```
                                                                   |
                                                                   |
                 (\                                                |
        . ` ' {(((-](8                                             |
       '         (/                                                |
       `.                                                          |
         .                         ANAGALLIS                       |
          `.                                                       |
            .           A PROGRAM FOR PARSIMONY ANALYSIS           |
           ,                OF CHARACTER HIERARCHIES           '   |   '
          .                                                     \ ^ /
         '                                                   `---[x]---'
         ` .                            _._                   ,'/°-°\`.
             ` .                     _ /   \ _               / |     | \
                ,                  .'  \   /  `.
                 .        , ' ` . \ _.[o]._ /
                  ` . .              /   |   \
                                     \__/ \__/
                                     '       '
```

v0.998 beta 16 April 2018
provided as is and without warranty of any kind
Jan De Laet, Gothenburg Botanic Garden

This documentation dump was generated with command
'help dump of anagallis_v0.998_beta.doc w80'

It is formatted to view with a non-proportional font
and 80 characters per line

```
                                        __
                                (\  ,--'  `--.
                                 '><    - ;°-<
                                (/  `--.__,.-'
```

```
USAGE anagallis [--help][-c][-h][-s][-v] [commands]
=====
 --help    Show usage information and program options, and exit
 -c        Start the program with context-sensitive enabled (as if command
           'program set context =' ('psc =') is executed)
 -h        Show usage information and program options, and exit
 -s        Suppress display of program header at the start of an interactive
           session
 -v        Show program version and exit
 commands  A command sequence as it would be entered from the program prompt
           (use ';' as command separator). To pass the commands properly,
           characters with special meaning in the shell from which the program
           is called (such as ';' or '*' in most shells) must be escaped. It
           mostly suffices to put the sequence between single quotes.
```

After the command sequence is executed, the program prompt is displayed. To execute commands in file 'fname', put command 'script execute start fname' ('sestafname') in the command sequence. Full batch mode can be achieve by including command 'program quit' ('pq') in the command sequence or in the command file.

A list of known bugs and issues is maintained at www.anagallis.be/anagallis

Use command 'help summary' ('hs') for an overview of commands and help topics ('hsc' for just commands, 'hst' for just topics), 'help dump f fname' ('hd f fname') for a dump of all documentation to file 'fname', 'help command - help' ('hc-h' or '?h') for more information about the help command.

Command 'program set context =' ('psc=') enables context sensitive help.

Some help commands to get more information about how to work with character hierarchies:

```
   * read character data                                         '? crn'
   * define character hierarchies                                '? cps'
   * search for optimal trees                                    '? tseam'
   * list summary scores of character hierarchies                '? csc'
   * plot character hierarchy optimizations, with score details  '? cdp'
```

OVERVIEW OF HELP TOPICS AND COMMANDS

```
====================================

> name, shortest unambiguous abbreviation, short description

Help topics
-----------
  theory                th    Theory and background
  general               g     General information
    command_structure   gc    Command structure
    tree_buffer         gt    Tree buffer
  input                 i     Input
    native              inn   anagallis data formats
    import              ii    Importing other data formats
  tntmode               tn    TNT mode
  references            r     References

Regular commands
----------------
  "                   "       echo the input until an end-of-command character
  #                   #       skip input until the end of the input line
  >                   >       provide context-sensitive help (only available
                              from the command prompt under 'program set context
                              =' or 'psc=')
  ?                   ?       shorthand for 'help command -': use '?xyz' for
                              basic help about command 'xyz'
  characters          c       current number of characters and terminals; has
                              optional subcommands to do stuff with characters
    diagnose          cd      show character state optimizations on tree nodes;
                              requires a subcommand
      plot            cdp     show tree plots of character state optimizations
      tabulate        cdt     show tables of character state optimizations
    properties        cp      set/show basic character properties or show
```

```
                            derived character properties; requires a
                            subcommand
    minmax          cpm     list minimum and maximum number of steps for all
                            characters outside character hierarchies
    set             cps     set/show basic character properties (defaults:
                            prior weight 1, active, non-additive, not part of
                            a character hierarchy)
  read              cr      read character data; requires a subcommand (use
                            command 'import' to import TNT or nexus character
                            data)
    alphanumeric    cra     read character data with up to 30 regular
                            character states coded as 0-9 and a-t (or A-T)
    numeric         crn     read character data with up to ten regular
                            character states coded as 0-9
  score             csc     a summary of the scores of all characters and
                            character hierarchies on one or more trees
  show              csh     show the current dataset
help                h       show basic usage information and program options;
                            has optional subcommands to get more detailed
                            information
  command           hc      show information about a specific regular command
  dump              hd      show all built-in program documentation
  summary           hs      overview of available commands and/or help topics
  topic             ht      get information about a specific help topic
import              i       import data from other file formats; requires a
                            subcommand
  nexus             in      execute a nexus datafile (supported nexus subset
                            still empty in this version...)
  tnt               it      execute a TNT datafile (limited support for a tiny
                            subset of TNT commands)
log                 l       name and status of log file, if there is one; has
                            optional subcommands
  pauze             lp      suspend output to the logfile
  resume            lr      resume output to the logfile
  start             lsta    open a log file (append mode by default)
  stop              lsto    close the current logfile
optimality          o       set/show optimality criterion; requires a
                            subcommand
  set               os      overview of settings that relate to the optimality
                            criterion; has optional subcommands
    deviation       osd     set/show allowed deviation from optimality in tree
                            searches and tree buffer cleaning
    searchmode      oss     set/show search mode: look for best (=, default)
                            or worst (-) trees
program             p       quit the program or set/show general settings;
                            requires a subcommand
  quit              pq      quit the program
  set               ps      overview of general settings; has optional
                            subcommands
    context         psc     set/show if context-sensitive help (command '>')
                            is available from the command line (=) or not (-,
                            default)
    longlists       psl     set/show if lists of command completions are
                            multilevel (=) or not (-, default)
    randomseed      psr     set/show seed for generator of pseudorandom
                            numbers
    shortcommands   pss     set/show if command abbreviations are allowed (=,
                            default) or not (-)
    tntmode         pst     set/show if TNT mode is on
    unicode         psu     set/show if tree plotting can use multibyte UTF-8
```

```
                              characters (=, default) or not (-)
  script              s       overview of open script files; has optional
                              subcommands
    execute           se      overview of script files that are opened for
                              execution; has optional subcommands
      pauze           sep     temporarily suspend execution of current script
                              file and get interactive input
      resume          ser     resume reading from the current script file that
                              is open for execution
      start           sesta   open a script file and start executing its
                              commands
      stop            sesto   close the current script file that is open for
                              execution
    record            sr      name and status of the file that is open for
                              recording commands, if there is one; has optional
                              subcommands
      pauze           srp     temporarily suspend writing to the current file
                              for recording commands
      resume          srr     resume recording commands to the script file for
                              recording
      start           srsta   open a file for recording commands (append mode by
                              default)
      stop            srsto   close the current file for recording commands
  trees               t       current number of trees in memory; has optional
                              subcommands to do stuff with trees
    consense          tc      calculate consensus trees; requires a subcommand
      majority        tcm     majority rule consensus tree
      strict          tcs     strict consensus tree
    read              trr     read trees in parenthetical notation (use command
                              'import' to import TNT or nexus trees)
    score             tsc     list the score of one or more trees
    search            tsea    search trees; requires a subcommand
      mult            tseam   do one or more replicates of building a tree and
                              swapping it (spr or tbr)
      swap            tseas   swap trees from the tree buffer (spr or tbr)
    select            tsel    manipulate trees in the tree buffer; requires a
                              subcommand
      best            tselb   discard suboptimal trees
      delete          tseld   discard the trees in the specified tree scopes
      keep            tselk   discard the trees that are outside the specified
                              tree scopes
      unique          tselu   discard duplicate trees
    set               tset    overview of tree related settings; has optional
                              subcommands
      current         tsetc   set/show the default tree that is for example used
                              when showing trees or character optimizations on
                              trees
      outgroup        tseto   set/show terminal(s) to be used as outgroup(s)
                              when showing trees
      width           tsetw   set/show default maximum width of a line when
                              plotting trees
      zerocollapse    tsetz   set/show the rule for collapsing zero-length
                              branches
    show              tsh     show trees; requires a subcommand
      plot            tshp    plot trees using character graphics
      write           tshw    write trees in parenthetical notation

TNT mode commands
-----------------
> only available in TNT mode (command 'program set tntmode =' or 'pst=') and in
```

```
imported TNT files (command 'import tnt' or 'it')
  ccode         c     set/show character settings
  help          h     show documentation for the commands that are available in
                      TNT mode and in imported TNT files
  nstates       n     set the TNT default datatype
  program       p
    set         ps
      tntmode   pst   set/show if TNT mode is on
  quit          q     leave TNT mode, go back to regular mode
  tread         t     read trees in parenthetical notation (numbering of
                      terminals starts from 0)
  xread         x     read alphanumeric or dna data (no support for interleaved
                      data)
```

Use 'help topic - topicname' ('ht-topicname') for more information about topic 'topicname' (topicname can be abbreviated).
Use 'help command - commandname' ('hc-commandname') for more information about regular command 'commandname' (commandname can be abbreviated).

HELP TOPICS

1 Theory and background

In parsimony analysis, the problems with missing characters or inapplicables (Maddison 1993) can be overcome by maximizing the amount of similarity that can be interpreted as homology (De Laet 2005, 2015). This holds for inapplicables as they arise in the analysis of unaligned sequence data and for inapplicables as they arise in the classic setting. With inapplicables in the analysis of unaligned sequence data, the computational complexity of the optimization of homology on a given tree makes the use of heuristic approximations unavoidable. With inapplicables as they arise in morphology, however, that computational complexity is greatly reduced and exact algorithms for the optimal homology score of a character hierachy with inapplicables becomes practiclly feasible (well...). Anagallis is a program that provides tree searches with such algorithms.

The general theoretical framework also applies to inapplicables in the classic setting (De Laet 2005:110). One way to present it is in terms of homoplasy: instances of observed similarity, as coded in datasets, that cannot be explained by inheritance from a common ancestor (Farris 1983, 2008). Identifying the tree(s) that minimize (independent) homoplasy then amounts to identifying the trees that maximize the amount of similarity that can be explained by common ancestry - homology. As such these trees, as hypotheses of genaeology, are hypotheses of maximum explanatory power. When inapplicables are present, minimal homoplasy or maximum homology is obtained when the sum of losses and gains an absence/presence characters, transformations in other characters, and subcharacters is minimized (De Laet 2005, 2015).

Consider a group of terminals in which each terminal has a tail that is either red or green. These two features - tails that are red and tails that are green - can be coded as a character 'tail color', with states 'green' and red'. This coding entails the assumptions that
* observed similarities in tail color can come about by inheritance and common ancestry;
* red tails can evolve into green tails and the other way around.
The coding itself implies the that such a transformation happened at least once.

