specifying the index of an internal node will drop all descendants of that node.

## Details

This function is more robust than [ape::drop.tip\(\)](#) as it does not require any particular internal node numbering schema. It is not presently as fast, though it is ripe for optimization; if you are finding this function is a rate-limiting step, please get in touch and I'll prioritise writing a faster implementation.

## Value

DropTip() returns a tree of class `phylo`, in [Preorder](#), with the requested leaves removed.

## Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

## See Also

Other tree manipulation: [AddTip\(\)](#), [CollapseNode\(\)](#), [ConsensusWithout\(\)](#), [EnforceOutgroup\(\)](#), [LeafLabelInterchange\(\)](#), [MakeTreeBinary\(\)](#), [RenumTips\(\)](#), [RenumTree\(\)](#), [Renumber\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [SortTree\(\)](#), [Subtree\(\)](#)

**Examples**

```
tree <- BalancedTree(8)
plot(tree)
plot(DropTip(tree, c('t4', 't5')))
```

---

EdgeAncestry

*Ancestors of an edge*

---

**Description**

Quickly identify edges that are 'ancestral' to a particular edge in a tree.

**Usage**

```
EdgeAncestry(edge, parent, child, stopAt = (parent == min(parent)))
```

**Arguments**

edge	Integer specifying the number of the edge whose child edges should be returned.
parent	Integer vector corresponding to the first column of the edge matrix of a tree of class <a href="#">phylo</a> , i.e. <code>tree\$edge[, 1]</code>
child	Integer vector corresponding to the second column of the edge matrix of a tree of class <a href="#">phylo</a> , i.e. <code>tree\$edge[, 2]</code> .
stopAt	Integer or logical vector specifying the edge(s) at which to terminate the search; defaults to the edges with the smallest parent, which will be the root edges if nodes are numbered <a href="#">Cladewise</a> or in <a href="#">Preorder</a> .

**Value**

`EdgeAncestry()` returns a logical vector stating whether each edge in turn is a descendant of the specified edge.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other tree navigation: [AncestorEdge\(\)](#), [CladeSizes\(\)](#), [DescendantEdges\(\)](#), [EdgeDistances\(\)](#), [ListAncestors\(\)](#), [MRCA\(\)](#), [NDescendants\(\)](#), [NodeDepth\(\)](#), [NodeOrder\(\)](#), [NonDuplicateRoot\(\)](#), [RootNode\(\)](#)

### Examples

```
tree <- PectinateTree(6)
plot(tree)
ape::edgelabels()
parent <- tree$edge[, 1]
child <- tree$edge[, 2]
EdgeAncestry(7, parent, child)
which(EdgeAncestry(7, parent, child, stopAt = 4))
```

---

EdgeDistances

*Distance between edges*

---

### Description

Number of nodes that must be traversed to navigate from each edge to each other edge within a tree

### Usage

```
EdgeDistances(tree)
```

### Arguments

tree            A tree of class [phylo](#).

### Value

EdgeDistances() returns a symmetrical matrix listing the number of edges that must be traversed to travel from each numbered edge to each other. The two edges straddling the root of a rooted tree are treated as a single edge. Add a 'root' tip using [AddTip\(\)](#) if the position of the root is significant.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

Other tree navigation: [AncestorEdge\(\)](#), [CladeSizes\(\)](#), [DescendantEdges\(\)](#), [EdgeAncestry\(\)](#), [ListAncestors\(\)](#), [MRCA\(\)](#), [NDescendants\(\)](#), [NodeDepth\(\)](#), [NodeOrder\(\)](#), [NonDuplicateRoot\(\)](#), [RootNode\(\)](#)

### Examples

```
tree <- BalancedTree(5)
plot(tree)
ape::edgelabels()

EdgeDistances(tree)
```

---

EnforceOutgroup	<i>Generate a tree with a specific outgroup</i>
-----------------	---

---

### Description

Given a tree or a list of taxa, `EnforceOutgroup()` rearranges the ingroup and outgroup taxa such that the two are sister taxa across the root, without changing the relationships within the ingroup or within the outgroup.

### Usage

```
EnforceOutgroup(tree, outgroup)

## S3 method for class 'phylo'
EnforceOutgroup(tree, outgroup)

## S3 method for class 'character'
EnforceOutgroup(tree, outgroup)
```

### Arguments

tree	Either a tree of class <code>phylo</code> ; or (for <code>EnforceOutgroup()</code> ) a character vector listing the names of all the taxa in the tree, from which a random tree will be generated.
outgroup	Character vector containing the names of taxa to include in the outgroup.

### Value

`EnforceOutgroup()` returns a tree of class `phylo` where all outgroup taxa are sister to all remaining taxa, without modifying the ingroup topology.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

For a more robust implementation, see [RootTree\(\)](#), which will eventually replace this function ([#30](#)).

Other tree manipulation: [AddTip\(\)](#), [CollapseNode\(\)](#), [ConsensusWithout\(\)](#), [DropTip\(\)](#), [LeafLabelInterchange\(\)](#), [MakeTreeBinary\(\)](#), [ReorderTips\(\)](#), [ReorderTree\(\)](#), [Reorder\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [SortTree\(\)](#), [Subtree\(\)](#)

### Examples

```
tree <- EnforceOutgroup(letters[1:9], letters[1:3])
plot(tree)
```

---

`GenerateTree`*Generate pectinate, balanced or random trees*

---

**Description**

`RandomTree()`, `PectinateTree()`, `BalancedTree()` and `StarTree()` generate trees with the specified shapes and leaf labels.

**Usage**

```
RandomTree(tips, root = FALSE)
```

```
PectinateTree(tips)
```

```
BalancedTree(tips)
```

```
StarTree(tips)
```

**Arguments**

<code>tips</code>	An integer specifying the number of tips, or a character vector naming the tips, or any other object from which <code>TipLabels()</code> can extract leaf labels.
<code>root</code>	Character or integer specifying tip to use as root, if desired; or FALSE for an unrooted tree.

**Value**

Each function returns an unweighted binary tree of class `phylo` with the specified leaf labels. Trees are rooted unless `root = FALSE`.

`RandomTree()` returns a topology drawn at random from the uniform distribution (i.e. each binary tree is drawn with equal probability). Trees are generated by inserting each tip in term at a randomly selected edge in the tree. Random numbers are generated using a Mersenne Twister. If `root = FALSE`, the tree will be unrooted, with the first tip in a basal position. Otherwise, the tree will be rooted on `root`.

`PectinateTree()` returns a pectinate (caterpillar) tree.

`BalancedTree()` returns a balanced (symmetrical) tree.

`StarTree()` returns a completely unresolved (star) tree.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other tree generation functions: [NJTree\(\)](#), [SingleTaxonTree\(\)](#), [TreeNumber](#)



## Examples

```
RandomTree(LETTERS[1:10])

data('Lobo')
RandomTree(Lobo.phy)

plot(PectinateTree(LETTERS[1:10]))

plot(BalancedTree(LETTERS[1:10]))
plot(StarTree(LETTERS[1:10]))
```

---

LabelSplits

*Label splits*

---

## Description

Labels the edges associated with each split on a plotted tree.

## Usage

```
LabelSplits(tree, labels, unit = "", ...)
```

## Arguments

tree	A tree of class <a href="#">phylo</a> .
labels	Named vector listing annotations for each split. Names should correspond to the node associated with each split; see <a href="#">as.Splits()</a> for details.
unit	Character specifying units of labels, if desired. Include a leading space if necessary.
...	Additional parameters to <a href="#">ape::edgelabels()</a> .

## Details

As the two root edges of a rooted tree denote the same split, only the rightmost (plotted at the bottom, by default) edge will be labelled. If the position of the root is significant, add a tip at the root using [AddTip\(\)](#).

## Value

`LabelSplits()` returns `invisible()`, after plotting labels on each relevant edge of a plot (which should already have been produced using `plot(tree)`).

**See Also**

Calculate split support: [SplitFrequency\(\)](#)

Colour labels according to value: [SupportColour\(\)](#)

Other Splits operations: [NSplits\(\)](#), [NTip\(\)](#), [SplitFrequency\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#), [TipsInSplits\(\)](#), [as.Splits\(\)](#), [match\(\)](#)

**Examples**

```
tree <- BalancedTree(LETTERS[1:5])
splits <- as.Splits(tree)
plot(tree)
LabelSplits(tree, as.character(splits), frame = 'none', pos = 3L)
LabelSplits(tree, TipsInSplits(splits), unit = ' tips', frame = 'none',
            pos = 1L)

# An example forest of 100 trees, some identical
forest <- as.phylo(c(1, rep(10, 79), rep(100, 15), rep(1000, 5)), nTip = 9)

# Generate an 80% consensus tree
cons <- ape::consensus(forest, p = 0.8)
plot(cons)

splitFreqs <- SplitFrequency(cons, forest)
LabelSplits(cons, splitFreqs, unit = '%',
            col = SupportColor(splitFreqs / 100),
            frame = 'none', pos = 3L)
```

---

LeafLabelInterchange    *Leaf label interchange*

---

**Description**

LeafLabelInterchange() exchanges the position of leaves within a tree.

**Usage**

```
LeafLabelInterchange(tree, n = 2L)
```

**Arguments**

**tree**                    A tree of class [phylo](#).

**n**                        Integer specifying number of leaves whose positions should be exchanged.

**Details**

Modifies a tree by switching the positions of  $n$  leaves. To avoid later swaps undoing earlier exchanges, all  $n$  leaves are guaranteed to change position. Note, however, that no attempt is made to avoid swapping equivalent leaves, for example, a pair that are each others' closest relatives. As such, the relationships within a tree are not guaranteed to be changed.

**Value**

LeafLabelInterchange() returns a tree of class `phylo` on which the position of  $n$  leaves have been exchanged. The tree's internal topology will not change.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other tree manipulation: [AddTip\(\)](#), [CollapseNode\(\)](#), [ConsensusWithout\(\)](#), [DropTip\(\)](#), [EnforceOutgroup\(\)](#), [MakeTreeBinary\(\)](#), [Renumertips\(\)](#), [Renumertree\(\)](#), [Renumber\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [SortTree\(\)](#), [Subtree\(\)](#)

**Examples**

```
tree <- PectinateTree(8)
plot(LeafLabelInterchange(tree, 3L))
```

---

ListAncestors

*List ancestors*


---

**Description**

ListAncestors() reports all ancestors of a given node.

**Usage**

```
ListAncestors(parent, child, node = NULL)
```

```
AllAncestors(parent, child)
```

**Arguments**

parent	Integer vector corresponding to the first column of the edge matrix of a tree of class <code>phylo</code> , i.e. <code>tree\$edge[, 1]</code>
child	Integer vector corresponding to the second column of the edge matrix of a tree of class <code>phylo</code> , i.e. <code>tree\$edge[, 2]</code> .
node	Integer giving the index of the node or tip whose ancestors are required, or NULL to return ancestors of all nodes.

**Details**

Note that if `node = NULL`, the tree's edges must be listed such that each internal node (except the root) is listed as a child before it is listed as a parent, i.e. its index in `child` is less than its index in `parent`. This will be true of trees listed in [Preorder](#).

**Value**

If `node = NULL`, `ListAncestors()` returns a list. Each entry  $i$  contains a vector containing, in order, the nodes encountered when traversing the tree from node  $i$  to the root node. The last entry of each member of the list is therefore the root node, with the exception of the entry for the root node itself, which is a zero-length integer.

If `node` is an integer, `ListAncestors()` returns a vector of the numbers of the nodes ancestral to the given node, including the root node.

**Functions**

- `AllAncestors`: Alias for `ListAncestors(node = NULL)`.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Implemented less efficiently in `phangorn:::Ancestors`, on which this code is based.

Other tree navigation: [AncestorEdge\(\)](#), [CladeSizes\(\)](#), [DescendantEdges\(\)](#), [EdgeAncestry\(\)](#), [EdgeDistances\(\)](#), [MRCA\(\)](#), [NDescendants\(\)](#), [NodeDepth\(\)](#), [NodeOrder\(\)](#), [NonDuplicateRoot\(\)](#), [RootNode\(\)](#)

Other tree navigation: [AncestorEdge\(\)](#), [CladeSizes\(\)](#), [DescendantEdges\(\)](#), [EdgeAncestry\(\)](#), [EdgeDistances\(\)](#), [MRCA\(\)](#), [NDescendants\(\)](#), [NodeDepth\(\)](#), [NodeOrder\(\)](#), [NonDuplicateRoot\(\)](#), [RootNode\(\)](#)

**Examples**

```
tree <- PectinateTree(5)
edge <- tree$edge

# Identify desired node with:
plot(tree)
nodelabels()
tiplabels()

# Ancestors of specific nodes:
ListAncestors(edge[, 1], edge[, 2], 4L)
ListAncestors(edge[, 1], edge[, 2], 8L)

# Ancestors of each node, if tree numbering system is uncertain:
lapply(seq_len(max(edge)), ListAncestors,
```

```
parent = edge[, 1], child = edge[, 2])

# Ancestors of each node, if tree is in preorder:
ListAncestors(edge[, 1], edge[, 2])

# Alias:
AllAncestors(edge[, 1], edge[, 2])
```

---

Lobo.data

*Data from Zhang et al. 2016*

---

### Description

Phylogenetic data from Zhang *et al.* (2016) in raw (Lobo.data) and phyDat (Lobo.phy) formats.

