node API for Tree version 0.4.0

These methods are implemented by `Tree` version `0.4.0` for its nodes

cd

change value of the Tree's `currentPath`

syntax:

```
cd [ path ]
```

fastWalk

work way through this node's directory structure. sending a particular message to each child node in turn

syntax:

```
fastWalk [ depthFirst ] message
```

if `depthFirst` is given as a keyword for the first argument, then the message is applied to the leaves before being applied to this node.

otherwise, apply message to this node before descending down to the leaves.

NOTE THIS IS A NON-OBJECT-ORIENTED METHOD, optimised for speed.

isNode

determine if a path represents a valid node from this node.

syntax:

```
isNode path
```

`path` is either relative or absolute

if relative, is relative to this node. **NOTE** how this is different from the Tree's concept of a relative path.

returns:

the absolute path if it exists, null string otherwise.

ls

list all nodes that are children of this one

syntax:

```
ls
```

myPath

return the path for this directory

syntax:

```
myPath
```

node

reference a node in a Tree from another node

syntax:

```
node path messsage
```

path is either relative or absolute

if relative, is relative to this node. **NOTE** how this differs from the Tree's concept of relative path.

parseMessage

For a dictionary, parses a message and converts it to a canonical form of *method arguments*

specifically

```
key --> getKey key
key: value --> setKey key value
path message --> node path message
```

syntax:

```
parseMessage message
```

returns a message in the form

$ *method arguments*

NOTE This uses a deprecated methodology which is no longer consistent with other Offsiders. Should just override `sugar` instead.

removeAll

remove this node, and all children

syntax:

```
removeAll
```

tree

reference the tree directory that this node is contained within

syntax:

```
tree [ message ]
```

walk

work way through this node's directory structure. sending a particular message to each child node in turn

syntax:

```
walk [ depthFirst ] message
```

is `depthFirst` is given as a keyword for the first argument, then the message is applied to the leaves before being applied to this node.

otherwise, apply message to this node before descending down to the leaves.