On trees where this character requires one step, all coded similarities can be explained by inhertitance and common ancestry and there is no homoplasy. On trees where this character requires more steps, the amount of homoplasy or unexplained similarity is equal to the extra number of steps beyond the first one (hence the expression that homoplasy amounts to extra steps). As an example, if the character requires two steps on a tree, there is either exactly one independent similarity in tail redness or exactly one independent similarity in tail greenness that the tree cannot explain by inheritance and common descent.

Consider inext a group of terminals in which each terminal either has no tail, or has a tail that is either red or green. Absence and presence of a tail can be coded in a binary character that codes the distribution of tail absence and presence. The above character 'tail color' can be retained but now there are some terminals for which it is not applicable. The first underlying assumption of this character is still the same: observed similarities in tail color can come about by inheritance common ancestry. The second one still holds, strictly speaking, but it is not longer complete: redness of a tail cannot only originate as a transformation from greenness (and the other way around), it can also just happen to be color of the tail in the ancestor that acquired a tail. The character 'tail color' now implies the assumption that at least two such events - tail transformation or tail gain (including plesiomorphic presence) - happened.

On trees where this character requires exactly two such events, all coded similarities in tail color can be explained by inheritance and common descent and there is no homoplasy in tail color. Here are some of the possiblities (several more exist):

* one gain of a tail (either red or green) and a subsequent transformation of tail color;
* symplesiomorphic presence of a red tail that got green in one part of the tree and got lost in another part.
* two gains of a tail (one red, the other green);
* a gain of a green tail followed by loss of a tail followed by gain of a red tail;

Note that the last two examples do not entail homoplasy in tail color (all observed shared tail colors can be explained by inheritance and common descent), but they do have an unexplained similarity in the absence or presence of a tail. In the first two examples, all observed similarities in absence and presence of a tail can be fully explained by common descent within this group of terminals as well.

On trees where the character 'tail color' requires more than two such events, not all observed similarities in tail color can be explained by inheritance and common descent. As an example, if there is a gain of a green tail followed by a transformation to a red tail, and elsewhere on the tree an independent gain of a red tail, there is one independent observed similarity in redness that the tree cannot explain by inheritance and common descent.

So homoplasy or unexplained similarity in tail color cannot only come about by extra transformations between the two colors, but also by extra gains or losses of tails. Interestinly, it is not the total number of tail gains and losses that determines the effect on color homoplasy, but the resulting number of different regions on the tree in which tails are present. This is the number of subcharacters (De Laet 2005) of this character. So the number of unexplaned similarities in tail color on a tree equals its number of color transformations plus its number of subcharacters minus two, the minimum number of steps required on any tree.

Trees in which tail color has no homoplasy may nevertheless have homoplasy in the presence or absence of a tail as such. So to minimize homoplasy in all features pertaining to the tail, the homoplasy in both characters has to be minimized simultaneously. But when the two characters are optimized independently, contradictions may arise: independent minimization of homoplasy in tail color may for example imply tail presence at inner nodes that lack a tail when homoplasy in the absence/presence character is minimized.

To maximize homology accross a full-blown absence/presence character hierarchy, it suffices to minimize the sum of gains/losses, transformations, and regions of applicability (subcharacters) over the constituent characters of that character hierarchy, subject to the constraint that the overall explanation must be free of internal contradictions (De Laet 2005, 2015).Anagallis is a computer program that provides tree evaluation and tree searches with such algorithms.

2 General information

```
--> Subtopics
      general command_structure  gc  Command structure
      general tree_buffer        gt  Tree buffer
```

General introduction.

2.1 Command structure

Commands are not case-sensitive, options are. Options are parsed from left to right.

Commands are structured in a hierarchy. Take command 'trees'. By itself, it just gives the number of trees in the tree buffer. In addition it has a number of subordinate commands to do all kinds of things with these trees. Command 'trees select', for example, can be used to manipulate trees in the tree buffer. It does nothing by itself, but requires a further subcommand.Examples are 'trees select best' to select all optimal trees, or 'trees select unique' to remove duplicate trees from the tree buffer.
Likewise, 'trees search', the command to search for trees, requires a further subcommand (trees search mult or trees search swap).
Command 'trees set', as a last example, can be used to set a number of properties related to trees, or to show the current values of those settings. By itself, it lists all tree-related settings. With additional subcommands, those different settings can either be set or displayed individulally. 'Trees set collapse', for example sets or displays the way in which zero-length branches are treated. Or 'trees set width' sets the maximum width (in characters) that is used when trees are displayed using character graphics (trees that are wider are chopped into fitting pieces).

'help summary' (hs) gives an overview of the full command hierarchy.For multilevel commands, unambiguous abbreviations for the different levels do not need white space in between the different levels. Ambiguous abbreviations can be disambigued by lengthening the abbreviation or by inserting white space (or in some cases even by shortening the abbreviation). 'Program quit', for example, can also be entered as 'p q' or as 'pq'.This leads to some subtleties when it comes to determining the shortest unambiguous abbreviation for a multi-level command.
The numeric code for the shortest unambiguous abbreviations of, say, a two-level command is of the following form: 'i[.+]j': an integer i followed by one of the characters ' ', '.', or '+', followed by an integer j.
The first integer indicates the number of characters that are required for the top-level command, the second for the subcommand. When the character in between

is ' ', then a shortest unambiguous abbreviation exists that has a space in between the abbreviations of the two levels (but there might be an alternative equally short unambiguous abbreviation without a space and with an additional character from the top-level command). When the character is a '+', the shortest unambiguous abreviation of the full command consists of the direct juxtaposition of the indicated abbreviations for the top-level command and the subcommand (without a blank in between); in addition, no ambiguities will arise if longer chuncks are used. When the character is a '.', the shortest unambiguous abreviation of the full command still consists of the direct juxtaposition of the indicated abbreviations for the top-level command and the subcommand, but ambiguities will arise for some larger chunks of the top-level command.

It can happen that there are multiple shortest abbreviations. In those cases, and when not using numeric codes, the abbreviation shown is always the one that does not contain spaces.

No blanks are required to separate commands from their options or to separate different options.

Options and commands are case-insensitive, filenames are case-sensitive or not, depending on the underlying operating system.
Filenames are not allowed to contain blanks, irrespective of the underlying operating system. White space is used to separate a filename argument from subsequent options.
Likewise, names of terminals are not allowed to contain blanks (and white space is used to separate a terminal name from character data when reading data matrices).Names of terminals are case_sensitive.

2.2 Tree buffer

The tree buffer always reflects the current character settings (command 'characters properties set') and the current setting of collapsing mode for zero-length branches (command 'trees set zerocollapse') (using a lazy re-evaluation approach). So by changing any of these, a tree buffer with no duplicate trees may end up with duplicate trees. Or a tree buffer in which all trees have the same score may end up with trees of different scores.

3 Input

```
--> Subtopics
      input native  inn  anagallis data formats
      input import  ii   Importing other data formats
```

3.1 Anagallis data formats

To read character matrices: see command 'characters read'. To set character properties (weights, additive/non-additive, ...) and to define character hierarchies: see command 'characters properties set'. To read trees: see command trees read.

3.2 Importing other data formats

See command 'import' and its subcommands 'import tnt' and 'import nexus' (but nexus import is not supported yet). More information about support for TNT can be found in section 'tntmode'.Importing other data formats: currently only TNT data files.

4 TNT mode

In addition to its native mode, anagallis has a TNT mode, a mode that partially supports a tiny subset of TNT commands (Goloboff et al. 2008). Use command 'program set tntmode' ('pst') to toggle between both modes. The command prompt indicates which mode is on: in TNT mode it has a 't'. TNT mode is also implicitly entered when importing a TNT file with command 'import tnt' ('it').

The idea of TNT mode is to be able to read files with data matrices in TNT format. Commands that are currently (partially) supported are 'ccode', 'help', 'nstates', 'xread', and 'quit' (the last one as a synomym for 'program set tntmode -' or 'pst-'. An overview of the degree to which they are supported is obtained by entering 'help*' in TNT mode.

Shortest unambiguous abbreviations are determined on the basis of this set of supported commands. So, differently as in TNT, 'q' means 'quit', not 'qcollapse'.

Commands that are not supported (or plain wrong) are skipped with appropriate warnings (they are flagged as invalid commands) but don't trigger errors. Unsupported options of partially supported commands (as available in TNT version 1.1) are flagged as unsupported options but don't trigger errors either. So in both cases anagallis keeps reading a TNT datafile rather than closing it with an error message.

Data sets, character codes, and trees in the tree buffer persist accross mode changes. So you can read a matrix in regular mode, change its character settings in TNT mode, and finally go back to analyze the data regular mode. Just switch between counting from one and counting from zero when doing so.

Changes of the default datatype for TNT (TNT command 'nstates') do not persist accross different TNT files or different TNT mode sessions.

5 References

Brazeau, M. D., Guillerme T., Smith, M.R. 2017. Morphological phylogenetic analysis with inapplicable data. BioRxiv preprint first posted online October 26, 2017. doi: http://dx.doi.org/10.1101/209775.

Coddington and Scharff 1994. Problems with zero-length branches. Cladistics 10: 415-423.

De Laet, J. 2005. Parsimony and the problem of inapplicables in sequence data. Pp. 81-116 in Albert, V.A. (ed.) Parsimony, phylogeny and genomics. Oxford University Press, ISBN 0-19-856493-7.

De Laet, J. 2015. Parsimony analysis of unaligned sequence data: maximization of homology and minimization of homoplasy, not minimization of operationally defined total cost or minimization of equally weighted transformations. Cladistics.

De Laet, J. 2017. A note on Brazeau et al.'s (2017) algorithm for characters with inapplicable data, illustrated with an analysis of their Fig. 3d using anagallis, a program for parsimony analysis of character hierarchies. Technical Report, November 5, 2017. DOI: 10.13140/RG.2.2.31309.54245

Farris, J.S. 1983. The logical basis of hylogenetic analysis. In Advances in Cladistics Vol. 2 (eds. N.I. Platnick, and V.A. Funk), pp. 7-36. Columbia University Press, New York, New York.

Farris, J.S. 1989. The retention index and the rescaled consistency index. Cladistics 6: 91-100.

Farris, J.S. 2008. Parsimony and explanatory power. Cladistics 24: 825-847.

Goloboff, P.A, Nixon, K.C, and J.S. Farris 2008. TNT, a free program for phylogenetic analysis. Cladistics 24: 774-786.

Maddison, W.P. 2003. Missing data versus missing characters in phylogenetic analysis. Syst. Biol. 42: 576-581.

REGULAR COMMANDS

Command '"'

> Echo the input until an end-of-command character.