### Usage

Lobo.data

Lobo.phy

### Format

An object of class `list` of length 48.

An object of class `phyDat` of length 48.

### Source

Zhang X, Smith MR, Yang J, Hou J (2016). “Onychophoran-like musculature in a phosphatized Cambrian lobopodian.” *Biology Letters*, **12**(9), 20160492. doi: [10.1098/rsbl.2016.0492](https://doi.org/10.1098/rsbl.2016.0492), <https://dx.doi.org/10.1098/rsbl.2016.0492>.

---

logDoubleFactorials

*Natural logarithms of double factorials*

---

### Description

logDoubleFactorials is a numeric vector with pre-calculated values of double factorials up to 50 000!!.

### Usage

logDoubleFactorials

**Format**

An object of class `numeric` of length 50000.

**See Also**

Other double factorials: [DoubleFactorial\(\)](#), [doubleFactorials](#)

---

MakeTreeBinary

*Generate binary tree by collapsing polytomies*

---

**Description**

`MakeTreeBinary()` resolves, at random, all polytomies in a tree or set of trees, such that all trees compatible with the input topology are drawn with equal probability.

**Usage**

```
MakeTreeBinary(tree)
```

**Arguments**

`tree`            A tree of class [phylo](#).

**Value**

`MakeTreeBinary()` returns a rooted binary tree of class `phylo`, corresponding to tree uniformly selected from all those compatible with the input tree topologies.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Resolve polytomies such that some resolutions are more probable than others using [ape::multi2di\(\)](#).

Other tree manipulation: [AddTip\(\)](#), [CollapseNode\(\)](#), [ConsensusWithout\(\)](#), [DropTip\(\)](#), [EnforceOutgroup\(\)](#), [LeafLabelInterchange\(\)](#), [RenumberTips\(\)](#), [RenumberTree\(\)](#), [Renumber\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [SortTree\(\)](#), [Subtree\(\)](#)

**Examples**

```
MakeTreeBinary(CollapseNode(PectinateTree(7), c(9, 11, 13)))
UnrootTree(MakeTreeBinary(StarTree(5)))
```

---

match	<i>Split matching</i>
-------	-----------------------

---

### Description

`match()` returns a vector of the positions of (first) matches of splits in its first argument in its second. `%in%` is a more intuitive interface as a binary operator, which returns a logical vector indicating whether there is a match or not for each split in its left operand.

### Usage

```
match(x, table, ...)  
  
## S3 method for class 'Splits'  
match(x, table, ...)  
  
## S3 method for class 'list'  
match(x, table, ...)  
  
x %in% table  
  
## S3 method for class 'Splits'  
x %in% table  
  
in.Splits(x, table)
```

### Arguments

<code>x, table</code>	Object of class <code>Splits</code> .
<code>...</code>	Specify <code>nomatch =</code> to provide an integer value that will be used in place of NA in the case where no match is found.

### Details

`in.Splits()` is an alias for `%in%`, included for backwards compatibility. It will be deprecated in a future release.

### Value

`match()` returns an integer vector specifying the position in `table` that matches each element in `x`, or `nomatch` if no match is found.

`%in%` returns a logical vector specifying which of the splits in `x` are present in `table`.

### See Also

Corresponding base functions are documented in [match\(\)](#).

Other `Splits` operations: [LabelSplits\(\)](#), [NSplits\(\)](#), [NTip\(\)](#), [SplitFrequency\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#), [TipsInSplits\(\)](#), [as.Splits\(\)](#)

### Examples

```
splits1 <- as.Splits(BalancedTree(7))
splits2 <- as.Splits(PectinateTree(7))

match(splits1, splits2)
splits1 %in% splits2
```

---

MRCA

*Most recent common ancestor*

---

### Description

MRCA() calculates the last common ancestor of specified nodes.

### Usage

```
MRCA(x1, x2, ancestors)
```

### Arguments

x1, x2	Integer specifying index of leaves or nodes whose most recent common ancestor should be found.
ancestors	List of ancestors for each node in a tree. Perhaps produced by <a href="#">ListAncestors()</a> .

### Details

MRCA() requires that node values within a tree increase away from the root, which will be true of trees listed in Preorder. No warnings will be given if trees do not fulfil this requirement.

### Value

MRCA() returns an integer specifying the node number of the last common ancestor of x1 and x2.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

Other tree navigation: [AncestorEdge\(\)](#), [CladeSizes\(\)](#), [DescendantEdges\(\)](#), [EdgeAncestry\(\)](#), [EdgeDistances\(\)](#), [ListAncestors\(\)](#), [NDescendants\(\)](#), [NodeDepth\(\)](#), [NodeOrder\(\)](#), [NonDuplicateRoot\(\)](#), [RootNode\(\)](#)



**Examples**

```

tree <- BalancedTree(7)

# Verify that node numbering increases away from root
plot(tree)
nodelabels()

# ListAncestors expects a tree in Preorder
tree <- Preorder(tree)
edge <- tree$edge
ancestors <- ListAncestors(edge[, 1], edge[, 2])
MRCA(1, 4, ancestors)

# If a tree must be in postorder, use:
tree <- Postorder(tree)
edge <- tree$edge
ancestors <- lapply(seq_len(max(edge)), ListAncestors,
                    parent = edge[, 1], child = edge[, 2])

```

---

MSTEdges

*Minimum spanning tree*


---

**Description**

Calculate or plot the minimum spanning tree of a distance matrix.

**Usage**

```
MSTEdges(distances, plot = FALSE, x = NULL, y = NULL, ...)
```

```
MSTLength(distances, mst = NULL)
```

**Arguments**

distances	Either a matrix that can be interpreted as a distance matrix, or an object of class <code>dist</code> .
plot	Logical specifying whether to add the minimum spanning tree to an existing plot.
x, y	Numeric vectors specifying the X and Y coordinates of each element in distances. Necessary only if <code>plot = TRUE</code> .
...	Additional parameters to send to <code>[lines()]</code> .
mst	Optional parameter specifying the minimum spanning tree in the format returned by <code>MSTEdges()</code> ; if <code>NULL</code> , calculated from distances.

**Value**

MSTEdges() returns a matrix in which each row corresponds to an edge of the minimum spanning tree, listed in non-decreasing order of length. The two columns contain the indices of the entries in distances that each edge connects, with the lower value listed first.

MSTLength() returns the length of the minimum spanning tree.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**References**

Gower JC, Ross GJS (1969). “Minimum spanning trees and single linkage cluster analysis.” *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **18**(1), 54–64. doi: [10.2307/2346439](https://doi.org/10.2307/2346439), <https://doi.org/10.2307/2346439>.

**See Also**

Slow implementation returning the association matrix of the minimum spanning tree: `ape::mst()`.

**Examples**

```
# Corners of an almost-regular octahedron
points <- matrix(c(0, 0, 2, 2, 1.1, 1,
                 0, 2, 0, 2, 1, 1.1,
                 0, 0, 0, 0, 1, -1), 6)
distances <- dist(points)
mst <- MSTEdges(distances)
MSTLength(distances, mst)
plot(points[, 1:2], ann = FALSE, asp = 1)
MSTEdges(distances, TRUE, x = points[, 1], y = points[, 2], lwd = 2)
```

---

N1Spr

*Number of trees one SPR step away*

---

**Description**

N1Spr() calculates the number of trees one subtree prune-and-regraft operation away from a binary input tree using the formula given by Allen and Steel (2001); IC1Spr() calculates the information content of trees at this distance: i.e. the entropy corresponding to the proportion of all possible  $n$ -tip trees whose SPR distance is at most one from a specified tree.

**Usage**

N1Spr(n)

IC1Spr(n)

**Arguments**

n                    Integer vector specifying the number of tips in a tree.

**Value**

N1Spr() returns an integer vector denoting the number of trees one SPR rearrangement away from the input tree..

IC1Spr() returns an numeric vector giving the phylogenetic information content of trees 0 or 1 SPR rearrangement from an  $n$ -leaf tree, in bits.

**References**

Allen BL, Steel MA (2001). "Subtree transfer operations and their induced metrics on evolutionary trees." *Annals of Combinatorics*, 5(1), 1–15. doi: [10.1007/s0002600180068](https://doi.org/10.1007/s0002600180068), <https://doi.org/10.1007/s00026-001-8006-8>.

**Examples**

```
N1Spr(4:6)
IC1Spr(5)
```

---

NDescendants	<i>Count descendants for each node in a tree</i>
--------------	--

---

**Description**

NDescendants() counts the number of nodes (including leaves) directly descended from each node in a tree.

**Usage**

```
NDescendants(tree)
```

**Arguments**

tree                A tree of class `phylo`.

**Value**

NDescendants() returns an integer listing the number of direct descendants (leaves or internal nodes) for each node in a tree.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other tree navigation: [AncestorEdge\(\)](#), [CladeSizes\(\)](#), [DescendantEdges\(\)](#), [EdgeAncestry\(\)](#), [EdgeDistances\(\)](#), [ListAncestors\(\)](#), [MRCA\(\)](#), [NodeDepth\(\)](#), [NodeOrder\(\)](#), [NonDuplicateRoot\(\)](#), [RootNode\(\)](#)

**Examples**

```
tree <- CollapseNode(BalancedTree(8), 12:15)
NDescendants(tree)
plot(tree)
nodeLabels(NDescendants(tree))
```

---

NewickTree

*Write Newick Tree*

---

**Description**

`NewickTree()` encodes a tree as a Newick-format string. This differs from `write.tree()` in the encoding of spaces as spaces, rather than underscores.

**Usage**

```
NewickTree(tree)
```

**Arguments**

tree            A tree of class `phylo`.

**Value**

`NewickTree()` returns a character string denoting tree in Newick format.

**See Also**

Use tip numbers, rather than leaf labels: [as.Newick](#)

**Examples**

```
NewickTree(BalancedTree(LETTERS[4:9]))
```

---

NJTree	<i>Generate a neighbour joining tree</i>
--------	--

---

**Description**

NJTree() generates a rooted neighbour joining tree from a phylogenetic dataset.

**Usage**

```
NJTree(dataset, edgeLengths = FALSE)
```

**Arguments**

dataset	A phylogenetic data matrix of class <a href="#">phyDat</a> , whose names correspond to the labels of any accompanying tree.
edgeLengths	Logical specifying whether to include edge lengths.

**Value**

NJTree returns an object of class `phylo`.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other tree generation functions: [GenerateTree](#), [SingleTaxonTree\(\)](#), [TreeNumber](#)

**Examples**

```
data('Lobo')
NJTree(Lobo.phy)
```

---

NodeDepth	<i>Distance of each node from tree exterior</i>
-----------	---

---

**Description**

NodeDepth() evaluates how 'deep' each node is within a tree.

**Usage**

```
NodeDepth(x, shortest = FALSE, includeTips = TRUE)
```

**Arguments**

<code>x</code>	A tree of class <code>phylo</code> , its <code>\$edge</code> property, or a list thereof.
<code>shortest</code>	Logical specifying whether to calculate the length of the shortest away-from-root path to a leaf. If <code>FALSE</code> , the length of the longest such route will be returned.
<code>includeTips</code>	Logical specifying whether to include leaves (each of depth zero) in return value.

**Details**

For a rooted tree, the depth of a node is the minimum (if `shortest = TRUE`) or maximum (`shortest = FALSE`) number of edges that must be traversed, moving away from the root, to reach a leaf.

Unrooted trees are treated as if a root node occurs in the 'middle' of the tree, meaning the position that will minimise the maximum node depth.

**Value**

`NodeDepth()` returns an integer vector specifying the depth of each external and internal node in `x`.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

`ape::node.depth` returns the number of tips descended from a node.

Other tree navigation: [AncestorEdge\(\)](#), [CladeSizes\(\)](#), [DescendantEdges\(\)](#), [EdgeAncestry\(\)](#), [EdgeDistances\(\)](#), [ListAncestors\(\)](#), [MRCA\(\)](#), [NDescendants\(\)](#), [NodeOrder\(\)](#), [NonDuplicateRoot\(\)](#), [RootNode\(\)](#)

**Examples**

```
tree <- CollapseNode(BalancedTree(10), c(12:13, 19))
plot(tree)
nodelabels(NodeDepth(tree, includeTips = FALSE))
```

---

NodeOrder

*Order of each node in a tree*

---

**Description**

`NodeOrder()` calculates the number of edges incident to each node in a tree. Includes the root edge in rooted trees.

**Usage**

```
NodeOrder(x, includeAncestor = TRUE, internalOnly = FALSE)
```

**Arguments**

x	A tree of class phylo, its \$edge property, or a list thereof.
includeAncestor	Logical specifying whether to count edge leading to ancestral node in calculation of order.
internalOnly	Logical specifying whether to restrict to results to internal nodes, i.e. to omit leaves. Irrelevant if includeAncestor = FALSE.