Command double-quote is useful to document script files with information that gets echoed to the the output of the script, or to insert comments in open log files. As currently implemented, the command ends when it encounters an end-of-line or a semi-colon. So there is no way to include a semicolon in the comment itself.

For comments that do not get echoed to the output, use command '#'.

Command '#'

> Skip input until the end of the input line.

Useful to insert comments in script files or to comment out commands in script files.

To insert comments that are echoed to the output of the script, use command '"' (double quote).

Command '>'

> Provide context-sensitive help (only available from the command prompt under 'program set context =' or 'psc=').

When context-sensitive help is enabled, the current location in the command hierarchy is indicated before the command prompt.

Experimental feature. It seems to be working quite well but remains to be thouroughly tested.

Command '?' <commandname>

> Shorthand for 'help command -': use '?xyz' for basic help about command 'xyz'.

```
> Argument
    commandname:   the command name or program option for which help is
                   requested (required)

For the basic documentation of command or program option 'xyz', enter '? xyz'
('?xyz'). Use 'help summary c' ('hsc') for a list of command names and their
shortest abbreviations.

--------------------------------------------------------------------------------
Command 'characters' (c)   {cd, cp, cr, csc, csh}

> Current number of characters and terminals; has optional subcommands to do
  stuff with characters.
> Subcommands
    characters diagnose    cd   show character state optimizations on tree
                                nodes; requires a subcommand
    characters properties  cp   set/show basic character properties or show
                                derived character properties; requires a
                                subcommand
    characters read        cr   read character data; requires a subcommand (use
                                command 'import' to import TNT or nexus
                                character data)
    characters score       csc  a summary of the scores of all characters and
                                character hierarchies on one or more trees
    characters show        csh  show the current dataset

--------------------------------------------------------------------------------
Command 'characters diagnose' (cd)   {cdp, cdt}

> Show character state optimizations on tree nodes; requires a subcommand.
> Subcommands
    characters diagnose plot      cdp  show tree plots of character state
                                       optimizations
    characters diagnose tabulate  cdt  show tables of character state
                                       optimizations

--------------------------------------------------------------------------------
Command 'characters diagnose plot' (cdp)  [abcCdDfgijJkKnorstTuvW] <scopes>

> Show tree plots of character state optimizations.
> Options:
    a          label the terminals with their names (this is the default; it is
               overwritten when option 'n' is present; this option is useful to
               have terminal names in such cases as well)
    b          suppress numbering of internal nodes
    c          interpret scopes that follow as characters to include; cannot be
               combined with option 'C'
    C          interpret scopes that follow as characters to exclude; cannot be
               combined with option 'c'
    d          dry run to set custom defaults: remember all other options in
               this invocation for use with the following invocations in this
               session (with no other options, the built-in defaults are
               restored)
    D          dry run to show the current custom defaults
    f fname    write output also to file fname (append mode by default)
    i          when plotting subtrees that branch at the same level, plot small
               subtrees last, and equally sized non-leaf subtrees sorted
               according to their decreasing numeric code; this option also
               inverses the plot order of leaves that branch at the same level
    j          indent increment for each next level in a character hierarchy
```

	(default 3; minimum 1, maximum 20)
J	add a blank line between each plot header and the plot itself
g n	use alternative ASCII glyph set 1 or 2 for plotting trees (has only effect under 'program set unicode -' or 'psu-')
k	condensed output (shorter branches)
K n	truncate terminal names (to a minimum of n characters, n > 0) when they would exceed the specified width (option 'W') for plotting
n	label the terminals with their numeric code (their sequential number in the data matrix; see option 'a' for more information)
r charnum	include all characters from character hierarchy wth main root character charnum
o	modifies option 'f': use overwite mode, not append mode
s	sort terminals that branch at same level according to their increasing numeric code (default: ascending alphabetical order of terminal names)
t	interpret scopes that follow as trees to include; cannot be combined with option 'T'; this is the default interpretation of scopes
T	interpret scopes that follow as trees to exclude; cannot be combined with option 't' or with default scopes
u	unravel statesets of optimizations with different subcharacters (results in one tree per character); only (possibly) affects characters in character hierarchies
v	verbose (only has effect with character hierarchies)
W n	maximum width (in characters) of a single line (beyond this, the tree is broken into subtrees); use -1 for the width of the current window (default), 0 to turn off this feature (valid values 20 - 5000)

```
> Argument
    scopes:   one or more scopes (character scopes by default, no default scope)
```

Shows the final state sets of one or more characters on one or more trees. There is no default for the character(s) to show. The default tree is the current tree.

Scopes and options may be intermingled. By default, scopes are interpreted as scopes of characters to include. This default interpretation can be changed with option 'C' (interpret scopes that follow as characters to exclude), 't' (interpret scopes that follow as trees to include) and 'T' (interpret scopes that follow as trees to exclude). Options 'c' and 'C' cannot be combinedi. Neither can 't' and 'T'.

Polytomies are not optimized as polytomies. Optimizations shown are the ones for the underlying dichotomous resolutions that the program happened to find first.

When a character that is part of a character hierarchy is specified as a character to plot, all characters that lead from the main root character of the hierarchy to that character are automatically included as well. As an example, consider character hierarchy <1 5 <7 8 9 <10 11 15>>>. Requesting an optimization plot for character 15 will automatically trigger plots for characters 1, 7, and 10.

Character optimizations are not plotted in the order in which the characters are entered in this command, but, with the exception of characters that are part of a hierarchy, in numeric character order. Characters that are part of a character hierarchy are plotted as part of that hierarchy under the main root character of that hierarchy. With option 'v', a reference to that main root character is added at the numeric position of each character of the hierarchy for which a plot was requested.

To plot optimizations for a full character hierarchy, all characters of that hierarchy must be specified in the invocation of this command. Alternatively, option 'r charnum' can be used, with charnum the number of the main root character of that hierarchy. With this option, all characters of that hierarchy are automatically included. It does not change the interpretation of scopes as set with options 'c', 'C', 't' and 'T.

For each character that is part of a character hierarchy, there is by default one plot that displays, at each node, the union of all states that are part of at least one optimal solution for the hierarchy as a whole. These are called the aggregate final or optimal statesets for that character.

With the option 'u' (for un-aggregate), this single plot may be decomposed into multiple plots that make it easier to reconstruct individual optimal solutions for the hierarchy as a whole. The statessets that are shown with this option are called the non-aggregated statesets. One advantage of this option is that it allows to output the number of subcharacters and gains/losses or transformations for each individual plot. As such it provides a refinement of scores provided witch commands 'trees score' and 'characters score': summing subcharacters, gains/losses and transformations though any full path through a character hierarchy ias shown with option 'u' of this command will yield the total score for that hierarchy. This works quite well for simpler hierarchies but remains experimental for two reasons:

First, while (programming bugs apart) it is guaranteed that all possible optimizations will be shown, it is currently not guaranteed that an optimization will be shown only once. In other words, with option 'u', some optimizations may be shown multiple times.

Second, for some character hierarchies on some trees, it can happen that there are distinct regions in a tree that can be optimized separately and that the program has optimized separately. With option 'u', the optimization for any such region is then shown in a full treeplot with '[+]' as reconstructed stateset for the other regions. The '+' serves as a placeholder for the statesets that are plotted in the plots for those other regions. This is a compact way of representing such cases, but it complicates the presentation of the results for non-trivial nested hierarchies, an issue that is not solved in this version. So, with option 'u', the program may for some nested characters in some character hierarchies indicate that it is unable to plot the final statesets of the optimization. This does not mean that they have not been correctly calculated or that the reported scores are not correct. It is just a complex case of presentation of hierarchic results that the program can't handle yet. That said, the aggregated statesets can still be plotted for such cases.

```
--------------------------------------------------------------------------------
Command 'characters diagnose tabulate' (cdt)  [cCtT] <scope(s)>

> Show tables of character state optimizations.
> Options:
    c  interpret scopes that follow as characters to include; cannot be combined
       with option 'C'
    C  interpret scopes that follow as characters to exclude; cannot be combined
       with option 'c'
    t  interpret scopes that follow as trees to include; cannot be combined with
       option 'T'; this is the default interpretation of scopes
    T  interpret scopes that follow as trees to exclude; cannot be combined with
       option 't' or with default scopes
> Argument
    scopes:   one or more scopes (character scopes by default, no default
             character scope)
```

Arguments and options can be intermingled.

Tabulate the final statesets of reconstructed inner nodes for one or more characters on the current tree. Use option 't' or option 'T' to select other trees.

Numbers of inner nodes correspond to the numbers as obtained with command 'trees show plot'
Polytomies are not optimized (yet). Optimizations shown are the ones for the underlying dichotomous resolutions that the program happened to find first.

For characters in a character hierarchy, the statesets shown are the aggregate statesets.

```
--------------------------------------------------------------------------------
Command 'characters properties' (cp)   {cpm, cps}

> Set/show basic character properties or show derived character properties;
  requires a subcommand.
> Subcommands
    characters properties minmax  cpm  list minimum and maximum number of steps
                                       for all characters outside character
                                       hierarchies
    characters properties set     cps  set/show basic character properties
                                       (defaults: prior weight 1, active,
                                       non-additive, not part of a character
                                       hierarchy)

--------------------------------------------------------------------------------
Command 'characters properties minmax' (cpm)

> List minimum and maximum number of steps for all characters outside character
  hierarchies.
```

For a character outside a character hierarchy, the maximum number of steps, max (its number of steps on a star-tree; g of Farris 1989), is calculated directly and exactly. The minimum number of steps for such a character, min (m of Farris 1989), is calculated using a single tree build using only those terminals that have a different state set. This gives correct results for non-additive characters or when no polymorphic terminals are present. It remains to be determined if this always works for additive polymorphic terminals.

These numbers can be considered derived or secondary character properties: they depend on the basic character properties but also on the set of terminals being considered.

For characters in character hierarchies, no such numbers are provided. For any character hierarchy of interest, they can be obtained heuristically as follows.

For the minimum numbers, first inacticate all characters except those of that character hierarchy. Next do a tree search under 'optimality set searchmode =' ('oss=') The score obtained provides the minimal score for the character hierarchy as a whole. The distribution(s) of this score over the individual characters of the hierarchy provide(s) the (range of possible) minimum numbers for these characters.