**Value**

NodeOrder() returns an integer listing the order of each node; entries are named with the number of each node.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other tree navigation: [AncestorEdge\(\)](#), [CladeSizes\(\)](#), [DescendantEdges\(\)](#), [EdgeAncestry\(\)](#), [EdgeDistances\(\)](#), [ListAncestors\(\)](#), [MRCA\(\)](#), [NDescendants\(\)](#), [NodeDepth\(\)](#), [NonDuplicateRoot\(\)](#), [RootNode\(\)](#)

**Examples**

```
tree <- CollapseNode(BalancedTree(8), 12:15)
NodeOrder(tree)
plot(tree)
nodelabels(NodeOrder(tree, internalOnly = TRUE))
```

---

NPartitionPairs

*Distributions of tips consistent with a partition pair*

---

**Description**

NPartitionPairs() calculates the number of terminal arrangements matching a specified configuration of two splits.

**Usage**

```
NPartitionPairs(configuration)
```

**Arguments**

configuration	Integer vector of length four specifying the number of terminals that occur in both (1) splits A1 and A2; (2) splits A1 and B2; (3) splits B1 and A2; (4) splits B1 and B2.
---------------	---

**Details**

Consider splits that divide eight terminals, labelled A to H.

Bipartition 1: ABCD:EFGH    A1 = ABCD    B1 = EFGH  
 Bipartition 2: ABE:CDFGH    A2 = ABE    B2 = CDFGH

This can be represented by an association matrix:

	<i>A2</i>	<i>B2</i>
<i>A1</i>	AB	C
<i>B1</i>	E	FGH

The cells in this matrix contain 2, 1, 1 and 3 terminals respectively; this four-element vector (c(2, 1, 1, 3)) is the configuration implied by this pair of bipartition splits.

**Value**

The number of ways to distribute sum(configuration) taxa according to the specified pattern.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**Examples**

```
NPartitionPairs(c(2, 1, 1, 3))
```

---

NRooted

*Number of trees*

---

**Description**

These functions return the number of rooted or unrooted binary trees consistent with a given pattern of splits.

**Usage**

```
NRooted(tips)
```

```
NUnrooted(tips)
```

```
NRooted64(tips)
```

```
NUnrooted64(tips)
```

```
LnUnrooted(tips)
```



```

LnUnrooted.int(tips)
Log2Unrooted(tips)
Log2Unrooted.int(tips)
LnRooted(tips)
LnRooted.int(tips)
Log2Rooted(tips)
Log2Rooted.int(tips)
LnUnrootedSplits(...)
Log2UnrootedSplits(...)
NUnrootedSplits(...)
LnUnrootedMult(...)
Log2UnrootedMult(...)
NUnrootedMult(...)

```

### Arguments

tips	Integer specifying the number of leaves.
...	Integer vector, or series of integers, listing the number of leaves in each split.

### Details

Functions starting N return the number of rooted or unrooted trees. Replace this initial N with Ln for the natural logarithm of this number; or Log2 for its base 2 logarithm.

Calculations follow Cavalli-Sforza & Edwards (1967) and Carter *et al.* 1990, Theorem 2.

### Functions

- NUnrooted: Number of unrooted trees
- NRooted64: Exact number of rooted trees as 64-bit integer ( $13 < nTip < 19$ )
- NUnrooted64: Exact number of unrooted trees as 64-bit integer ( $14 < nTip < 20$ )
- LnUnrooted: Log Number of unrooted trees
- LnUnrooted.int: Log Number of unrooted trees (as integer)
- LnRooted: Log Number of rooted trees
- LnRooted.int: Log Number of rooted trees (as integer)

- NUnrootedSplits: Number of unrooted trees consistent with a bipartition split.
- NUnrootedMult: Number of unrooted trees consistent with a multi-partition split.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### References

Carter M, Hendy M, Penny D, Székely LA, Wormald NC (1990). “On the distribution of lengths of evolutionary trees.” *SIAM Journal on Discrete Mathematics*, **3**(1), 38–47. doi: [10.1137/0403005](https://doi.org/10.1137/0403005), <https://doi.org/10.1137/0403005>.

Cavalli-Sforza LL, Edwards AWF (1967). “Phylogenetic analysis: models and estimation procedures.” *Evolution*, **21**(3), 550–570. ISSN 00143820, doi: [10.1111/j.15585646.1967.tb03411.x](https://doi.org/10.1111/j.15585646.1967.tb03411.x), <https://doi.org/10.1111/j.1558-5646.1967.tb03411.x>.

### See Also

Other tree information functions: [CladisticInfo\(\)](#), [TreesMatchingTree\(\)](#)

### Examples

```

NRooted(10)
NUnrooted(10)
LnRooted(10)
LnUnrooted(10)
Log2Unrooted(10)
# Number of trees consistent with a character whose states are
# 00000 11111 222
NUnrootedMult(c(5,5,3))

NUnrooted64(18)
LnUnrootedSplits(c(2,4))
LnUnrootedSplits(3, 3)
Log2UnrootedSplits(c(2,4))
Log2UnrootedSplits(3, 3)
NUnrootedSplits(c(2,4))
NUnrootedSplits(3, 3)

```

---

nRootedShapes

*Number of rooted / unrooted tree shapes*

---

### Description

nRootedShapes and nUnrootedShapes give the number of (un)rooted binary trees on  $n$  unlabelled leaves.

**Usage**

nRootedShapes

nUnrootedShapes

**Format**

An object of class integer64 of length 55.

An object of class integer64 of length 60.

**Source**

nRootedShapes corresponds to the Wedderburn-Etherington numbers, [OEIS A001190](#)

nUnrootedShapes is [OEIS A000672](#)

---

NSplits	<i>Number of distinct splits</i>
---------	----------------------------------

---

**Description**

NSplits() counts the unique bipartition splits in a tree or object.

**Usage**

NSplits(x)

NPartitions(x)

## S3 method for class 'phylo'

NSplits(x)

## S3 method for class 'list'

NSplits(x)

## S3 method for class 'multiPhylo'

NSplits(x)

## S3 method for class 'Splits'

NSplits(x)

## S3 method for class 'numeric'

NSplits(x)

## S3 method for class ``NULL``

NSplits(x)

```
## S3 method for class 'ClusterTable'
NSplits(x)
```

```
## S3 method for class 'character'
NSplits(x)
```

### Arguments

x                    A phylogenetic tree of class `phylo`; a list of such trees (of class `list` or `multiPhylo`); a `Splits` object; a vector of integers; or a character vector listing tips of a tree, or a character of length one specifying a tree in Newick format.

### Value

`NSplits()` returns an integer specifying the number of bipartitions in the specified objects, or in a binary tree with `x` tips.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

Other tree properties: [ConsensusWithout\(\)](#), [NTip\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#), [TreeIsRooted\(\)](#)

Other Splits operations: [LabelSplits\(\)](#), [NTip\(\)](#), [SplitFrequency\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#), [TipsInSplits\(\)](#), [as.Splits\(\)](#), [match\(\)](#)

### Examples

```
NSplits(8L)
NSplits(PectinateTree(8))
NSplits(as.Splits(BalancedTree(8)))
```

---

NTip

*Number of leaves in a phylogenetic tree*

---

### Description

`NTip()` extends [ape::Ntip\(\)](#) to handle objects of class `Splits` and `list`, and edge matrices (equivalent to `tree$edge`).

**Usage**

```
NTip(phy)

## Default S3 method:
NTip(phy)

## S3 method for class 'Splits'
NTip(phy)

## S3 method for class 'list'
NTip(phy)

## S3 method for class 'phylo'
NTip(phy)

## S3 method for class 'multiPhylo'
NTip(phy)

## S3 method for class 'matrix'
NTip(phy)
```

**Arguments**

phy                    Object representing one or more phylogenetic trees.

**Value**

NTip() returns an integer specifying the number of tips in each object in phy.

**See Also**

Other tree properties: [ConsensusWithout\(\)](#), [NSplits\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#), [TreeIsRooted\(\)](#)

Other Splits operations: [LabelSplits\(\)](#), [NSplits\(\)](#), [SplitFrequency\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#), [TipsInSplits\(\)](#), [as.Splits\(\)](#), [match\(\)](#)

---

PairwiseDistances            *Distances between each pair of trees*

---

**Description**

Distances between each pair of trees

**Usage**

```
PairwiseDistances(trees, Func, valueLength = 1L, ...)
```

**Arguments**

trees	List of trees of class phylo.
Func	Function returning a distance between two trees.
valueLength	Integer specifying expected length of the value returned by Func.
...	Additional arguments to Func.

**Value**

Matrix detailing distance between each pair of trees. Identical trees are assumed to have zero distance.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**Examples**

```
trees <- list(BalancedTree(8), PectinateTree(8), StarTree(8))
TCIDiff <- function (tree1, tree2) {
  TotalCopheneticIndex(tree1) - TotalCopheneticIndex(tree2)
}
PairwiseDistances(trees, TCIDiff, 1)
TCIRange <- function (tree1, tree2) {
  range(TotalCopheneticIndex(tree1), TotalCopheneticIndex(tree2))
}
PairwiseDistances(trees, TCIRange, 2)
```

---

<code>print.TreeNumber</code>	<i>Print TreeNumber object</i>
-------------------------------	--------------------------------

---

**Description**

S3 method for objects of class `TreeNumber`.

**Usage**

```
## S3 method for class 'TreeNumber'
print(x, ...)
```

**Arguments**

x	Object of class <code>TreeNumber</code> .
...	Additional arguments for consistency with S3 method (unused).

---

ReadCharacters	<i>Read phylogenetic characters from file</i>
----------------	---

---

**Description**

Parse a Nexus or TNT file, reading character states and names.

**Usage**

```
ReadCharacters(filepath, character_num = NULL, session = NULL)
```

```
ReadTntCharacters(filepath, character_num = NULL, type = NULL, session = NULL)
```

```
ReadAsPhyDat(filepath)
```

```
ReadTntAsPhyDat(filepath)
```

```
PhyDat(dataset)
```

**Arguments**

filepath	character string specifying location of file
character_num	Index of character(s) to return. NULL, the default, returns all characters.
session	(Optional) A Shiny session with a numericInput named character_num whose maximum should be updated.
type	Character vector specifying categories of data to extract from file. Setting type = c('num', 'dna') will return only characters following a &[num] or &[dna] tag in a TNT input file. Leave as NULL (the default) to return all characters.
dataset	list of taxa and characters, in the format produced by <a href="#">read.nexus.data</a> : a list of sequences each made of a single character vector, and named with the taxon name.

**Details**

Tested with matrices downloaded from [MorphoBank](#), but should also work more widely; please [report](#) incorrectly parsed files.

Matrices must contain only continuous or only discrete characters; maximum one matrix per file. Continuous characters will be read as strings (i.e. base type 'character').

**Value**

ReadCharacters() and ReadTNTCharacters() return a matrix whose row names correspond to tip labels, and column names correspond to character labels, with the attribute state.labels listing the state labels for each character; or a list of length one containing a character string explaining why the function call was unsuccessful.

ReadAsPhyDat() and ReadTntAsPhyDat() return a [phyDat](#) object.

## Functions

- **PhyDat**: A convenient wrapper for **phangorn**'s `phyDat()`, which converts a **list** of morphological characters into a `phyDat` object. If your morphological characters are in the form of a **matrix**, perhaps because they have been read using `read.table()`, try `MatrixToPhyDat()` instead.

## Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

## References

Maddison DR, Swofford DL, Maddison WP (1997). "Nexus: an extensible file format for systematic information." *Systematic Biology*, **46**, 590–621. doi: [10.1093/sysbio/46.4.590](https://doi.org/10.1093/sysbio/46.4.590), <https://doi.org/10.1093/sysbio/46.4.590>.

## See Also

- Convert between matrices and `phyDat` objects: `MatrixToPhyDat()`
- Write characters to TNT-format file: `WriteTntCharacters()`

## Examples

```
fileName <- paste0(system.file(package='TreeTools'),
                  '/extdata/input/dataset.nex')
ReadCharacters(fileName)

fileName <- paste0(system.file(package='TreeTools'),
                  '/extdata/tests/continuous.nex')
continuous <- ReadCharacters(fileName)

# To convert from strings to numbers:
at <- attributes(continuous)
continuous <- suppressWarnings(as.numeric(continuous))
attributes(continuous) <- at
continuous
```

---

ReadTntTree

*Parse TNT Tree*

---

## Description

Read a tree from TNT's parenthetical output.



**Usage**

```
ReadTntTree(filename, relativePath = NULL, keepEnd = 1L, tipLabels = NULL)
```

```
TntText2Tree(treeText)
```

```
TNTText2Tree(treeText)
```

**Arguments**

<code>filename</code>	character string specifying path to TNT <code>.tre</code> file, relative to the R working directory (visible with <code>getwd()</code> ).
<code>relativePath</code>	(discouraged) character string specifying location of the matrix file used to generate the TNT results, relative to the current working directory. Taxon names will be read from this file if they are not specified by <code>tipLabels</code> .
<code>keepEnd</code>	(optional, default 1) integer specifying how many elements of the file path to conserve when creating relative path (see examples).
<code>tipLabels</code>	(optional) character vector specifying the names of the taxa, in the sequence that they appear in the TNT file. If not specified, taxon names will be loaded from the data file linked in the first line of the <code>.tre</code> file specified in <code>filename</code> .
<code>treeText</code>	Character string describing a tree, in the parenthetical format output by TNT.

**Details**

**TNT** is software for parsimony analysis. Whilst its implementation of tree search is extremely rapid, analysis of results in TNT is made difficult by its esoteric and scantily documented scripting language.

`ReadTntTree()` aims to aid the user by facilitating the import of trees generated in TNT into R for further analysis.

The function depends on tree files being saved by TNT in parenthetical notation, using the TNT command `tsav*`. Trees are easiest to load into R if taxa have been saved using their names (TNT command `taxname=`). In this case, the TNT `.tre` file contains tip labels and can be parsed directly. The downside is that the uncompressed `.tre` files will have a larger file size.

`ReadTntTree()` can also read `.tre` files in which taxa have been saved using their numbers (`tax-name-`). Such files contain a hard-coded link to the matrix file that was used to generate the trees, in the first line of the `.tre` file. This poses problems for portability: if the matrix file is moved, or the `.tre` file is accessed on another computer, the taxon names may be lost. As such, it is important to check that the matrix file exists in the expected location – if it does not, either use the `relativePath` argument to point to its new location, or specify `tipLabels` to manually specify the tip labels.

`TntText2Tree()` converts text representation of a tree in TNT to an object of class `phylo`.

**Value**

`ReadTntTree()` returns a tree of class `phylo`, corresponding to the tree in `filename`, or `NULL` if no trees are found.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**Examples**

```
# In the examples below, TNT has read a matrix from
# "c:/TreeTools/input/dataset.nex"
# The results of an analysis were written to
# "c:/TreeTools/output/results1.tnt"
#
# results1.tnt will contain a hard-coded reference to
# "c:/TreeTools/input/dataset.nex".

# On the original machine (but not elsewhere), it would be possible to read
# this hard-coded reference from results.tnt:
# ReadTntTree('output/results1.tnt')

# These datasets are provided with the 'TreeTools' package, which will
# probably not be located at c:/TreeTools on your machine:

oldWD <- getwd() # Remember the current working directory
setwd(system.file(package = 'TreeTools'))

# If taxon names were saved within the file (using `taxname=` in TNT),
# then our job is easy:
ReadTntTree('extdata/output/named.tre')

# But if taxa were compressed to numbers (using `taxname-`), we need to
# look up the original matrix in order to dereference the tip names.
#
# We need to extract the relevant file path from the end of the
# hard-coded path in the original file.
#
# We are interested in the last two elements of
# c:/TreeTools/input/dataset.nex
#           2      1
#
# '.' means "relative to the current directory"
ReadTntTree('extdata/output/numbered.tre', './extdata', 2)

# If working in a lower subdirectory
setwd('./extdata/otherfolder')

# then it will be necessary to navigate up the directory path with '..':
ReadTntTree('../output/numbered.tre', '..', 2)

setwd(oldWD) # Restore original working directory

TNTText2Tree("(A (B (C (D E ))));")
```

---

Renumber	<i>Renumber a tree's nodes and tips</i>
----------	---

---

**Description**

Renumber() numbers the nodes and tips in a tree to conform with the phylo standards.

**Usage**

```
Renumber(tree)
```

**Arguments**

tree            A tree of class [phylo](#).

**Details**

The 'ape' class [phylo](#) is not formally defined, but expects trees' internal representation to conform to certain principles: for example, nodes should be numbered sequentially, with values increasing away from the root.

Renumber() attempts to reformat any tree into a representation that will not cause 'ape' functions to produce unwanted results or to crash R.

**Value**

Renumber() returns a tree of class [phylo](#), numbered in a [Cladewise](#) fashion consistent with the expectations of 'ape' functions.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

[Preorder\(\)](#) provides a faster and simpler alternative, but also rotates nodes.

Other tree manipulation: [AddTip\(\)](#), [CollapseNode\(\)](#), [ConsensusWithout\(\)](#), [DropTip\(\)](#), [EnforceOutgroup\(\)](#), [LeafLabelInterchange\(\)](#), [MakeTreeBinary\(\)](#), [RenumberTips\(\)](#), [RenumberTree\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [SortTree\(\)](#), [Subtree\(\)](#)

**Examples**

```
tree <- RandomTree(letters[1:10])
Renumber(tree)
```

---

RenumberTips	<i>Renumber a tree's tips</i>
--------------	-------------------------------

---

**Description**

RenumberTips(tree, tipOrder) sorts the tips of a phylogenetic tree tree such that the indices in tree\$edge[, 2] correspond to the order of leaves given in tipOrder.

**Usage**

```
RenumberTips(tree, tipOrder)

## S3 method for class 'phylo'
RenumberTips(tree, tipOrder)

## S3 method for class 'multiPhylo'
RenumberTips(tree, tipOrder)

## S3 method for class 'list'
RenumberTips(tree, tipOrder)
```

**Arguments**

tree	A tree of class <a href="#">phylo</a> .
tipOrder	A character vector containing the values of tree\$tip.label in the desired sort order, or an object (perhaps of class phylo or Splits) with tip labels.

**Value**

RenumberTips() returns tree, with the tips' internal representation numbered to match tipOrder.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other tree manipulation: [AddTip\(\)](#), [CollapseNode\(\)](#), [ConsensusWithout\(\)](#), [DropTip\(\)](#), [EnforceOutgroup\(\)](#), [LeafLabelInterchange\(\)](#), [MakeTreeBinary\(\)](#), [RenumberTree\(\)](#), [Renumber\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [SortTree\(\)](#), [Subtree\(\)](#)

**Examples**

```
data('Lobo') # Loads the phyDat object Lobo.phy
tree <- RandomTree(Lobo.phy)
tree <- RenumberTips(tree, names(Lobo.phy))
```

---

RootNode	<i>Which node is a tree's root?</i>
----------	-------------------------------------

---

**Description**

RootNode() identifies the root node of a (rooted or unrooted) phylogenetic tree. Unrooted trees are represented internally by a rooted tree with a polytomy at the root.

**Usage**

```
RootNode(x)
```

**Arguments**

x	A tree of class phylo, or its edge matrix; or a list or multiPhylo object containing multiple trees.
---	--

**Value**

RootNode() returns an integer denoting the root node for each tree. Badly conformed trees trigger an error.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Test whether a tree is rooted: [TreeIsRooted\(\)](#)

phangorn::[getRoot\(\)](#)

Other tree navigation: [AncestorEdge\(\)](#), [CladeSizes\(\)](#), [DescendantEdges\(\)](#), [EdgeAncestry\(\)](#), [EdgeDistances\(\)](#), [ListAncestors\(\)](#), [MRCA\(\)](#), [NDescendants\(\)](#), [NodeDepth\(\)](#), [NodeOrder\(\)](#), [NonDuplicateRoot\(\)](#)

**Examples**

```
RootNode(BalancedTree(8))  
RootNode(UnrootTree(BalancedTree(8)))
```

RootTree

*Root or unroot a phylogenetic tree***Description**

RootTree() roots a tree on the smallest clade containing the specified tips; RootOnNode() roots a tree on a specified internal node; UnrootTree() collapses a root node, without the undefined behaviour encountered when using `ape::unroot()` on trees in preorder.

**Usage**

```
RootTree(tree, outgroupTips)
```

```
RootOnNode(tree, node, resolveRoot = FALSE)
```

```
UnrootTree(tree)
```

**Arguments**

tree	A tree of class <code>phylo</code> , or a list of trees of class <code>list</code> or <code>multiPhylo</code> .
outgroupTips	Vector of type character, integer or logical, specifying the names or indices of the tips to include in the outgroup.
node	integer specifying node (internal or tip) to set as the root.
resolveRoot	logical specifying whether to resolve the root node.

**Value**

RootTree() returns a tree of class `phylo`, rooted on the smallest clade that contains the specified tips, with edges and nodes numbered in preorder.

RootOnNode() returns a tree of class `phylo`, rooted on the requested node and ordered in `Preorder`.

UnrootTree() returns tree, in preorder, having collapsed the first child of the root node in each tree.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

- `ape::root()`
- `EnforceOutgroup()`

Other tree manipulation: `AddTip()`, `CollapseNode()`, `ConsensusWithout()`, `DropTip()`, `EnforceOutgroup()`, `LeafLabelInterchange()`, `MakeTreeBinary()`, `RenumberTips()`, `RenumberTree()`, `Renumber()`, `SingleTaxonTree()`, `SortTree()`, `Subtree()`

**Examples**

```

tree <- PectinateTree(8)
plot(tree)
ape::nodeLabels()

plot(RootTree(tree, c('t6', 't7')))

plot(RootOnNode(tree, 12))
plot(RootOnNode(tree, 2))

```

sapply64

*Apply a function that returns 64-bit integers over a list or vector***Description**

Wrappers for members of the `lapply()` family intended for use when a function FUN returns a vector of integer64 objects. `vapply()`, `sapply()` or `replicate()` drop the integer64 class, resulting in a vector of numerics that require conversion back to 64-bit integers. These functions restore the missing class attribute.

**Usage**

```
sapply64(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

```
vapply64(X, FUN, FUN.LEN = 1, ...)
```

```
replicate64(n, expr, simplify = "array")
```

**Arguments**

X	a vector (atomic or list) or an <a href="#">expression</a> object. Other objects (including classed objects) will be coerced by <code>base::as.list</code> .
FUN	the function to be applied to each element of X: see ‘Details’. In the case of functions like <code>+</code> , <code>%*%</code> , the function name must be backquoted or quoted.
...	optional arguments to FUN.
simplify	logical or character string; should the result be simplified to a vector, matrix or higher dimensional array if possible? For <code>sapply</code> it must be named and not abbreviated. The default value, <code>TRUE</code> , returns a vector or matrix if appropriate, whereas if <code>simplify = "array"</code> the result may be an <a href="#">array</a> of “rank” ( <code>=length(dim(.))</code> ) one higher than the result of <code>FUN(X[[i]])</code> .
USE.NAMES	logical; if <code>TRUE</code> and if X is character, use X as <a href="#">names</a> for the result unless it had names already. Since this argument follows ... its name cannot be abbreviated.
FUN.LEN	Integer specifying the length of the output of FUN.
n	integer: the number of replications.
expr	the expression (a <a href="#">language object</a> , usually a call) to evaluate repeatedly.

**Details**

For details of the underlying functions, see [base::lapply\(\)](#).

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

[bit64::integer64\(\)](#)

**Examples**

```
sapply64(as.phylo(1:6, 6), as.TreeNumber)
vapply64(as.phylo(1:6, 6), as.TreeNumber, 1)
set.seed(0)
replicate64(6, as.TreeNumber(RandomTree(6)))
```

---

SingleTaxonTree	<i>Generate a single taxon tree</i>
-----------------	-------------------------------------

---

**Description**

SingleTaxonTree() creates a phylogenetic 'tree' that contains a single taxon.

**Usage**

```
SingleTaxonTree(label)
```

**Arguments**

label            a character vector specifying the label of the tip.

**Value**

SingleTaxonTree() returns a phylo object containing a single tip with the specified label.

**See Also**

Other tree manipulation: [AddTip\(\)](#), [CollapseNode\(\)](#), [ConsensusWithout\(\)](#), [DropTip\(\)](#), [EnforceOutgroup\(\)](#), [LeafLabelInterchange\(\)](#), [MakeTreeBinary\(\)](#), [RenumberTips\(\)](#), [RenumberTree\(\)](#), [Renumber\(\)](#), [RootTree\(\)](#), [SortTree\(\)](#), [Subtree\(\)](#)

Other tree generation functions: [GenerateTree](#), [NJTree\(\)](#), [TreeNumber](#)

**Examples**

```
SingleTaxonTree('Homo_sapiens')
plot(SingleTaxonTree('root') + BalancedTree(4))
```



---

SortTree	<i>Sort tree</i>
----------	------------------

---

### Description

SortTree() sorts each node into a consistent order, so that node rotation does not obscure similarities between similar trees.

### Usage

```
SortTree(tree)
```

### Arguments

tree            A tree of class [phylo](#).

### Details

At each node, clades will be listed in `tree$edge` in decreasing size order.

Clades that contain the same number of leaves are sorted in decreasing order of minimum leaf number, so (2, 3) will occur before (1, 4).

As trees are plotted from 'bottom up', the largest clades will 'sink' to the bottom of a plotted tree.

tree must (presently) be binary ([#25](#)).

### Value

SortTree() returns a tree of class `phylo`, with each node sorted such that the larger clade is first.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

`Preorder()` also rearranges trees into a consistent shape, but based on the index of leaves rather than the size of subtrees.

Other tree manipulation: [AddTip\(\)](#), [CollapseNode\(\)](#), [ConsensusWithout\(\)](#), [DropTip\(\)](#), [EnforceOutgroup\(\)](#), [LeafLabelInterchange\(\)](#), [MakeTreeBinary\(\)](#), [Renumertips\(\)](#), [Renumertree\(\)](#), [Renumber\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [Subtree\(\)](#)

**Examples**