Likewise, for the maximum numbers, first inacticate all characters except those of that character hierarchy. Next do a tree search under 'optimality set searchmode -' ('oss-') The score obtained provides the maximum score for the character hierarchy as a whole. The distribution(s) of this score over the individual characters of the hierarchy provide(s) the (range of possible) miaximum numbers for these characters.

```
Command 'characters properties set' (cps)  [/+-[]<>acefnopqv] <character scopes>

> Set/show basic character properties (defaults: prior weight 1, active,
  non-additive, not part of a character hierarchy).
> Options:
    /n       set character weight to weight n for the character scopes that
             follow
    +        set the character additive for the character scopes that follow
             (see -)
    -        set the character non-additive for the character scopes that follow
             (see +)
    [        activate the character scopes that follow (see ])
    ]         inactivate the character scopes that follow (see [)
    <>       define a character hierarchy (see ><)
    ><       undefine a character hierarchy (see <>)
    a        show the active/nonactive status of all characters
    c        show the character hierarchies that are currently defined
    e str    str must be 'tnt'; write the character codes as a readable
             statement for TNT (as far as they exist there); this implies a
             renumbering of characters to start counting from zero
    f fname  write output also to file fname (append mode by default)
    n        show the non-additive/additive status of all characters
    o        modify option 'f': use overwrite mode, not append mode
    p        show the character weights in use
    q        don't show any of the current settings (takes precedence over other
             output modifiers)
    v        increase verbosity (up to two levels)
> Argument
    scopes:   one or more character scope (no default; scopes can be
             intermingled with options)
```

This is a multiline command, so it always needs a semicolon (';') to terminate. Numbering of characters starts from 1 (but see option 'e'). There are two groups of options:
1. character modifiers such as '+' or '[' that modify the interpretation of characters in the scopes that follow
2. options such as 'a', 'e', or 'n' that set/modify the output of the command.

By default characters are active, non-additive, and with weight one; and no character hierarchies are defined. character acivity, additivity and weight can be changed with the options '[', ']', '+', '-', and '/'

Character scopes and options may be intermingled, and the modifiers that are active are applied to the scopes as they are read. As with other commands, such changes get only activated after the whole command finishes without an error (as triggered; for example, by a non-existing option). An appropriate warning is shown when nothing gets changed because of such errors.

By default, all current settings (character additivites, weights and activities, and defined character hierarchies) are displayed in a program readable form after each run of the command. Option 'q' turns this off. By default, character additivities, weights, and activities are reported in one statement and in the most condensed form . This implies that the character scopes that are used will not necessarily show all characters in order. With one occurrence of option 'v', characters are reported in order but grouped in scopes as much as possible. With two occurrences of option 'v', characters are reported individually. When character hierarchies have been defined, these are always reported separately from activity, additivity and weight (see below for the meaning of option 'v' in that case).

Options 'a', 'c', 'n', and 'p' can be used to select what will be shown in the report as described above. From the moment one of these four is specified, only the character properties so selected will be shown. When none of these is specified, the command behaves as if all four are specified.

Outside character hierarchies, the dash character ('-') is treated (and shown) as missing information ('?'). The possibility to toggle this to treating it as a separate state is not available. The interpretation of a dash in a character hierarchy is explained in detail below.

Compared to the ccode statement of programs as Hennig86, Nona or TNT, an important extension is the possibility to define and undefine character hierarchies.

To define or specify a character hierarchy, '<' and '>' can be used to set off levels of (in)applicability. Within each level, the first character that is specified is the absence/presence character that determines inapplicability further on. Throughout such a character is referred to as a root characters because it is at the root of a character (sub)hierarchy. The root character of the complete hierarchy (the outer or upper level) is called the main root character of the hierarchy. Within each level, the characters that follow the root character are called subordinate characters at that level. This includes root characters of one level down: a root character of a nested level is a subordinate character one level up. These are called complex subordinate characters (as opposed to simple subordinate characters).

As an example,

```
    <1 5 6 <7 8>>
```

is a character hierarchy with main root character 1. This character codes absence/presence of some feature. Characters 5, 6, and 7 are subordinate characters at that level: they code three aspects of that feature where it is present. Characters 5 and 6 are simple subordinate characters, character 7 is a complex subordinate character: it is at the root of its own subhierarchy (it codes absence/presence of a subfeature). Character 8, finally, is subordinate at that subordinate level and describes some feature of that subfeature.

Root characters can have at most four different state codes:
* a first one denoting absence
* a second one denoting presence
* a third one for inapplicability
* '?' for missing data.

Polymorphisms are not allowed in root characters. The main root character of a hierarchy, in addition, is not allowed to have inapplicable data. If this would be the case in an existing dataset, that datset must first manually be augmented with one or more additional outer levels of applicability. The outer additionl level can then be used as the main root of the hierarchy.

By default, state code 0 is taken to indicate absence and state code - is taken to indicate inapplicability. Presence is coded by whatever third state code used (apart from '?' (but there are some constraints; see below). To let a different state code denote absence, specify this after the character using a '!' followed by that other state. To let a different state code denote inaplicability, specify this after the character using a ':' followed by that other state. So in root character 7 of hierarchy

```
    <1 5 6 <7 !1:0 8>>
```

state code 1 codes absence and state code 0 inapplicability; in character 1 it is still default state code 0 that codes absence.

In simple subordinate characters in a character hierarchy (characters 5, 6 and 8 in the example), state code '-' is by default taken to indicate inapplicability. To let a different state code denote inapplicability, specify this after the character using a ':' followed by that other state. So in

```
    <1 5 :2 6 <7 8 :0>>
```

state code 2 denotes inapplicability in character 5 and state code 0 denotes inapplicability in character 8; in character 6 it is still default state code - that codes inapplicability.

Polymorphisms that involve inapplicability are not allowed.

In verbose mode (option 'v'), all root characters in the hierarchy will be shown with their ':' and '!' modifiers. and all simple subordinate characters with their ':'modifier. In non-verbose mode, these are only indicated where their values differ from the defaults.

Within the specification of a character hierarchy, the regular code modifiers and the options that modify output are not allowed, but the modifiers that are active are applied while reading the specification.

Root character that don't have subordinate characters are allowed (as long as their observed states can be interpreted as an absence/presence character), but such trivial hierarchies don't serve any meaningful purpose. From a practical point of view, they will slow down tree searches. Also note that the unweighted score of such a a trivial character hierarchy will be one higher than for that same character outside that trivial character hierarchy. This is so because subcharacters are only taken into account inside character hierarchies.

Character scopes that span multiple characters are not allowed at the start of a (sub)hierarchy: root characters must be explicitly specified as a single character. Character scopes that span multiple characters are allowed for simple subordinate characters. When such a scope is followed by ':' and a state code, that state code is taken to denote inapplicability in all characters of the scope.

A character hierarchy imposes a number of logical constraints on the character state distributions:

1. When a given terminal has missing data ('?') in a root character, it must have missing data in all subordinate characters as well. (If it is not known if a feature is present or not in a terminal, aspects or subfeatures are not known either).
2. When a given terminal has absence in a root character, all subordinate characters must have inapplicability for that terminal.
3. When a given terminal has inapplicability in a root character, all subordinate characters must have inapplicability as well.
4. When a given terminal has presence in a root character, none of the subordinate characters at that level can have inapplicability (but see below).

When a hierachy specification is submitted, it will only succeed when these constraints are met. Failure for one of those reasons means that there is an internal contradiction between the requested hierarchy and the state distribution of the characters involved. If that's the case, the program will point out which of these constratnts is not met for which terminal and which character.

The idea behind this way of specifying character hierarchies is to provide a flexible way to adapt the specifications to existing datasets without having to edit these datasets (too much) rather than the other way around. For the same reason, even if the default values are different, it is not enforced that the state code for inapplicability (':' modifier) and the state code for absence ('!' modifier) in root characters must be different (when a hierarchy is properly defined, the program has sufficient information to figure out the correct interpretation). However, when they are the same, the fourth constraint is skipped in subordinate root characters because it can no longer be unequivocally tested

To undefine a character hierarchy, enclose its main root character between '>' and '<'. So

```
    '>1<'
```

undefines the hierarchy of the example.

A character hierarchy can be (in)activated as a whole by (in)activating its main root character. It is currently not allowed to (in)activate part of a hierarchy. This is not an intrinsic limitation, but removing it requires some additional programming that I didn't do yet. As a work-around, the hierarchy can be undefined and then redefined with the unwanted characters excluded. These can then be inactivated outside the hierarchy.

Alternatively, the prior weights of the characters to be excluded can be set to 0. This does not require redefinition of the hierarchy but will slow down tree searches: inactive characters are not optimized during tree searches but characters that are part of a hierarchy and have weight zero are. (outside hierarchies, zero-weight characters are not optimized during tree searches). Be aware that this can have unexpected side effects. If, for example, one would give zero weight to tail absence/presence with non-zero weight for tail color, one can end up with an awful lot of tail gains and losses (that each have weight zero).

To avoid confusion or ambiguity, some additional constraints are imposed when setting character hierarchies:
state code '-' is not allowed as a regular state code;
'-' for absence in a character (!) is not allowed;
'?' is not accepted as code for inapplicability or as code for absence.

The restriction to allow '-' only to mean inapplicability in character hierarchies makes it possible to use it unequivocally as a character that separates subcharacters when showing optimizations of character hierarchies on trees with commands 'characters diagnose plot' and 'characters diagnose tabulate'). Whatever state code has been used for inapplicability when defining a hierarchy (:), these two commands will use '-' to indicate inapplicability in their output. This makes it easier to interpret the output of these commands, especially in root characters where inapplicability and absence have been coded with the same state code.

Characters that are inactive, whether part of a hierarchy or not, are not optimized during tree searches. Afterwards their optimizations are available for commands such as 'charactersDIASTEPS' or 'charactersDIAPLOT' though. The same is true for characters that have weight 0 and are not part of a character group. Neither are taken into account when collapsing zero-length branches.

Command 'characters read' (cr)   {cra, crn}

> Read character data; requires a subcommand (use command 'import' to import TNT or nexus character data).

```
> Subcommands
    characters read alphanumeric  cra  read character data with up to 30 regular
                                       character states coded as 0-9 and a-t (or
                                       A-T)
    characters read numeric       crn  read character data with up to ten
                                       regular character states coded as 0-9
```

Support for interleaved data sets and for concatenating multiple data sets is not available yet: all character data for an analysis have to be in a single non-interleaved dataset.

After a first data set has been read, it is possible to read a new data set, but it will replace the original one.

Command 'characters read alphanumeric' (cra)

> Read character data with up to 30 regular character states coded as 0-9 and a-t (or A-T).

Works mostly as NONA's xread command. Digits 0-9 and letters a-t (not case sensitive) are available for a total of up to 30 different states. In addition, '?' is available for missing data and '-' for inapplicable data.

A dash is by default interpreted and shown as missing data ('?'). It only acquires its special meaning of inapplicable data in character hierarchies (see command 'characters properties set'

As in Nona, there can be an optional comment between the command name and the number of characters between the command name, enclosed by single quotes. The parser of anagallis does not reconize escaped single quotes as such, so make sure the comment itself does not contain single quotes.

Command 'characters read numeric' (crn)

> Read character data with up to ten regular character states coded as 0-9.