```

messyTree <- as.phylo(10, 6)
plot(messyTree)

sorted <- SortTree(messyTree)
plot(sorted)
ape::nodeLabels()
ape::edgeLabels()

```

---

SplitFrequency

*Frequency of splits*


---

**Description**

SplitFrequency() provides a simple way to count the number of times that bipartition splits, as defined by a reference tree, occur in a forest of trees. May be used to calculate edge ("node") support for majority consensus or bootstrap trees.

**Usage**

```

SplitFrequency(reference, forest)

SplitNumber(tips, tree, tipIndex, powersOf2)

ForestSplits(forest, powersOf2)

TreeSplits(tree)

```

**Arguments**

reference	A tree of class phylo, a Splits object.
forest	a list of trees of class phylo, or a multiPhylo object; or a Splits object. See <a href="#">vignette</a> for possible methods of loading trees into R.
tips	Integer vector specifying the tips of the tree within the chosen split.
tree	A tree of class <a href="#">phylo</a> .
tipIndex	Character vector of tip names, in a fixed order.
powersOf2	Integer vector of same length as tipIndex, specifying a power of 2 to be associated with each tip in turn.

**Details**

If multiple calculations are required, some time can be saved by using the constituent functions (see examples)

**Value**

SplitFrequency() returns the number of trees in forest that contain each split in reference. If reference is a tree of class phylo, then the sequence will correspond to the order of nodes (use ape::nodelabels() to view). Note that the three nodes at the root of the tree correspond to a single split; see the example for how these might be plotted on a tree.

**Functions**

- SplitNumber: Assign a unique integer to each split
- ForestSplits: Frequency of splits in a given forest of trees
- TreeSplits: Deprecated. Listed the splits in a given tree. Use as.Splits instead.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other Splits operations: [LabelSplits\(\)](#), [NSplits\(\)](#), [NTip\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#), [TipsInSplits\(\)](#), [as.Splits\(\)](#), [match\(\)](#)

**Examples**

```
# An example forest of 100 trees, some identical
forest <- as.phylo(c(1, rep(10, 79), rep(100, 15), rep(1000, 5)), nTip = 9)

# Generate an 80% consensus tree
cons <- ape::consensus(forest, p = 0.8)
plot(cons)

splitFreqs <- SplitFrequency(cons, forest)
LabelSplits(cons, splitFreqs, unit = '%',
            col = SupportColor(splitFreqs / 100),
            frame = 'none', pos = 3L)
```

---

SplitInformation

*Phylogenetic information content of splitting leaves into two partitions*

---

**Description**

Calculate the phylogenetic information content (*sensu* Steel & Penny, 2006) of a split, which reflects the probability that a uniformly selected random tree will contain the split: a split that is consistent with a smaller number of trees will have a higher information content.

**Usage**

SplitInformation(A, B)

MultiSplitInformation(partitionSizes)

**Arguments**

A Integer specifying the number of taxa in each partition.  
 B Integer specifying the number of taxa in each partition.  
 partitionSizes Integer vector specifying the number of taxa in each partition of a multi-partition split.

**Details**

SplitInformation() addresses bipartition splits, which correspond to edges in an unrooted phylogeny; MultiSplitInformation() supports splits that subdivide taxa into multiple partitions, which may correspond to multi-state characters in a phylogenetic matrix.

A simple way to characterise trees is to count the number of edges. (Edges are almost, but not quite, equivalent to nodes.) Counting edges (or nodes) provides a quick measure of a tree's resolution, and underpins the Robinson-Foulds tree distance measure. Not all edges, however, are created equal.

An edge splits the leaves of a tree into two subdivisions. The more equal these subdivisions are in size, the more instructive this edge is. Intuitively, the division of mammals from reptiles is a profound revelation that underpins much of zoology; recognizing that two species of bat are more closely related to each other than to any other mammal or reptile is still instructive, but somewhat less fundamental.

Formally, the phylogenetic (Shannon) information content of a split  $S$ ,  $h(S)$ , corresponds to the probability that a uniformly selected random tree will contain the split,  $P(S)$ :  $h(S) = -\log P(S)$ . Base 2 logarithms are typically employed to yield an information content in bits.

As an example, the split AB|CDEF occurs in 15 of the 105 six-leaf trees;  $h(AB|CDEF) = -\log P(AB|CDEF) = -\log(15/105) \sim 2.81$  bits. The split ABC|DEF subdivides the leaves more evenly, and is thus more instructive: it occurs in just nine of the 105 six-leaf trees, and  $h(ABC|DEF) = -\log(9/105) \sim 3.54$  bits.

As the number of leaves increases, a single even split may contain more information than multiple uneven splits – see the examples section below.

Summing the information content of all splits within a tree, perhaps using the 'TreeDist' function `SplitwiseInfo()`, arguably gives a more instructive picture of its resolution than simply counting the number of splits that are present – though with the caveat that splits within a tree are not independent of one another, so some information may be double counted. (This same charge applies to simply counting nodes, too.)

Alternatives would be to count the number of quartets that are resolved, perhaps using the 'Quartet' function `QuartetStates()`, or to use a different take on the information contained within a split, the clustering information: see the 'TreeDist' function `ClusteringInfo()` for details.

**Value**

SplitInformation() and MultiSplitInformation() return the phylogenetic information content, in bits, of a split that subdivides leaves into partitions of the specified sizes.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**References**

- Steel MA, Penny D (2006). “Maximum parsimony and the phylogenetic information in multi-state characters.” In Albert VA (ed.), *Parsimony, Phylogeny, and Genomics*, 163–178. Oxford University Press, Oxford.

**See Also**

Sum the phylogenetic information content of splits within a tree: [TreeDist::SplitwiseInfo\(\)](#)

Sum the clustering information content of splits within a tree: [TreeDist::ClusteringInfo\(\)](#)

Other split information functions: [CharacterInformation\(\)](#), [SplitMatchProbability\(\)](#), [TreesMatchingSplit\(\)](#), [UnrootedTreesMatchingSplit\(\)](#)

**Examples**

```
# Eight leaves can be split evenly:
SplitInformation(4, 4)

# or unevenly, which is less informative:
SplitInformation(2, 6)

# A single split that evenly subdivides 50 leaves contains more information
# that seven maximally uneven splits on the same leaves:
SplitInformation(25, 25)
7 * SplitInformation(2, 48)

# Three ways to split eight leaves into multiple partitions:
MultiSplitInformation(c(2, 2, 4))
MultiSplitInformation(c(2, 3, 3))
MultiSplitInformation(rep(2, 4))
```

---

SplitMatchProbability *Probability of matching this well*

---

**Description**

(Ln)SplitMatchProbability() calculates the probability that two random splits of the sizes provided will be at least as similar as the two specified.

**Usage**

```
SplitMatchProbability(split1, split2)
```

```
LnSplitMatchProbability(split1, split2)
```

**Arguments**

`split1`, `split2` Logical vectors listing terminals in same order, such that each terminal is identified as a member of the ingroup (TRUE) or outgroup (FALSE) of the respective bipartition split.

**Value**

`SplitMatchProbability()` returns a numeric giving the proportion of permissible non-trivial splits that divide the terminals into bipartitions of the sizes given, that match as well as `split1` and `split2` do.

`LnSplitMatchProbability()` returns the natural logarithm of the probability.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other split information functions: [CharacterInformation\(\)](#), [SplitInformation\(\)](#), [TreesMatchingSplit\(\)](#), [UnrootedTreesMatchingSplit\(\)](#)

**Examples**

```
split1 <- as.Splits(c(rep(TRUE, 4), rep(FALSE, 4)))
split2 <- as.Splits(c(rep(TRUE, 3), rep(FALSE, 5)))
SplitMatchProbability(split1, split2)
LnSplitMatchProbability(split1, split2)
```

---

SplitsInBinaryTree      *Maximum splits in an n-leaf tree*

---

**Description**

`SplitsInBinaryTree()` is a convenience function to calculate the number of splits in a fully-resolved (binary) tree with  $n$  leaves.

**Usage**

```
SplitsInBinaryTree(tree)

## S3 method for class 'list'
SplitsInBinaryTree(tree)

## S3 method for class 'multiPhylo'
SplitsInBinaryTree(tree)

## S3 method for class 'numeric'
```

```
SplitsInBinaryTree(tree)

## S3 method for class '`NULL`'
SplitsInBinaryTree(tree)

## Default S3 method:
SplitsInBinaryTree(tree)

## S3 method for class 'Splits'
SplitsInBinaryTree(tree)

## S3 method for class 'phylo'
SplitsInBinaryTree(tree)
```

### Arguments

**tree**                    An object of a supported format that represents a tree or set of trees, from which the number of leaves will be calculated.

### Value

`SplitsInBinaryTree()` returns an integer vector detailing the number of unique non-trivial splits in a binary tree with  $n$  leaves.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

Other tree properties: [ConsensusWithout\(\)](#), [NSplits\(\)](#), [NTip\(\)](#), [TipLabels\(\)](#), [TreeIsRooted\(\)](#)

Other Splits operations: [LabelSplits\(\)](#), [NSplits\(\)](#), [NTip\(\)](#), [SplitFrequency\(\)](#), [TipLabels\(\)](#), [TipsInSplits\(\)](#), [as.Splits\(\)](#), [match\(\)](#)

### Examples

```
tree <- TreeTools::BalancedTree(8)
SplitsInBinaryTree(tree)
SplitsInBinaryTree(TreeTools::as.Splits(tree))
SplitsInBinaryTree(8)
SplitsInBinaryTree(list(tree, tree))
```

---

Stemwardness	<i>'Stemwardness' of a leaf</i>
--------------	---------------------------------

---

### Description

Functions to describe the position of a leaf relative to the root. 'Stemmier' leaves ought to exhibit a smaller root-node distance and a larger sister size,

### Usage

```
SisterSize(tree, tip)

## S3 method for class 'numeric'
SisterSize(tree, tip)

## S3 method for class 'character'
SisterSize(tree, tip)

RootNodeDistance(tree, tip)

## S3 method for class 'numeric'
RootNodeDistance(tree, tip)

## S3 method for class 'character'
RootNodeDistance(tree, tip)

RootNodeDist(tree, tip)
```

### Arguments

tree	A tree of class <code>phylo</code> .
tip	Either a numeric specifying the index of a single tip, or a character specifying its label.

### Details

`RootNodeDistance()` calculates the number of nodes between the chosen leaf and the root of tree. This is an unsatisfactory measure, as the range of possible distances is a function of the shape of the tree. As an example, leaf *X1* in the tree `(.,(.,(.,(X1,(a,b))))))` falls outside the clade  $(a, b)$  and has a root-node distance of 4, whereas leaf *X2* in the tree `(.,((.,(.,)),(b,(X2,a))))` falls within the clade  $(a, b)$ , so should be considered more 'crownwards', yet has a smaller root-node distance (3).





```

pec8 <- PectinateTree(8)

SisterSize(bal8, 3)
SisterSize(pec8, 't3')
SisterSize(RootTree(pec8, 't3'), 't3')

RootNodeDist(bal8, 3)
RootNodeDist(pec8, 't3')
RootNodeDist(RootTree(pec8, 't3'), 't3')

```

---

StringToPhyDat                      *Convert between strings and phyDat objects*

---

### Description

PhyDatToString() converts a [phyDat](#) object as a string; StringToPhyDat() converts a string of character data to a phyDat object.

### Usage

```
StringToPhyDat(string, tips, byTaxon = TRUE)
```

```
StringToPhydat(string, tips, byTaxon = TRUE)
```

```
PhyToString(
  phy,
  parentheses = "{",
  collapse = "",
  ps = "",
  useIndex = TRUE,
  byTaxon = TRUE,
  concatenate = TRUE
)
```

```
PhyDatToString(
  phy,
  parentheses = "{",
  collapse = "",
  ps = "",
  useIndex = TRUE,
  byTaxon = TRUE,
  concatenate = TRUE
)
```

```
PhydatToString(
  phy,
  parentheses = "{",
  collapse = "",

```

```

    ps = "",
    useIndex = TRUE,
    byTaxon = TRUE,
    concatenate = TRUE
  )

```

### Arguments

string	String of tokens, optionally containing whitespace, with no terminating semi-colon.
tips	Character vector corresponding to the names (in order) of each taxon in the matrix.
byTaxon	Logical. If TRUE, write one taxon followed by the next. If FALSE, write one character followed by the next.
phy	An object of class <a href="#">phyDat</a> .
parentheses	Character specifying format of parentheses with which to surround ambiguous tokens. Choose from: { (default), [, (, <.
collapse	Character specifying text, perhaps ,, with which to separate multiple tokens within parentheses.
ps	Character specifying text, perhaps ;, to append to the end of the string.
useIndex	Logical (default: TRUE) specifying whether to print duplicate characters multiple times, as they appeared in the original matrix.
concatenate	Logical specifying whether to concatenate all characters/taxa into a single string, or to return a separate string for each entry.

### Value

StringToPhyDat() returns an object of class phyDat.

PhyToString() returns a character vector listing a text representation of the phylogenetic character state for each taxon in turn.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

Other phylogenetic matrix conversion functions: [MatrixToPhyDat\(\)](#)

### Examples

```

StringToPhyDat("-?01231230?-", c('Lion', 'Gazelle'), byTaxon = TRUE)
# encodes the following matrix:
# Lion      -?0123
# Gazelle  1230?-

fileName <- paste0(system.file(package='TreeTools'),

```

```

                                '/extdata/input/dataset.nex')
phyDat <- ReadAsPhyDat(fileName)
PhyToString(phyDat, concatenate = FALSE)

```

---

Subsplit                      *Subset of a split on fewer leaves*

---

### Description

Subsplit() removes leaves from a Splits object.

### Usage

```
Subsplit(splits, tips, keepAll = FALSE, unique = TRUE)
```

### Arguments

splits	An object of class <a href="#">Splits</a> .
tips	A vector specifying a subset of the leaf labels applied to split.
keepAll	logical specifying whether to keep entries that define trivial splits (i.e. splits of zero or one leaf) on the subset of leaves.
unique	logical specifying whether to remove duplicate splits.

### Value

Subsplit() returns an object of class [Splits](#), defined on the leaves tips.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

Other split manipulation functions: [TrivialSplits\(\)](#)

### Examples

```

splits <- as.Splits(PectinateTree(letters[1:9]))
splits
efgh <- Subsplit(splits, tips = letters[5:8], keepAll = TRUE)
summary(efgh)

TrivialSplits(efgh)

summary(Subsplit(splits, tips = letters[5:8], keepAll = FALSE))

```

---

Subtree	<i>Extract a subtree</i>
---------	--------------------------

---

### Description

Subtree() safely extracts a clade from a phylogenetic tree.

### Usage

```
Subtree(tree, node)
```

### Arguments

tree	A tree of class <code>phylo</code> , with internal numbering in cladewise order (use <code>Preorder(tree)</code> or (slower) <code>Cladewise(tree)</code> ).
node	The number of the node at the base of the clade to be extracted.

### Details

Modified from the **ape** function `extract.clade`, which sometimes behaves erratically. Unlike `extract.clade`, this function supports the extraction of 'clades' that constitute a single tip.

### Value

Subtree() returns a tree of class `phylo` that represents a clade extracted from the original tree.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

Other tree manipulation: [AddTip\(\)](#), [CollapseNode\(\)](#), [ConsensusWithout\(\)](#), [DropTip\(\)](#), [EnforceOutgroup\(\)](#), [LeafLabelInterchange\(\)](#), [MakeTreeBinary\(\)](#), [RenumberTips\(\)](#), [RenumberTree\(\)](#), [Renumber\(\)](#), [RootTree\(\)](#), [SingleTaxonTree\(\)](#), [SortTree\(\)](#)

### Examples

```
tree <- Preorder(BalancedTree(8))
plot(tree)
ape::nodelabels()
ape::nodelabels(13, 13, bg='yellow')

plot(Subtree(tree, 13))
```

---

SupportColour	<i>Colour for node support value</i>
---------------	--------------------------------------

---

### Description

Colour value with which to display node support.

### Usage

```
SupportColour(
  support,
  show1 = TRUE,
  scale = rev(diverge_hcl(101, h = c(260, 0), c = 100, l = c(50, 90), power = 1)),
  outOfRange = "red"
)
```

```
SupportColor(
  support,
  show1 = TRUE,
  scale = rev(diverge_hcl(101, h = c(260, 0), c = 100, l = c(50, 90), power = 1)),
  outOfRange = "red"
)
```

### Arguments

support	A numeric vector of values in the range 0–1.
show1	Logical specifying whether to display values of 1. A transparent white will be returned if FALSE.
scale	101-element vector listing colours in sequence. Defaults to a diverging HCL scale.
outOfRange	Colour to use if results are outside the range 0–1.

### Value

SupportColour() returns the appropriate value from scale, or outOfRange if a value is outwith the valid range.

### See Also

Use in conjunction with [LabelSplits\(\)](#) to colour split labels, possibly calculated using [SplitFrequency\(\)](#).

### Examples

```
SupportColour((-1):4 / 4, show1 = FALSE)

# An example forest of 100 trees, some identical
forest <- as.phylo(c(1, rep(10, 79), rep(100, 15), rep(1000, 5)), nTip = 9)
```

```
# Generate an 80% consensus tree
cons <- ape::consensus(forest, p = 0.8)
plot(cons)

splitFreqs <- SplitFrequency(cons, forest)
LabelSplits(cons, splitFreqs, unit = '%',
            col = SupportColor(splitFreqs / 100),
            frame = 'none', pos = 3L)
```

---

TipLabels

*Extract tip labels*

---

### Description

TipLabels() extracts labels from an object: for example, names of taxa in a phylogenetic tree or data matrix. AllTipLabels() extracts all labels, where entries of a list of trees may pertain to different taxa.

### Usage

```
TipLabels(x, single = TRUE)

## S3 method for class 'matrix'
TipLabels(x, single = TRUE)

## S3 method for class 'phylo'
TipLabels(x, single = TRUE)

## Default S3 method:
TipLabels(x, single = TRUE)

## S3 method for class 'TreeNumber'
TipLabels(x, single = TRUE)

## S3 method for class 'Splits'
TipLabels(x, single = TRUE)

## S3 method for class 'list'
TipLabels(x, single = FALSE)

AllTipLabels(x)

## S3 method for class 'list'
AllTipLabels(x)

## S3 method for class 'multiPhylo'
```

```

AllTipLabels(x)

## S3 method for class 'phylo'
AllTipLabels(x)

## S3 method for class 'Splits'
AllTipLabels(x)

## S3 method for class 'TreeNumber'
AllTipLabels(x)

## S3 method for class 'matrix'
AllTipLabels(x)

## S3 method for class 'multiPhylo'
TipLabels(x, single = FALSE)

## S3 method for class 'character'
TipLabels(x, single = TRUE)

## S3 method for class 'numeric'
TipLabels(x, single = TRUE)

## S3 method for class 'phyDat'
TipLabels(x, single = TRUE)

## Default S3 method:
TipLabels(x, single = TRUE)

```

### Arguments

x	An object of a supported class (see Usage section above).
single	Logical specifying whether to report the labels for the first object only (TRUE), or for each object in a list (FALSE).

### Value

TipLabels() returns a character vector listing the tip labels appropriate to x. If x is a single integer, this will be a vector t1, t2 ... tx, to match the default of `ape::rtree()`.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

Other tree properties: [ConsensusWithout\(\)](#), [NSplits\(\)](#), [NTip\(\)](#), [SplitsInBinaryTree\(\)](#), [TreeIsRooted\(\)](#)  
 Other Splits operations: [LabelSplits\(\)](#), [NSplits\(\)](#), [NTip\(\)](#), [SplitFrequency\(\)](#), [SplitsInBinaryTree\(\)](#), [TipsInSplits\(\)](#), [as.Splits\(\)](#), [match\(\)](#)



**Examples**

```

TipLabels(BalancedTree(letters[5:1]))
TipLabels(5)

data('Lobo')
head(TipLabels(Lobo.phy))

AllTipLabels(c(BalancedTree(4), PectinateTree(8)))

```

---

TipLabels	<i>Tips contained within splits</i>
-----------	-------------------------------------

---

**Description**

TipLabels() specifies the number of tips that occur within each bipartition split in a Splits object.

**Usage**

```

TipLabels(splits, keep.names = TRUE, ...)

## S3 method for class 'Splits'
TipLabels(splits, keep.names = TRUE, ...)

## S3 method for class 'phylo'
TipLabels(splits, keep.names = TRUE, ...)

SplitImbalance(splits, keep.names = TRUE, ...)

## S3 method for class 'Splits'
SplitImbalance(splits, keep.names = TRUE, ...)

## S3 method for class 'phylo'
SplitImbalance(splits, keep.names = TRUE, ...)

```

**Arguments**

splits	Object of class Splits or phylo.
keep.names	Logical specifying whether to include the names of splits in the output.
...	Additional parameters to pass to as.Splits().

**Value**

TipsInSplits() returns a named vector of integers, specifying the number of tips contained within each split in splits.

SplitImbalance() returns a named vector of integers, specifying the number of leaves within a split that are not 'balanced' by a leaf outside it; i.e. a split that divides leaves evenly has an imbalance of zero; one that splits two tips from ten has an imbalance of  $10 - 2 = 8$ .

**See Also**

Other Splits operations: [LabelSplits\(\)](#), [NSplits\(\)](#), [NTip\(\)](#), [SplitFrequency\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#), [as.Splits\(\)](#), [match\(\)](#)

**Examples**

```
tree <- PectinateTree(8)
splits <- as.Splits(tree)
TipsInSplits(splits)

plot(tree)
LabelSplits(tree, as.character(splits), frame = 'none', pos = 3L, cex = 0.7)
LabelSplits(tree, TipsInSplits(splits), unit = ' tips', frame = 'none',
            pos = 1L)
```

---

TotalCopheneticIndex    *Total Cophenetic Index*

---

**Description**

TotalCopheneticIndex() calculates the total cophenetic index (Mir *et al.* 2013) for any tree, a measure of its balance; TCIContext() lists its possible values.

**Usage**

```
TotalCopheneticIndex(x)

TCIContext(x)

## S3 method for class 'numeric'
TCIContext(x)
```

**Arguments**

x                    A tree of class phylo, its \$edge property, or a list thereof.

## Details

The Total Cophenetic Index is a measure of tree balance – i.e. whether a (phylogenetic) tree comprises symmetric pairs of nodes, or has a pectinate 'caterpillar' shape. The index has a greater resolution power than Sackin's and Colless' indices, and can be applied to trees that are not perfectly resolved.

For a tree with  $n$  leaves, the Total Cophenetic Index can take values of 0 to  $\text{choose}(n, 3)$ . The minimum value is higher for a perfectly resolved (i.e. dichotomous) tree (see Lemma 14 of Mir *et al.* 2013). Formulae to calculate the expected values under the Yule and Uniform models of evolution are given in Theorems 17 and 23.

Full details are provided by Mir *et al.* (2013).

## Value

TotalCopheneticIndex() returns an integer denoting the total cophenetic index.

TCIContext() returns a data frame detailing the maximum and minimum value obtainable for the Total Cophenetic Index for rooted binary trees with the number of leaves of the given tree, and the expected value under the Yule and Uniform models. The variance of the expected value is given under the Yule model, but cannot be obtained by calculation for the Uniform model.

## Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

## References

Mir A, Rosselló F, Rotger LA (2013). "A new balance index for phylogenetic trees." *Mathematical Biosciences*, **241**(1), 125–136. doi: [10.1016/j.mbs.2012.10.005](https://doi.org/10.1016/j.mbs.2012.10.005), <https://doi.org/10.1016/j.mbs.2012.10.005>.

## See Also

cophen.index() in the package 'CollessLike' provides an alternative implementation of this index and its predecessors.

Other tree characterization functions: [CladisticInfo\(\)](#), [Stemwardness](#)

## Examples

```
# Balanced trees have the minimum index for a binary tree;
# Pectinate trees the maximum:
TCIContext(8)
TotalCopheneticIndex(PectinateTree(8))
TotalCopheneticIndex(BalancedTree(8))
TotalCopheneticIndex(StarTree(8))

# Examples from Mir et al. (2013):
tree12 <- ape::read.tree(text='(1, (2, (3, (4, 5)))));' #Fig. 4, tree 12
TotalCopheneticIndex(tree12) # 10
tree8 <- ape::read.tree(text='((1, 2, 3, 4), 5);' #Fig. 4, tree 8
```

```
TotalCopheneticIndex(tree8) # 6
TCIContext(tree8)
TCIContext(5L) # Context for a tree with 5 leaves.
```

---

TreeIsRooted	<i>Is tree rooted?</i>
--------------	------------------------

---

### Description

TreeIsRooted() is a fast alternative to `ape::is.rooted()`.

### Usage

```
TreeIsRooted(tree)
```

### Arguments

tree           A phylogenetic tree of class `phylo`.

### Value

TreeIsRooted() returns a logical specifying whether a root node is resolved.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

Other tree properties: [ConsensusWithout\(\)](#), [NSplits\(\)](#), [NTip\(\)](#), [SplitsInBinaryTree\(\)](#), [TipLabels\(\)](#)

### Examples

```
TreeIsRooted(BalancedTree(6))
TreeIsRooted(UnrootTree(BalancedTree(6)))
```

---

TreeNumber                      *Unique integer indices for bifurcating tree topologies*

---

### Description

Functions converting between phylogenetic trees and their unique decimal representation.

### Usage

```
as.TreeNode(x, ...)
```

```
## S3 method for class 'phylo'
as.TreeNode(x, ...)
```

```
## S3 method for class 'multiPhylo'
as.TreeNode(x, ...)
```

```
## S3 method for class 'character'
as.TreeNode(x, nTip, tipLabels = TipLabels(nTip), ...)
```