Works mostly as NONA's xread command. Digits 0-9 are available for a total of up to 10 different states. In addition, '?' is available for missing data and '-' for inapplicable data.

A dash is by default interpreted and shown as missing data ('?'). It only acquires its special meaning of inapplicable data in character hierarchies (see command 'characters properties set').

As in Nona, there can be an optional comment between the command name and the number of characters between the command name, enclosed by single quotes. The parser of anagallis does not reconize escaped single quotes as such, so make sure the comment itself does not contain single quotes.

> Examples

```
     #read the datacharacters read numeric
     '
     Tail example of Maddison (1991).
     Characters 13 and 14 are tail a/p and tail color.
     Maximizing homology, there is no long distance effect of tail color between
the non-homologous tails of A-D and K-N and two trees are obtained.
     '
     14 15
     out 000000000000 0-
     A   111100000011 11
     B   111100000001 11
     C   111100000010 12
     D   111100000011 12
     E   111000000000 0-
     F   110000000000 0-
     G   100000000000 0-
     H   000010000000 0-
     I   000011000000 0-
     J   000011100000 0-
     K   000011110000 12
     L   000011111000 12
     M   000011111100 11
     N   000011111100 11
     ;
     #define the character hierarchy
     characters properties set <13 14>;
     #do a tree search
     trees search mult *5;
     #give the number of trees found
     trees
     #plot all trees
     trees show plot;
```

Command 'characters score' (csc) [tT] <treescopes>

> A summary of the scores of all characters and character hierarchies on one or more trees.
> Options:
    t interpret scopes that follow as trees to include; cannot be combined with option 'T'; this is the default interpretation of scopes
    T interpret scopes that follow as trees to exclude; cannot be combined with option 't' or with default scopes
> Argument
    treescopes: trees to list the character scores of (defaults to the current tree)

List the character scores on the trees in the specified tree scopes (default: current tree). For characters with non-unity prior weight, the score is reported as prior weight times basic score. The scores of inactive characters are included in the output of this command (in the overview of all characters they are put between square brackets).

For a character hierarchy, the total score of the hierarchy is listed for the main absence/presence character of that hierarchy. For the other characters of that hierarchy, a placeholder reference to their immediate parent

```
absence/presence character is provided. This is done because, in general,
character hierarchies have no unique distribution of their total score over
their constituent characters. So in a brief summary as provided here, it only
makes sense to give the score of the full hierarchy.

More detailed indormation about the score of character hierarchies can be
obtained with command 'characters diagnose plot' ('cdp').

--------------------------------------------------------------------------------
Command 'characters show' (csh)  [abefinoru]

> Show the current dataset.
> Options:
    a        show active characters only
    b n      set block size to n (insert a space every nth character; default is
             10)
    e str    str must be 'tnt' or 'fasta': show the data in TNT or fasta format
    f fname  write output also to file fname (append mode by default)
    i        show informative characters only
    n        no characters, just terminal names
    o        modify option 'f': use overwrite mode, not append mode
    r        show data as originally read (discard changes such as making a
             character state set continuous for an additive character in a
             polymorphic terminal)
    u        use upper case for alphabetical state codes (default: use lower
             case)

--------------------------------------------------------------------------------
Command 'help' (h)   {hc, hd, hs, ht}

> Show basic usage information and program options; has optional subcommands to
  get more detailed information.
> Subcommands
    help command  hc  show information about a specific regular command
    help dump     hd  show all built-in program documentation
    help summary  hs  overview of available commands and/or help topics
    help topic    ht  get information about a specific help topic

--------------------------------------------------------------------------------
Command 'help command' (hc)  [forsw] <- commandname>

> Show information about a specific regular command.
> Options:
    f fname    write output also to file fname (append mode by default); implies
               'w80'
    o          modify option 'f': use overwrite mode, not append mode
    r          include documentation of subcommands (no effect when 's' is
               specified as well)
    s          summary (only short description, completions, and list of options
               and arguments, no long description and no examples)
    w numcols  set line width (in characters); beyond this, lines are wrapped;
               defaults to the width of the current window
> Argument
    - commandname:   a dash followed by the command name or program option for
                    which help is requested (required)

For the documentation of command or program option 'xyz', enter 'help command -
xyz' ('hc-xyz'). The dash before the command name is required to disambigue
cases where a command starts with a leter that is also an option of this help
command. When using this command without options, command '? xyz' can be used as
```

```
a shorthand for 'help command - xyz'.

By default, the command name can be abbreviated (check command 'program set
shortcommands' to change this). Use 'help summary c' ('hsc') for a list of
command names and their shortest abbreviations.

Values for option 'w' must be in the range 66..10000. When using a w-value
higher than the width of the current window, output will not wrap correctly in
the window. With option 'f', wrapping is by default done at 80 characters (and
not according to the curent window). This default can be changed with option
'w'.

--------------------------------------------------------------------------------
Command 'help dump' (hd)  [cfilostTuw]

> Show all built-in program documentation.
> Options:
    c          dump documentation of regular commands ('c', 'i', 't', 'T', and
               'u' are additive)
    f fname    write output also to file fname (append mode by default); implies
               'w80'
    i          dump an index of all commands and topics ('c', 'i', 't', 'T', and
               u are additive) (as with command 'help summary')
    l          use multilevel lists of command and topic completions
    o          modify option 'f': use overwrite mode, not append mode
    s          summary (show only show short description, completions, and list
               of options and arguments for the commands)
    t          dump topics ('c', 'i', 't', 'T', and 'u' are additive)
    T          dump documentation of TNT mode commands ('c', 'i', 't', 'T', and
               'u' are additive)
    u          dump program usage and options ('c', 'i', 't', 'T', and 'u' are
               additive)
    w numcols  set line width (in characters); beyond this, lines are wrapped;
               defaults to the width of the current window

By default, all built-in documentation is dumped. When at least one of options
'c', 'i', 't', 'T' is present, output follows absence/presence of these three
options.

Values for the option 'w' must be in the range 66..10000. When using a w-value
higher than the width of the current window, output will not wrap correctly in
the window. With option 'f', wrapping is by default done at 80 characters (and
not according to the curent window). This default can be changed with option
'w'.

--------------------------------------------------------------------------------
Command 'help summary' (hs)  [bcCfilnostTvw]

> Overview of available commands and/or help topics.
> Options:
    b          brief: do not include the short descriptions
    c          show regular commands ('c', 't' and 'T' are additive)
    C c        use character 'c' as field delimiter (useful for creating csv
               output)
    f fname    write output also to file fname (append mode by default); implies
               'w80'
    i num      number of characters to indent for each next level in the command
               or topic hierarchy (2 by default, 100 at most; at least 0 with
               option 'v', at least 1 otherwise; numbers outside this range are
               set to the appropriate extreme)
    l num      number of levels to show (0 stands for all levels; this is the
               default when the option is not present)
    n          use numeric codes for shortest unambiguous abbreviations
    o          modify option 'f': use overwrite mode, not append mode
    s num      change the space between output fields (num is 2 by default, 1 at
               least, and 100 at most; numbers outside this range are set to the
               appropriate extreme)
    t          show topics ('c', 't' and 'T' are additive)
    T          show TNT mode commands ('c', 't', and 'T' are additive)
    v          verbose (spell out full command name at each level)
    w numcols  set line width (in characters); beyond this, lines are wrapped;
               defaults to the width of the current window
```

As long as options 'c', 't', and 'T' are all three absent, 'c' and 't' are the defaults.

Values for the option 'w' must be in the range 66..10000. When using a w-value higher than the width of the current window, output will not wrap correctly in the window. With option 'f', wrapping is by default done at 80 characters (and not according to the curent window). This default can be changed with option 'w'.

Command 'help topic' (ht) [forw] <- topicname>

```
> Get information about a specific help topic.
> Options:
    f fname    write output also to file fname (append mode by default); implies
               'w80'
    o          modify option 'f': use overwrite mode, not append mode
    r          include subtopics
    w numcols  set line width (in characters); beyond this, lines are wrapped;
               defaults to the width of the current window
> Argument
    - topicname:   a dash followed by the topic name for which help is requested
                  (required)
```

For the documentation of topic 'xyz', enter 'help topic - xyz' ('ht-xyz'). The dash before the topic name is required to disambigue cases where a topic starts with a leter that is also an option of this help command.

By default, the topic name can be abbreviated (check command 'program set shortcommands' to change this). Use 'help summary t' ('hst') for a list of topics and their shortest abbreviations.

All help information is rendered in ASCII for the time being, so no diacritics are available...

Values for option 'w' must be in the range 66..10000. When using a w-value higher than the width of the current window, output will not wrap correctly in the window. With option 'f', wrapping is by default done at 80 characters (and not according to the curent window). This default can be changed with option 'w'.

Command 'import' (i)   {in, it}

```
> Import data from other file formats; requires a subcommand.
> Subcommands
    import nexus  in  execute a nexus datafile (supported nexus subset still
                     empty in this version...)
    import tnt    it  execute a TNT datafile (limited support for a tiny subset
                      of TNT commands)

--------------------------------------------------------------------------------
Command 'import nexus' (in)  <nexusfilename>

> Execute a nexus datafile (supported nexus subset still empty in this
  version...).
> Argument
    nexusfilename:   name of the nexus file (required)

Under construction.

--------------------------------------------------------------------------------
Command 'import tnt' (it)  <tntfilename>

> Execute a TNT datafile (limited support for a tiny subset of TNT commands).
> Argument
    tntfilename:   name of the file that contains TNT commands (required)

Use command 'help summary T' ('hsT') to get an overview an overview of supported
TNT commands. These are only available in TNT mode (command 'program set tntmode
=' or 'pst=') and in TNT files that are imported with this command.

More information about these supported TNT commands is available with command
'help' within TNT mode.

--------------------------------------------------------------------------------
Command 'log' (l)   {lp, lr, lsta, lsto}

> Name and status of log file, if there is one; has optional subcommands.
> Subcommands
    log pauze   lp    suspend output to the logfile
    log resume  lr    resume output to the logfile
    log start   lsta  open a log file (append mode by default)
    log stop    lsto  close the current logfile

--------------------------------------------------------------------------------
Command 'log pauze' (lp)

> Suspend output to the logfile.

--------------------------------------------------------------------------------
Command 'log resume' (lr)

> Resume output to the logfile.

--------------------------------------------------------------------------------
Command 'log start' (lsta)  [nof]

> Open a log file (append mode by default).
> Options:
    n              suppress logging of a header with time and date
    f logfilename  name of the logfile to open (required, append mode by
                   default)
    o              open the file in overwrite mode

When a logfile is open, all screen output is also written to the logfile.
```

This is a genaral logging file that logs output of all commands until it is pauzed or closed. In addition, several commands have options to specify logging of their output to a file. Such command-specific logging happens in addition to this general logging.

```
--------------------------------------------------------------------------------
Command 'log stop' (lsto)

> Close the current logfile.

--------------------------------------------------------------------------------
Command 'optimality' (o)   {os}

> Set/show optimality criterion; requires a subcommand.
> Subcommands
    optimality set  os  overview of settings that relate to the optimality
                        criterion; has optional subcommands

--------------------------------------------------------------------------------
Command 'optimality set' (os)   {osd, oss}

> Overview of settings that relate to the optimality criterion; has optional
  subcommands.
> Subcommands
    optimality set deviation   osd  set/show allowed deviation from optimality
                                    in tree searches and tree buffer cleaning
    optimality set searchmode  oss  set/show search mode: look for best (=,
                                    default) or worst (-) trees

--------------------------------------------------------------------------------
Command 'optimality set deviation' (osd)  <n>

> Set/show allowed deviation from optimality in tree searches and tree buffer
  cleaning.
> Argument
    n:   a non-negative integer
```

This value is used during swapping (command 'trees search') and when selecting optimal trees (command 'trees select best'). It has only limited influence while building initial trees for swapping.

This has consequences that at first may seem counterintuitive. Assume a strongly structured dataset for which 20 replicates of tree building and swapping (using command 'trees search mult') all return the same tree of score 100. Next the deviation from optimality is set to 50 and 20 new replicates also just return that same tree. This does not necessarily mean that there are no trees of lengths 101-150. What happens most likely is that the build itself already finds that tree in all replicates. And because it is already in the tree buffer, it is not passed on to swapping in the replicates.

The best thing to do then after increasing the deviation and before initiating new search replicates, is to explicitly swap the trees in the tree buffer (command 'trees search swap').

When the deviation is decreased, the tree buffer is left as it is. An explicit invocation of command 'trees select best' is required to make it reflect the new situation.

When looking for worst trees (see command 'optimality set searchmode'), the requested level of suboptimality is reported as a negative integer.

```
--------------------------------------------------------------------------------
Command 'optimality set searchmode' (oss)  [-=]

> Set/show search mode: look for best (=, default) or worst (-) trees.
> Options:
    =  search best trees
    -  search worst trees

Looking for worst trees may be useful to get an indication of the spread of
possible tree scores

--------------------------------------------------------------------------------
Command 'program' (p)   {pq, ps}

> Quit the program or set/show general settings; requires a subcommand.
> Subcommands
    program quit  pq  quit the program
    program set   ps  overview of general settings; has optional subcommands

--------------------------------------------------------------------------------
Command 'program quit' (pq)

> Quit the program.

Close all open files and quit the program.

--------------------------------------------------------------------------------
Command 'program set' (ps)   {psc, psl, psr, pss, pst, psu}

> Overview of general settings; has optional subcommands.
> Subcommands
    program set context        psc  set/show if context-sensitive help (command
                                    '>') is available from the command line (=)
                                    or not (-, default)
    program set longlists      psl  set/show if lists of command completions are
                                    multilevel (=) or not (-, default)
    program set randomseed     psr  set/show seed for generator of pseudorandom
                                    numbers
    program set shortcommands  pss  set/show if command abbreviations are
                                    allowed (=, default) or not (-)
    program set tntmode        pst  set/show if TNT mode is on
    program set unicode        psu  set/show if tree plotting can use multibyte
                                    UTF-8 characters (=, default) or not (-)

--------------------------------------------------------------------------------
Command 'program set context' (psc)  [-=]

> Set/show if context-sensitive help (command '>') is available from the command
  line (=) or not (-, default).
> Options:
    -  turn of availability of context-sensitive help command '>'
    =  turn off availability of context-sensitive help command '>'

When context-sensitive help is on, commands that have no possible subcommands,
options, and arguments are still terminated by hitting the 'enter' key. Likewise
for commands that only have '-' and '=' as options and for which at least one
option has been provided. In all other cases, commands have to be terminated
explicitly with a semicolon.
```

Still an experimental feature. It seems to be working quite well but needs to be thorougly tested.

When context-sensitive help is off, a semicolon is only required for the few commands that possibly take a long (structured) argument list.

Command 'program set longlists' (psl)

```
> Set/show if lists of command completions are multilevel (=) or not (-,
  default).
```

Affects the list of command completions that is shown when a command is specified as an ambiguous abbreviation, with a missing required subcommand, or with an invalid subcommand. For the completion lists that are used in the help commands, check the options for those commands.

Command 'program set randomseed' (psr) <n>

```
> Set/show seed for generator of pseudorandom numbers.
> Argument
    n:   a non-negative number
```

Without an argument, the current seed for the pseudorandom number generator is shown (initially 0 by default). With a non-negative argument n, the seed is set to n.

The generator used is the implementation of the ISO C function rand() in the C library that is used. The program uses pseudorandom numbers on such various occasions as generating a pseudorandom addition sequence to add terminals to a growing tree or to resample datasets (the latter is not available in this beta).

By setting the seed to a specific value at the start of the program, it is guaranteed that, say, a tree search that is performed after starting the program will by default return the same trees every time. This default behavior ensures reproducability of default searches after program startup but it is not suited for parallellization of replicates for a given data set by starting a number of parallel anagallis batch sessions, either manually or script-driven. In such cases, each parallel invocation has to set its own random seed.

Command 'program set shortcommands' (pss) [-=]

```
> Set/show if command abbreviations are allowed (=, default) or not (-).
> Options:
    =  abbreviations allowed (default)
    -  abbreviations not allowed
```

This setting only applies in regular program mode when context-sensitive help is off (command 'program set context'). It does not apply in TNT mode.

Command 'program set tntmode' (pst) [=-]

```
> Set/show if TNT mode is on.
> Options:
    =  switch to TNT mode
    -  switch to regular mode
```

```
--------------------------------------------------------------------------------
Command 'program set unicode' (psu)  [=-]

> Set/show if tree plotting can use multibyte UTF-8 characters (=, default) or
  not (-).
> Options:
    =  enable unicode characters when plotting trees (default)
    -  disable unicode characters when plotting trees (just use ASCII
       characters)

When plotting trees, the program by default uses some UTF-8 encoded unicode
characters that are longer than one byte.

Most terminals properly deal with such characters, but when exporting the output
to a word processor make sure to use a non-proportional font that properly deals
with UTF-8 encoded unicode characters. In openoffice 3.2, for example, font
'Courier 10 Pitch' is problematic but 'FreeMono' is ok.

This command can be used to disable the use of UTF-8 encoded unicode characters,
making the output of the program 100% ASCII.

Note that input is always assumed to be 100% ASCII. This command does not change
that.

--------------------------------------------------------------------------------
Command 'script' (s)   {se, sr}

> Overview of open script files; has optional subcommands.
> Subcommands
    script execute  se  overview of script files that are opened for execution;
                        has optional subcommands
    script record   sr  name and status of the file that is open for recording
                        commands, if there is one; has optional subcommands

> Examples
     Script files can be opened for execution or for recording commands that are
entered from the command prompt.

--------------------------------------------------------------------------------
Command 'script execute' (se)   {sep, ser, sesta, sesto}

> Overview of script files that are opened for execution; has optional
  subcommands.
> Subcommands
    script execute pauze   sep    temporarily suspend execution of current
                                  script file and get interactive input
    script execute resume  ser    resume reading from the current script file
                                  that is open for execution
    script execute start   sesta  open a script file and start executing its
                                  commands
    script execute stop    sesto  close the current script file that is open for
                                  execution

--------------------------------------------------------------------------------
Command 'script execute pauze' (sep)

> Temporarily suspend execution of current script file and get interactive
  input.
```

```
--------------------------------------------------------------------------------
Command 'script execute resume' (ser)

> Resume reading from the current script file that is open for execution.

--------------------------------------------------------------------------------
Command 'script execute start' (sesta)  [f]

> Open a script file and start executing its commands.
> Options:
    f fname  name of the file that contains anagallis commands to execute
             (required)

Script files for execution can be nested up to 16 levels deep.

--------------------------------------------------------------------------------
Command 'script execute stop' (sesto)

> Close the current script file that is open for execution.

--------------------------------------------------------------------------------
Command 'script record' (sr)   {srp, srr, srsta, srsto}

> Name and status of the file that is open for recording commands, if there is
  one; has optional subcommands.
> Subcommands
    script record pauze   srp    temporarily suspend writing to the current file
                                 for recording commands
    script record resume  srr    resume recording commands to the script file
                                 for recording
    script record start   srsta  open a file for recording commands (append mode
                                 by default)
    script record stop    srsto  close the current file for recording commands

When there is an open unsuspended script file for recording commands to, all
commands that are entered from the command prompt (or supplied with the program
invocation) are written to that file. Command abbreviations are expanded before
doing so. Commands that are read from a scriptfile are not included. Useful to
keep a history of an interactive session, to edit/replay the session later on,
or to expand abbreviated commands.

--------------------------------------------------------------------------------
Command 'script record pauze' (srp)

> Temporarily suspend writing to the current file for recording commands.

--------------------------------------------------------------------------------
Command 'script record resume' (srr)

> Resume recording commands to the script file for recording.

--------------------------------------------------------------------------------
Command 'script record start' (srsta)  [fon]

> Open a file for recording commands (append mode by default).
> Options:
    n                 suppress logging of a header with time and date
    f scriptfilename  name of the scriptfile to open ifor recording commands
                      (required, append mode by default)
    o                 open the file in overwrite mode
```

When a file for recording anagallis commands is open, all commands entered from the command prompt are written to this file. To enhance readability, all abbreviated commands are expanded.

This command can also be used to expand abbreviations in an existing scriptfile, albeit indirectly: first open a file to record to, next paste the contents of that scriptfile to the command prompt.