```
## S3 method for class 'numeric'
as.phylo(x, nTip = attr(x, "nTip"), tipLabels = attr(x, "tip.label"), ...)
```

```
## S3 method for class 'TreeNode'
as.phylo(x, nTip = attr(x, "nTip"), tipLabels = attr(x, "tip.label"), ...)
```

### Arguments

x	Integer identifying the tree (see details).
...	Additional parameters for consistency with S3 methods (unused).
nTip	Integer specifying number of leaves in the tree.
tipLabels	Character vector listing the labels assigned to each tip in a tree, perhaps obtained using <a href="#">TipLabels()</a> .

### Details

There are  $N_{\text{Unrooted}}(n)$  unrooted trees with  $n$  leaves. As such, each  $n$ -leaf tree can be uniquely identified by a non-negative integer  $x < N_{\text{Unrooted}}(n)$ .

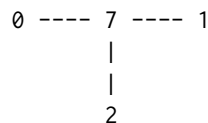
This integer can be converted by a tree by treating it as a mixed-base number, with bases 1, 3, 5, 7, ... ( $2n - 5$ ).

Each digit of this mixed base number corresponds to a leaf, and determines the location on a growing tree to which that leaf should be added.

We start with a two-leaf tree, and treat 0 as the origin of the tree.

0 ---- 1

We add leaf 2 by breaking an edge and inserting a node (numbered  $2 + n\text{Tip} - 1$ ). In this example, we'll work up to a six-leaf tree; this node will be numbered  $2 + 6 - 1 = 7$ . There is only one edge on which leaf 2 can be added. Let's add node 7 and leaf 2:



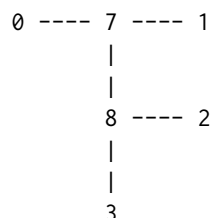
There are now three edges on which leaf 3 can be added. Our options are:

Option 0: the edge leading to 1;

Option 1: the edge leading to 2;

Option 2: the edge leading to 7.

If we select option 1, we produce:



1 is now the final digit of our mixed-base number.

There are five places to add leaf 4:

Option 0: the edge leading to 1;

Option 1: the edge leading to 2;

Option 2: the edge leading to 3;

Option 3: the edge leading to 7;

Option 4: the edge leading to 8.

If we chose option 3, then 3 would be the penultimate digit of our mixed-base number.

If we chose option 0 for the next two additions, we could specify this tree with the mixed-base number 0021. We can convert this into decimal:

$$\begin{aligned}
 &0 \times (1 \times 3 \times 5 \times 9) + \\
 &0 \times (1 \times 3 \times 5) + \\
 &3 \times (1 \times 3) + \\
 &1 \times (1) \\
 &= 10
 \end{aligned}$$

Note that the hyperexponential nature of tree space means that there are  $> 2^{30}$  unique 12-leaf trees. As integers  $> 2^{31}$  are not supported by R, numbers representing larger trees are represented internally as a vector of nine-digit integer 'chunks' and passed to the underlying C code, where they are combined into a single 64-bit integer. This allows trees with up to 42 leaves to be accommodated.

**Value**

`as.TreeNumber()` returns an object of class `TreeNumber`, which comprises a numeric vector, whose elements represent successive nine-digit chunks of the decimal integer corresponding to the tree topology (in big endian order). The `TreeNumber` object has attributes `nTip` and `tip.labels`.

`as.phylo.numeric()` returns a tree of class `phylo`.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**References**

Based on a concept by John Tromp, employed in Li *et al.* 1996.

Li M, Tromp J, Zhang L (1996). "Some notes on the nearest neighbour interchange distance." In Goos G, Hartmanis J, Leeuwen J, Cai J, Wong CK (eds.), *Computing and Combinatorics*, volume 1090, 343–351. Springer, Berlin, Heidelberg. ISBN 978-3-540-61332-9 978-3-540-68461-9, doi: [10.1007/3540613323\\_168](https://doi.org/10.1007/3540613323_168), [https://doi.org/10.1007/3-540-61332-3\\_168](https://doi.org/10.1007/3-540-61332-3_168).

**See Also**

Describe the shape of a tree topology, independent of leaf labels: [TreeShape\(\)](#)

Other tree generation functions: [GenerateTree](#), [NJTree\(\)](#), [SingleTaxonTree\(\)](#)

**Examples**

```
tree <- as.phylo(10, nTip = 6)
plot(tree)
as.TreeNumber(tree)

# Larger trees:
as.TreeNumber(BalancedTree(19))

# If > 9 digits, represent the tree number as a string.
treeNumber <- as.TreeNumber("1234567890123", nTip = 14)
tree <- as.phylo(treeNumber)
as.phylo(0:2, nTip = 6, tipLabels = letters[1:6])
```

---

TreesMatchingSplit      *Number of trees matching a bipartition split*

---

**Description**

Calculates the number of unrooted bifurcated trees that are consistent with a bipartition split that divides taxa into groups of size A and B.

**Usage**

```
TreesMatchingSplit(A, B)
```

```
LnTreesMatchingSplit(A, B)
```

```
Log2TreesMatchingSplit(A, B)
```

**Arguments**

A, B                    Integer specifying the number of taxa in each partition.

**Value**

TreesMatchingSplit() returns a numeric specifying the number of trees that are compatible with the given split.

LnTreesMatchingSplit() and Log2TreesMatchingSplit() give the natural and base-2 logarithms of this number.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other split information functions: [CharacterInformation\(\)](#), [SplitInformation\(\)](#), [SplitMatchProbability\(\)](#), [UnrootedTreesMatchingSplit\(\)](#)

**Examples**

```
TreesMatchingSplit(5, 6)
LnTreesMatchingSplit(5, 6)
Log2TreesMatchingSplit(5, 6)
```

---

TreesMatchingTree	<i>Number of trees containing a tree</i>
-------------------	--

---

**Description**

TreesMatchingTree() calculates the number of unrooted binary trees that are consistent with a tree topology on the same leaves.

**Usage**

```
TreesMatchingTree(tree)
```

```
LnTreesMatchingTree(tree)
```

```
Log2TreesMatchingTree(tree)
```



**Arguments**

tree                    A tree of class [phylo](#).

**Details**

Remember to unroot a tree first if the position of its root is arbitrary.

**Value**

TreesMatchingTree() returns a numeric specifying the number of unrooted binary trees that contain all the edges present in the input tree.

LnTreesMatchingTree() gives the natural logarithm of this number.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other tree information functions: [CladisticInfo\(\)](#), [NRooted\(\)](#)

**Examples**

```
partiallyResolvedTree <- CollapseNode(BalancedTree(8), 12:15)
TreesMatchingTree(partiallyResolvedTree)
LnTreesMatchingTree(partiallyResolvedTree)

# Number of rooted trees:
rootedTree <- AddTip(partiallyResolvedTree, where = 0)
TreesMatchingTree(partiallyResolvedTree)
```

---

TrivialSplits

*Identify and remove trivial splits*

---

**Description**

TrivialSplits() identifies trivial splits (which separate one or zero leaves from all others); WithoutTrivialSplits() removes them from a Splits object.

**Usage**

```
TrivialSplits(splits, nTip = attr(splits, "nTip"))
```

```
WithoutTrivialSplits(splits, nTip = attr(splits, "nTip"))
```

**Arguments**

splits                    An object of class [Splits](#).

nTip                      Integer specifying number of tips (leaves).

**Value**

TrivialSplits() returns a logical vector specifying whether each split in splits is trivial, i.e. includes or excludes only a single tip or no tips at all.

WithoutTrivialSplits() returns a Splits object with trivial splits removed.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other split manipulation functions: [Subsplit\(\)](#)

**Examples**

```
splits <- as.Splits(PectinateTree(letters[1:9]))
efgh <- Subsplit(splits, tips = letters[5:8], keepAll = TRUE)
summary(efgh)
```

```
TrivialSplits(efgh)
summary(WithoutTrivialSplits(efgh))
```

---

UnrootedTreesMatchingSplit

*Number of trees consistent with split*

---

**Description**

Calculates the number of unrooted bifurcating trees consistent with the specified multi-partition split, using the formula of Carter *et al.* (1990).

**Usage**

```
UnrootedTreesMatchingSplit(...)
```

```
LnUnrootedTreesMatchingSplit(...)
```

```
Log2UnrootedTreesMatchingSplit(...)
```

**Arguments**

... A series or vector of integers listing the number of tips in each of a number of tree splits (e.g. bipartitions). For example, 3, 5 states that a character divides a set of eight tips into a group of three and a group of five.

**Value**

UnrootedTreesMatchingSplit() returns an integer specifying the number of unrooted bifurcating trees consistent with the specified split.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**References**

See Theorem 2 in Carter M, Hendy M, Penny D, Székely LA, Wormald NC (1990). “On the distribution of lengths of evolutionary trees.” *SIAM Journal on Discrete Mathematics*, **3**(1), 38–47. doi: [10.1137/0403005](https://doi.org/10.1137/0403005), <https://doi.org/10.1137/0403005>.

**See Also**

Other split information functions: [CharacterInformation\(\)](#), [SplitInformation\(\)](#), [SplitMatchProbability\(\)](#), [TreesMatchingSplit\(\)](#)

**Examples**

```
UnrootedTreesMatchingSplit(c(3, 5))
UnrootedTreesMatchingSplit(3, 2, 1, 2)
```

---

UnshiftTree

*Add tree to start of list*

---

**Description**

UnshiftTree() adds a phylogenetic tree to the start of a list of trees. This is useful where the class of a list of trees is unknown.

**Usage**

```
UnshiftTree(add, treeList)
```

**Arguments**

add                   Tree to add to the list, of class [phylo](#).  
treeList               A list of trees, of class [list](#), [multiPhylo](#), or, if a single tree, [phylo](#).

**Details**

Caution: adding a tree to a multiPhylo object whose own attributes apply to all trees, for example trees read from a Nexus file, causes data to be lost.

**Value**

UnshiftTree() returns a list of class `list` or `multiPhylo` (following the original class of `treeList`), whose first element is the tree specified as `'add'`.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**Examples**

```
forest <- as.phylo(0:5, 6)
tree <- BalancedTree(6)

UnshiftTree(tree, forest)
UnshiftTree(tree, tree)
```

---

WriteTntCharacters      *Write morphological character matrix to TNT file*

---

**Description**

Write morphological character matrix to TNT file

**Usage**

```
WriteTntCharacters(
  dataset,
  filepath = NULL,
  comment = "Dataset written by `TreeTools::WriteTntCharacters()`",
  types = NULL,
  pre = "",
  post = ""
)

## S3 method for class 'phyDat'
WriteTntCharacters(
  dataset,
  filepath = NULL,
  comment = "Dataset written by `TreeTools::WriteTntCharacters()`",
  types = NULL,
  pre = "",
  post = ""
)

## S3 method for class 'matrix'
WriteTntCharacters(
```

```

dataset,
filepath = NULL,
comment = "Dataset written by `TreeTools::WriteTntCharacters()`",
types = NULL,
pre = "",
post = ""
)

```

### Arguments

dataset	Morphological dataset of class phyDat or matrix.
filepath	Path to file; if NULL, returns a character vector.
comment	Optional comment with which to entitle matrix.
types	Optional list specifying where different data types begin. c(num = 1, dna = 10) sets characters 1..9 as numeric, 10..end as DNA.
pre, post	Character vector listing text to print before and after the character matrix. Specify pre = 'piwe='; if the matrix is to be analysed using extended implied weighting (xpiwe=).

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

[ReadTntCharacters\(\)](#)

### Examples

```

data('Lobo', package = 'TreeTools')

WriteTntCharacters(Lobo.phy)