KNOWN BUG: when reading trees, the trees themselves don't make it to the scriptfile.

```
--------------------------------------------------------------------------------
Command 'script record stop' (srsto)

> Close the current file for recording commands.

--------------------------------------------------------------------------------
Command 'trees' (t)   {tc, trr, tsc, tsea, tsel, tset, tsh}

> Current number of trees in memory; has optional subcommands to do stuff with
  trees.
> Subcommands
    trees consense  tc    calculate consensus trees; requires a subcommand
    trees read      trr   read trees in parenthetical notation (use command
                          'import' to import TNT or nexus trees)
    trees score     tsc   list the score of one or more trees
    trees search    tsea  search trees; requires a subcommand
    trees select    tsel  manipulate trees in the tree buffer; requires a
                          subcommand
    trees set       tset  overview of tree related settings; has optional
                          subcommands
    trees show      tsh   show trees; requires a subcommand

--------------------------------------------------------------------------------
Command 'trees consense' (tc)   {tcm, tcs}

> Calculate consensus trees; requires a subcommand.
> Subcommands
    trees consense majority  tcm  majority rule consensus tree
    trees consense strict    tcs  strict consensus tree

--------------------------------------------------------------------------------
Command 'trees consense majority' (tcm)  [abcdDefgikKnosStTW] <scopes>

> Majority rule consensus tree.
> Options:
    a        label the terminals with their names (this is the default; it is
             overwritten when option 'n' is present; this option is useful to
             have terminal names in such cases as well)
    b        suppress numbering of internal nodes
    c n      n is the cutoff percentage (between 0 and 99): ony clades with
             higher occurrence are shown; for building a tree (default, option
             'p' and option 'w'), values below 50 are interpreted as 50 (but
             clades theat occur less frequently are still shown with the option
             'i')
    d        dry run to set custom defaults: remember all other options in this
             invocation for use with the following invocations in this session
             (with no other options, the built-in defaults are restored)
    D        dry run to show the current custom defaults
    i        when plotting subtrees that branch at the same level, plot small
             subtrees last, and equally sized non-leaf subtrees sorted according
             to their decreasing numeric code; this option also inverses the
             plot order of leaves that branch at the same level
    g n      use alternative ASCII glyph set 1 or 2 for plotting trees (has only
             effect under 'program set unicode -' or 'psu-')
    i        when plotting subtrees that branch at the same level, plot small
             subtrees last, and equally sized non-leaf subtrees sorted according
             to their decreasing numeric code; this option also inverses the
             plot order of leaves that branch at the same level
    k        condensed output (shorter branches)
    K n      truncate terminal names (to a minimum of n characters, n > 0) when
             they would exceed the specified width (option 'W') for plotting
    n        label the terminals with their numeric code (their sequential
             number in the data matrix; see option 'a' for more information)
    f fname  write output also to file fname (append mode by default)
    o        modifies option 'f': use overwite mode, not append mode
    s        sort terminals that branch at same level according to their
             increasing numeric code (default: ascending alphabetical order of
             terminal names)
    S        silent (suppress summary statement at start of output)
    t        interpret scopes that follow as trees to include; cannot be
             combined with option 'T'; this is the default interpretation of
             scopes
    T        interpret scopes that follow as trees to exclude; cannot be
             combined with option 't' or with default scopes
    W n      maximum width (in characters) of a single line (beyond this, the
             tree is broken into subtrees); use -1 for the width of the current
             window (default), 0 to turn off this feature (valid values 20 -
             5000)
> Argument
    scopes:   one or more tree scopes (defaults to all trees)

Scopes and options may be intermingled.

--------------------------------------------------------------------------------
Command 'trees consense strict' (tcs)  [abdDfgikKnosStTwW] <scopes>

> Strict consensus tree.
> Options:
    a        label the terminals with their names (this is the default; it is
             overwritten when option 'n' is present; this option is useful to
             have terminal names in such cases as well)
    b        suppress numbering of internal nodes
    d        dry run to set custom defaults: remember all other options in this
             invocation for use with the following invocations in this session
             (with no other options, the built-in defaults are restored)
    D        dry run to show the current custom defaults
    i        when plotting subtrees that branch at the same level, plot small
             subtrees last, and equally sized non-leaf subtrees sorted according
             to their decreasing numeric code; this option also inverses the
             plot order of leaves that branch at the same level
    g n      use alternative ASCII glyph set 1 or 2 for plotting trees (has only
             effect under 'program set unicode -' or 'psu-')
    k        condensed output (shorter branches)
    K n      truncate terminal names (to a minimum of n characters, n > 0) when
             they would exceed the specified width (option 'W') for plotting
    n        label the terminals with their numeric code (their sequential
             number in the data matrix; see option 'a' for more information)
    f fname  write output also to file fname (append mode by default)
    o        modifies option 'f': use overwite mode, not append mode
    s        sort terminals that branch at same level according to their
             increasing numeric code (default: ascending alphabetical order of
             terminal names)
    S        silent (suppress summary statement at start of output)
    t        interpret scopes that follow as trees to include; cannot be
             combined with option 'T'; this is the default interpretation of
             scopes
    T        interpret scopes that follow as trees to exclude; cannot be
             combined with option 't' or with default scopes
    w        write the consensus tree in parenthetical notation (and ignore the
             options to tweak plotting; see option 'p')
    W n      maximum width (in characters) of a single line (beyond this, the
             tree is broken into subtrees); use -1 for the width of the current
             window (default), 0 to turn off this feature (valid values 20 -
             5000)
> Argument
    scopes:   one or more tree scopes (defaults to all trees)

Scopes and options may be intermingled.

--------------------------------------------------------------------------------
Command 'trees read' (trr)  [a]

> Read trees in parenthetical notation (use command 'import' to import TNT or
  nexus trees).
> Options:
    a  the terminals in the trees are indicated with their alphanumeric names,
       not with their sequential number in the current dataset

By default, terminals are indicated using numbers that correspond to the current
dataset. The first terminal in the dataset is terminal 1 (not 0). Use option 'a'
to indicate the terminals with their names.

Parentheses within a tree must match and an outer pair of parentheses is
required (so '((1 2) (3 4))' is ok, '(1 2)(3 4)' not). Different trees within a
single statement must be separated using '*', the last tree must be followed by
a semicolon. This is so because reading trees is a command that can span
multiple lines; so the semicolon is required to indicate that no more input
trees will follow).

Polytomies are internally resolved in a further unspecified way. After being
read, the trees are evaluated according to the current dataset and the current
settings for collapsing zero-length branches (and therefore the number of
terminals and their labels must match). Only those that are not yet in the tree
buffer are retained.

Trees in other formats can be imported with command 'import'.

> Examples
     characters read numeric
      1 4
      a 0
      b 0
      c 1
      d 1
     ;
     trees read (1 2 (3 4)) * (1 4 (2 3));
     trr a
     (a c (b d))
    ;


--------------------------------------------------------------------------------
Command 'trees score' (tsc)  <treescopes>

> List the score of one or more trees.
> Argument
    tree scopes:   trees to show the score of (defaults to the current tree)

List the total (weighted) score of all active characters on the trees in the
specified scopes (defaults to the current tree).

--------------------------------------------------------------------------------
Command 'trees search' (tsea)   {tseam, tseas}

> Search trees; requires a subcommand.
> Subcommands
    trees search mult  tseam  do one or more replicates of building a tree and
                              swapping it (spr or tbr)
    trees search swap  tseas  swap trees from the tree buffer (spr or tbr)

--------------------------------------------------------------------------------
Command 'trees search mult' (tseam)  [*-ahkr] <n>

> Do one or more replicates of building a tree and swapping it (spr or tbr).
> Options:
    *    do tbr swapping (spr by default)
    -    skip swapping, just build an initial tree
    a    use addition sequence 'as is' in first replicate (default: random
         addition sequence)
    h n  hold at most n trees for each repetion
    k    keep the best trees of each replicate (even if they are not the best
         overall)
    r    use a random tree to initiate swapping
> Argument
    n:   repeat this n times (n > 0; defaults to 1)

Do a number of repetitions (1 by default) of building a tree and (by default)
swapping it. By default the build is done with a random terminal addition
sequence. With the 'r' option, the build just selects a random tree. With the
'a' option, the build of the first replicate uses the terminal addition sequence
as laid out in the dataset ('as is'). Documentation of details of swap process
(when switching to a wich next tree to swap and things like that): to be done.

During the build stage, character hierarchy definitions are skipped: the/a best
insertion point for the next terminal is searched as if no character
hierarachies have been defined (but the reported score of the full tree that is
obtained does take them into account afterwards). During swapping, defined
character hierarchies are properly taken into account.

Options and argument may be intermingled. When more than one number is
specified, the last one is used.

By default, only the best trees (according to current settings) over all
replicates are retained. With option 'k', the best trees of each replicate will
be retained.

See command 'trees search swap' for some comments on tree buffer maintenance.
```

```
During tree searches, the dot ('.') and the comma (',') have special meaning:
pressing the dot key during a tree search ends the current replicate, pressing
the comma key ends the complete search.

--------------------------------------------------------------------------------
Command 'trees search swap' (tseas)  [*k] <tree scopes>

> Swap trees from the tree buffer (spr or tbr).
> Options:
    *  do tbr swapping (spr by default)
    k  for each original tree being swapped, keep its best trees (even if they
       are not the best overall)
> Argument
    tree scopes:   trees to swap (required)

There is no default tree scope, so at least one tree scope must be specified
(this can be a trival scope of just one tree).  Multiple scopes may be
specified. Scopes and options may be intermingled. Use '.' to swap all trees in
memory.

Swapping any single starting tree from the specified scopes proceeds similarly
as swapping the starting tree that results from an initial build for a single
replicate with command 'trees search mult': all new trees that are derived from
the starting tree are themselves recursively swapped before proceeding to the
next starting tree; while swapping a starting tree, the numbers reported refer
to trees derived from that tree, not to all trees in the tree buffer; the best
length reported while swapping a starting tree refers to the best length
starting that starting tree, not from the current globally best length.

A difference with command 'trees search mult' is in tree buffer maintenance.
With command 'trees search mult', the tree buffer is checked and maintained
against the current best global length after each replicate. With command 'trees
search swap', global maintenance of the tree buffer is only performed after all
starting trees have been swapped.

While swapping, the dot ('.') and the comma (',') have special meaning: pressing
the dot key during a tree search ends swapping of the current tree from the
specified tree scops. pressing the comma key ends the complete search.

--------------------------------------------------------------------------------
Command 'trees select' (tsel)   {tselb, tseld, tselk, tselu}

> Manipulate trees in the tree buffer; requires a subcommand.
> Subcommands
    trees select best    tselb  discard suboptimal trees
    trees select delete  tseld  discard the trees in the specified tree scopes
    trees select keep    tselk  discard the trees that are outside the specified
                                tree scopes
    trees select unique  tselu  discard duplicate trees

--------------------------------------------------------------------------------
Command 'trees select best' (tselb)

> Discard suboptimal trees.

--------------------------------------------------------------------------------
Command 'trees select delete' (tseld)  <scopes>

> Discard the trees in the specified tree scopes.
> Argument
```

```
    scopes:   one or more tree scopes

--------------------------------------------------------------------------------
Command 'trees select keep' (tselk)  <scopes>

> Discard the trees that are outside the specified tree scopes.
> Argument
    scopes:   one or more tree scopes

--------------------------------------------------------------------------------
Command 'trees select unique' (tselu)

> Discard duplicate trees.
```

When the level of collapsing zero-length branches is increased to a more severe level (see command 'trees set zerocollapse'), trees that were different before may no longer be different. This command can then be used to weed out duplicates.

```
--------------------------------------------------------------------------------
Command 'trees set' (tset)   {tsetc, tseto, tsetw, tsetz}

> Overview of tree related settings; has optional subcommands.
> Subcommands
    trees set current       tsetc  set/show the default tree that is for example
                                   used when showing trees or character
                                   optimizations on trees
    trees set outgroup      tseto  set/show terminal(s) to be used as
                                   outgroup(s) when showing trees
    trees set width         tsetw  set/show default maximum width of a line when
                                   plotting trees
    trees set zerocollapse  tsetz  set/show the rule for collapsing zero-length
                                   branches

--------------------------------------------------------------------------------
Command 'trees set current' (tsetc)

> Set/show the default tree that is for example used when showing trees or
  character optimizations on trees.

--------------------------------------------------------------------------------
Command 'trees set outgroup' (tseto)  [a] <terminal number(s) or name(s)>

> Set/show terminal(s) to be used as outgroup(s) when showing trees.
> Options:
    a  force interpretation of argument(s) as terminal name(s)
> Argument
    terminal(s):   name(s) or numeric code(s) of terminal(s)
```

When one or more arguments are specified, this command sets the terminal(s) to be used as outgroup(s) when showing trees. All current tree evaluation algorithms are unrooted, so outgroups are extraneous to the analyses in the strict sense.