# Read with extended implied weighting
WriteTntCharacters(Lobo.phy, pre = 'piwe=10;', post = 'xpiwe=;')

# Write to a file with:
# WriteTntCharacters(Lobo.phy, 'example_file.tnt')

```

# Index

- \* **Splits operations**
  - as.Splits, 9
  - LabelSplits, 25
  - match, 31
  - NSplits, 43
  - NTip, 44
  - SplitFrequency, 58
  - SplitsInBinaryTree, 62
  - TipLabels, 71
  - TipsInSplits, 73
- \* **datasets**
  - brewer, 10
  - doubleFactorials, 19
  - Lobo.data, 29
  - logDoubleFactorials, 29
  - nRootedShapes, 42
- \* **double factorials**
  - DoubleFactorial, 18
  - doubleFactorials, 19
  - logDoubleFactorials, 29
- \* **methods**
  - match, 31
- \* **pairwise tree distances**
  - PairwiseDistances, 45
- \* **phylogenetic matrix conversion functions**
  - StringToPhyDat, 66
- \* **split information functions**
  - CharacterInformation, 11
  - SplitInformation, 59
  - SplitMatchProbability, 61
  - TreesMatchingSplit, 79
  - UnrootedTreesMatchingSplit, 82
- \* **split information function**
  - NRooted, 40
- \* **split manipulation functions**
  - Subsplit, 68
  - TrivialSplits, 81
- \* **tree characterization functions**
  - CladisticInfo, 13
  - Stemwardness, 64
  - TotalCopheneticIndex, 74
- \* **tree generation functions**
  - GenerateTree, 24
  - NJTree, 37
  - SingleTaxonTree, 56
  - TreeNumber, 77
- \* **tree information functions**
  - CladisticInfo, 13
  - NRooted, 40
  - TreesMatchingTree, 80
- \* **tree manipulation**
  - AddTip, 3
  - CollapseNode, 14
  - ConsensusWithout, 15
  - DropTip, 20
  - EnforceOutgroup, 23
  - LeafLabelInterchange, 26
  - MakeTreeBinary, 30
  - ReNumber, 51
  - ReNumberTips, 52
  - RootTree, 54
  - SingleTaxonTree, 56
  - SortTree, 57
  - Subtree, 69
- \* **tree navigation**
  - CladeSizes, 12
  - DescendantEdges, 17
  - EdgeAncestry, 21
  - EdgeDistances, 22
  - ListAncestors, 27
  - MRCA, 32
  - NDescendants, 35
  - NodeDepth, 37
  - NodeOrder, 38
  - RootNode, 53
- \* **tree properties**
  - ConsensusWithout, 15
  - NSplits, 43

- NTip, 44
- SplitsInBinaryTree, 62
- TipLabels, 71
- TreeIsRooted, 76
- \* **tree**
  - AddTip, 3
  - SingleTaxonTree, 56
- %in% (match), 31
- AddTip, 3, 15, 16, 20, 23, 27, 30, 51, 52, 54, 56, 57, 69
- AddTip(), 22, 25
- AddTipEverywhere (AddTip), 3
- AllAncestors (ListAncestors), 27
- AllDescendantEdges (DescendantEdges), 17
- AllTipLabels (TipLabels), 71
- AncestorEdge, 12, 18, 21, 22, 28, 32, 36, 38, 39, 53
- ape::consensus(), 16
- ape::drop.tip(), 20
- ape::edgelabels(), 25
- ape::mst(), 34
- ape::multi2di(), 30
- ape::node.depth, 38
- ape::Ntip(), 44
- ape::root(), 54
- ape::unroot, 54
- ape::write.tree(), 8
- ApeTime, 5
- array, 55
- ArtEx (ArtificialExtinction), 6
- ArtificialExtinction, 6
- as.list, 55
- as.logical.Splits (as.Splits), 9
- as.Newick, 8, 36
- as.phylo.numeric (TreeNumber), 77
- as.phylo.TreeNumber (TreeNumber), 77
- as.Splits, 9, 26, 31, 44, 45, 59, 63, 72, 74
- as.Splits(), 25
- as.TreeNumber (TreeNumber), 77
- BalancedTree (GenerateTree), 24
- base::lapply(), 56
- bind.tree, 4
- bit64::integer64(), 56
- brewer, 10
- CharacterInformation, 11, 61, 62, 80, 83
- CladeSizes, 12, 18, 21, 22, 28, 32, 36, 38, 39, 53
- Cladewise, 21, 51, 69
- CladisticInfo, 13, 42, 65, 75, 81
- CladisticInformation (CladisticInfo), 13
- CollapseEdge (CollapseNode), 14
- CollapseNode, 4, 14, 16, 20, 23, 27, 30, 51, 52, 54, 56, 57, 69
- ConsensusWithout, 4, 15, 15, 20, 23, 27, 30, 44, 45, 51, 52, 54, 56, 57, 63, 69, 72, 76
- DescendantEdges, 12, 17, 21, 22, 28, 32, 36, 38, 39, 53
- DoubleFactorial, 18, 19, 30
- DoubleFactorial64 (DoubleFactorial), 18
- doubleFactorials, 19, 19, 30
- DropTip, 4, 15, 16, 20, 23, 27, 30, 51, 52, 54, 56, 57, 69
- EdgeAncestry, 12, 18, 21, 22, 28, 32, 36, 38, 39, 53
- EdgeDistances, 12, 18, 21, 22, 28, 32, 36, 38, 39, 53
- edgelabels, 14, 17
- EnforceOutgroup, 4, 15, 16, 20, 23, 27, 30, 51, 52, 54, 56, 57, 69
- EnforceOutgroup(), 54
- expression, 55
- extract.clade, 69
- ForestSplits (SplitFrequency), 58
- GenerateTree, 24, 37, 56, 79
- getRoot(), 53
- IC1Spr (N1Spr), 34
- in.Splits (match), 31
- LabelSplits, 10, 25, 31, 44, 45, 59, 63, 72, 74
- LabelSplits(), 70
- language object, 55
- lapply(), 55
- LeafLabelInterchange, 4, 15, 16, 20, 23, 26, 30, 51, 52, 54, 56, 57, 69
- legend(), 16
- ListAncestors, 12, 18, 21, 22, 27, 32, 36, 38, 39, 53
- ListAncestors(), 32
- LnDoubleFactorial (DoubleFactorial), 18

- LnRooted (NRooted), 40
- LnSplitMatchProbability  
(SplitMatchProbability), 61
- LnTreesMatchingSplit  
(TreesMatchingSplit), 79
- LnTreesMatchingTree  
(TreesMatchingTree), 80
- LnUnrooted (NRooted), 40
- LnUnrootedMult (NRooted), 40
- LnUnrootedSplits (NRooted), 40
- LnUnrootedTreesMatchingSplit  
(UnrootedTreesMatchingSplit),  
82
- Lobo.data, 29
- Lobo.phy (Lobo.data), 29
- Log2DoubleFactorial (DoubleFactorial),  
18
- Log2Rooted (NRooted), 40
- Log2TreesMatchingSplit  
(TreesMatchingSplit), 79
- Log2TreesMatchingTree  
(TreesMatchingTree), 80
- Log2Unrooted (NRooted), 40
- Log2UnrootedMult (NRooted), 40
- Log2UnrootedSplits (NRooted), 40
- Log2UnrootedTreesMatchingSplit  
(UnrootedTreesMatchingSplit),  
82
- LogDoubleFactorial (DoubleFactorial), 18
- logDoubleFactorials, 19, 29
- MakeTreeBinary, 4, 15, 16, 20, 23, 27, 30, 51,  
52, 54, 56, 57, 69
- MarkMissing (ConsensusWithout), 15
- match, 10, 26, 31, 44, 45, 59, 63, 72, 74
- match(), 31
- MatrixToPhyDat, 67
- MatrixToPhyDat(), 48
- MRCA, 12, 18, 21, 22, 28, 32, 36, 38, 39, 53
- MSTEdges, 33
- MSTLength (MSTEdges), 33
- multiPhylo, 83
- MultiSplitInformation  
(SplitInformation), 59
- N1Spr, 34
- names, 55
- NDescendants, 12, 18, 21, 22, 28, 32, 35, 38,  
39, 53
- NewickTree, 36
- NewickTree(), 8
- NJTree, 24, 37, 56, 79
- NodeDepth, 12, 18, 21, 22, 28, 32, 36, 37, 39,  
53
- nodelabels, 14
- NodeOrder, 12, 18, 21, 22, 28, 32, 36, 38, 38,  
53
- NonDuplicateRoot, 12, 18, 21, 22, 28, 32, 36,  
38, 39, 53
- NPartitionPairs, 39
- NPartitions (NSplits), 43
- NRooted, 14, 40, 81
- NRooted64 (NRooted), 40
- nRootedShapes, 42
- NSplits, 10, 16, 26, 31, 43, 45, 59, 63, 72, 74,  
76
- NTip, 10, 16, 26, 31, 44, 44, 59, 63, 72, 74, 76
- NUnrooted (NRooted), 40
- NUnrooted64 (NRooted), 40
- NUnrootedMult (NRooted), 40
- nUnrootedShapes (nRootedShapes), 42
- NUnrootedSplits (NRooted), 40
- PairwiseDistances, 45
- PectinateTree (GenerateTree), 24
- PhyDat (ReadCharacters), 47
- phyDat, 37, 47, 66, 67
- PhyDatToString (StringToPhyDat), 66
- PhydatToString (StringToPhyDat), 66
- phylo, 4, 9, 12, 14, 17, 20–22, 25–27, 30, 35,  
36, 51, 52, 54, 57, 58, 64, 69, 81, 83
- PhylogeneticInfo (CladisticInfo), 13
- PhylogeneticInformation  
(CladisticInfo), 13
- PhyToString (StringToPhyDat), 66
- Preorder, 20, 21, 28, 54, 69
- print.TreeNumber, 46
- RandomTree (GenerateTree), 24
- read.nexus.data, 47
- read.table(), 48
- ReadAsPhyDat (ReadCharacters), 47
- ReadCharacters, 47
- ReadTntAsPhyDat (ReadCharacters), 47
- ReadTntCharacters (ReadCharacters), 47
- ReadTntCharacters(), 85
- ReadTntTree, 48



- Renumber, [4](#), [15](#), [16](#), [20](#), [23](#), [27](#), [30](#), [51](#), [52](#), [54](#),  
[56](#), [57](#), [69](#)
- RenumberTips, [4](#), [15](#), [16](#), [20](#), [23](#), [27](#), [30](#), [51](#),  
[52](#), [54](#), [56](#), [57](#), [69](#)
- RenumberTips(), [8](#)
- RenumberTree, [4](#), [15](#), [16](#), [20](#), [23](#), [27](#), [30](#), [51](#),  
[52](#), [54](#), [56](#), [57](#), [69](#)
- replicate64 (sapply64), [55](#)
- RootNode, [12](#), [18](#), [21](#), [22](#), [28](#), [32](#), [36](#), [38](#), [39](#), [53](#)
- RootNodeDist (Stemwardness), [64](#)
- RootNodeDistance (Stemwardness), [64](#)
- RootOnNode (RootTree), [54](#)
- RootTree, [4](#), [15](#), [16](#), [20](#), [23](#), [27](#), [30](#), [51](#), [52](#), [54](#),  
[56](#), [57](#), [69](#)
- RootTree(), [23](#)
- rtree, [72](#)
  
- sapply64, [55](#)
- SingleTaxonTree, [4](#), [15](#), [16](#), [20](#), [23](#), [24](#), [27](#),  
[30](#), [37](#), [51](#), [52](#), [54](#), [56](#), [57](#), [69](#), [79](#)
- SisterSize (Stemwardness), [64](#)
- SortTree, [4](#), [15](#), [16](#), [20](#), [23](#), [27](#), [30](#), [51](#), [52](#), [54](#),  
[56](#), [57](#), [69](#)
- SplitFrequency, [10](#), [26](#), [31](#), [44](#), [45](#), [58](#), [63](#),  
[72](#), [74](#)
- SplitFrequency(), [26](#), [70](#)
- SplitImbalance (TipsInSplits), [73](#)
- SplitInformation, [11](#), [59](#), [62](#), [80](#), [83](#)
- SplitMatchProbability, [11](#), [61](#), [61](#), [80](#), [83](#)
- SplitNumber (SplitFrequency), [58](#)
- Splits, [68](#), [81](#)
- SplitsInBinaryTree, [10](#), [16](#), [26](#), [31](#), [44](#), [45](#),  
[59](#), [62](#), [72](#), [74](#), [76](#)
- StarTree (GenerateTree), [24](#)
- Stemwardness, [14](#), [64](#), [75](#)
- StringToPhyDat, [66](#)
- StringToPhydat (StringToPhyDat), [66](#)
- Subsplit, [68](#), [82](#)
- Subtree, [4](#), [15](#), [16](#), [20](#), [23](#), [27](#), [30](#), [51](#), [52](#), [54](#),  
[56](#), [57](#), [69](#)
- SupportColor (SupportColour), [70](#)
- SupportColour, [70](#)
- SupportColour(), [26](#)
  
- TCIContext (TotalCopheneticIndex), [74](#)
- TipLabels, [10](#), [16](#), [26](#), [31](#), [44](#), [45](#), [59](#), [63](#), [71](#),  
[74](#), [76](#)
- TipLabels(), [24](#), [77](#)
- TipsInSplits, [10](#), [26](#), [31](#), [44](#), [45](#), [59](#), [63](#), [72](#),  
[73](#)
- TNTText2Tree (ReadTntTree), [48](#)
- TntText2Tree (ReadTntTree), [48](#)
- TotalCopheneticIndex, [14](#), [65](#), [74](#)
- TreeIsRooted, [16](#), [44](#), [45](#), [63](#), [72](#), [76](#)
- TreeIsRooted(), [53](#)
- TreeNumber, [24](#), [37](#), [56](#), [77](#)
- TreeShape(), [79](#)
- TreesMatchingSplit, [11](#), [61](#), [62](#), [79](#), [83](#)
- TreesMatchingTree, [14](#), [42](#), [80](#)
- TreeSplits (SplitFrequency), [58](#)
- TrivialSplits, [68](#), [81](#)
  
- UnrootedTreesMatchingSplit, [11](#), [61](#), [62](#),  
[80](#), [82](#)
- UnrootTree (RootTree), [54](#)
- UnshiftTree, [83](#)
  
- vapply64 (sapply64), [55](#)
  
- WithoutTrivialSplits (TrivialSplits), [81](#)
- write.tree(), [36](#)
- WriteTntCharacters, [84](#)
- WriteTntCharacters(), [48](#)