An outgroup terminal can be specified using its numeric code (its sequential number in the dataset) or its using its name, but both ways cannot be mixed in a single call. So if the first outgroup terminal is specified using its numeric code, all following outgroup terminals must be specified using numeric codes as well; and if the first outgroup terminal is specified using its name, then all following outgroup terminals must be specified using their names as well.

By default, numeric arguments are interpreted as numeric terminal codes. But anagallis allows terminal names to be numbers as well. In that case and if so required, option 'a' can be used to skip the default interpretation of numbers.

When multiple outgroups are specified, the first terminal is considered the primary outgroup. When showing any particular tree, the branch that is used for rooting is the branch that divides that tree in (1) the largest group that contains only outgroups and that includes the primary outgroup and (2) a group that also or uniquely contains non-outgroups.

Without arguments, the current settings are shown.

Command 'trees set width' (tsetw)

> Set/show default maximum width of a line when plotting trees.

Beyond the requested width, trees are broken into subtrees of appropriate sizes when plotted. The value as set here can be overwritten with option 'W' of the various commands that plot trees.

The default is the width of the window in which a tree is plotted. To restore this default, set the width to -1. Set the width to zero to turn off this feature (the program actually uses a large built-in maximum then). Lines that are longer than the current window will then just wrap to the next line or lines.

Command 'trees set zerocollapse' (tsetz) [012]

```
> Set/show the rule for collapsing zero-length branches.
> Options:
    0  don't collapse branches
    1  consider a branch supported when there is at least one character that may
       have a step on that branch (as 'ambiguous =' in Nona)
    2  consider a branch supported when there is at least one character that
       must have a step on that branch (as 'ambiguous -' in Nona). Default.
```

See Coddington and Scharff (1994) for background information. As known, the default rule for collapsing that is used here may occasionally overcollapse trees. But not to the degree that their strict consensus is affected (the majority rule consensus tree may be).

Whether or not a branch is collapsed is determined by operations on the final statesets at both ends of the branch. For a character that is part of a character hierarchy, those operations are currently done using the aggregeted statesets. It's probably better to loop over the non-aggregated statesets but the program doesn't do that yet.

Whether using aggregated statesets or non-aggregated statesets for characers in character hierarchies, characters with inapplicables pose yet another problem for collapsing branches. Assume state distribution ((1 -) (- 2)) on tree ((A B) (C D)). It has two optimizations: a first in which the two inner nodes have state '-', a second in which neither inner node has state '-'. The first optimization has two subcharacters and no transformations, the second has one subcharacter and one transformation. That transformation, between state 1 and state 2, has to occur on the path between A and D. So it can occur on the branch leading to A, on the inner branch, or on the branch leading to D. Depending on where the transformation is assumed to occur, both inner nodes can have state 1 or state 2. This is reflected in their (identical) optimized statesets as calculated by this program: [12] and [-] non-aggregated, [-12] aggregated. If support for the inner branch is calculated from such statesets (as is the case here), that inner branch will be collapsed, even if there exists an optimization on which a transformation occurs on that branch. That is ok under 'ambiguous -' but amounts to overcollapsing under 'ambiguous ='.One could alternatively mark that branch as supported under 'ambiguous -', but doing so leads to what might be called 'undercollapsing' in more complex cases. As an illustration, assume state distribution (1 (- (? (- 2)) on tree (A (B (C (D E))). Reasoning as above, all inner nodes have [-12] as their aggregated optimal stateset. The single transformation that may be present can now happen on the branch to A, on the branch to E, or on both inner branches. So these inner branches mght be considered supported under 'ambiguous -' and left uncollapsed. But they cannot be suported simultaneously by this character.

When switching to a more severe mode of collapsing, the tree buffer may end up with duplicate trees. Command 'trees select unique' can be used to remove the duplicates.

```
--------------------------------------------------------------------------------
Command 'trees show' (tsh)   {tshp, tshw}

> Show trees; requires a subcommand.
> Subcommands
    trees show plot   tshp  plot trees using character graphics
    trees show write  tshw  write trees in parenthetical notation

--------------------------------------------------------------------------------
Command 'trees show plot' (tshp)  [abdDfgikKnorsStTW] <scopes>

> Plot trees using character graphics.
> Options:
    a        label the terminals with their names (this is the default; it is
             overwritten when option 'n' is present; this option is useful to
             have terminal names in such cases as well)
    b        suppress numbering of internal nodes
    d        dry run to set custom defaults: remember all other options in this
             invocation for use with the following invocations in this session
             (with no other options, the built-in defaults are restored)
    D        dry run to show the current custom defaults
    i        when plotting subtrees that branch at the same level, plot small
             subtrees last, and equally sized non-leaf subtrees sorted according
             to their decreasing numeric code; this option also inverses the
             plot order of leaves that branch at the same level
    g n      use alternative ASCII glyph set 1 or 2 for plotting trees (has only
             effect under 'program set unicode -' or 'psu-')
    k        condensed output (shorter branches)
    K n      truncate terminal names (to a minimum of n characters, n > 0) when
             they would exceed the specified width (option 'W') for plotting
    n        label the terminals with their numeric code (their sequential
             number in the data matrix; see option 'a' for more information)
    f fname  write output also to file fname (append mode by default)
    o        modifies option 'f': use overwite mode, not append mode
    r        show the trees as fully resolved, using a further undefined
             minimal-score resolution of polytomies
    s        sort terminals that branch at same level according to their
             increasing numeric code (default: ascending alphabetical order of
             terminal names)
    S        silent (suppress summary statement at start of output)
    t        interpret scopes that follow as trees to include; cannot be
```

```
             combined with option 'T'; this is the default interpretation of
             scopes
    T        interpret scopes that follow as trees to exclude; cannot be
             combined with option 't' or with default scopes
    W n      maximum width (in characters) of a single line (beyond this, the
             tree is broken into subtrees); use -1 for the width of the current
             window (default), 0 to turn off this feature (valid values 20 -
             5000)
> Argument
    scopes:   one or more tree scopes, defaults to the current tree)
```

Scopes and options may be intermingled. When no scopes are present, the current tree is shown.

Default order of subtrees within a set of subtrees that branch at the same level is as follows: subtrees with less terminals are shown first; equally sized non-terminal subtrees are sorted by their increasing numeric code (their sequential order in the dataset). Terminals are shown in increasing alphabetical order.

Option 's' can be used to sort terminals that branch at the same level according to their increasing numeric code.

Option 'i' inverses ordering of subtrees that branch at the same level: smaller subtrees are plotted last, equally sized non-leaf subtrees are sorted according to their decreasing numeric code, and terminals either according to decreasing numeric code of the terminals (option 's' specified) or descending alphabetical order.

The sequential numeric labels of the inner nodes start from one more than the number of terminals. The sequence is determined using the in-order traversal of the complete tree with the default ordering of subtrees that branch at the same level, and starting from 1 more than the number of terminals. These labels keep being used when changing the default ordering.

```
Command 'trees show write' (tshw)  [adDeforRsStTuv] <scopes>

> Write trees in parenthetical notation.
> Options:
    a        label the terminals with their names (default: label them with
             their numeric code, their sequential order in the dataset)
    d        dry run to set custom defaults: remember all other options in this
             invocation for use with the following invocations in this session
             (with no other options, the built-in defaults are restored)
    D        dry run to show the current custom defaults
    e str    str must be 'ana' or 'tnt'; write the trees(s) as a readable
             statement for anagallis (ana) or TNT (tnt)
    f fname  write output also to file fname (append mode by default)
    o        modifies option 'f': use overwite mode, not append mode
    R        add a space between all tree elements, not just between terminals
    r        show the trees as fully resolved, using a further undefined
             minimal-score resolution of polytomies
    s        sort terminals that branch at same level according to their
             increasing numeric code (default: ascending alphabetical order of
             terminal names)
    S        silent (suppress summary statement at start of output)
    t        interpret scopes that follow as trees to include; cannot be
             combined with option 'T'; this is the default interpretation of
             scopes
    T        interpret scopes that follow as trees to exclude; cannot be
             combined with option 't' or with default scopes
    u        modifies r: use curly braces for unsupported nodes; no effect for
             TNT export (see option 'e')
    v        verbose (include tree scores in output); no effect for TNT export
             (see option 'e')
> Argument
    scopes:   one or more tree scopes, defaults to the current tree)
```

Scopes and options may be intermingled. When no scopes are present, the current tree is shown.

By default only supported nodes are shown. When such trees are used as input for anagallis later on, any polytomies that may occur will at that time be internally resolved using a further unspecified criterion. As a consequence, the reported scores may then be higher than the scores for the original trees. This can be avoided with option 'r': the trees are then written as they are internally resolved. Using such trees as input for anagallis later on will then restore the original internal resolution of any polytomies that may occur. As a result, reported tree scores will then be identical to the tree scores of the original trees.

By default space is only added between terminal names. With option 'R' extra space is added such that all tree elements (parentheses and terminals) are separated by a space.

TNT MODE COMMANDS

These are only available in TNT mode (command 'program set tntmode =') and in imported TNT files (command 'import tnt').

TNT mode command 'ccode' (c) [+-[]/!()=]

```
> Set/show character settings.
> Options:
    +   set the character additive for the following scopes (see -)
    -   set the character non-additive for the following scopes (see +)
    [   activate the following scopes (see ])
    ]   inactivate the following scopes (see [)
    /n  set prior character weight to weight n for the following scopes
    !   not supported
    (   not supported
    )   not supported
    =   not supported
```

Only the basic code specifiers are supported. The unsupported options are recognized but then ignored. This enables anagallis to read TNT files that have ccode statements with these options without flagging an error. Remember to start counting characters from zero when using this command interactively. The changes to character codes that are made in TNT mode persist when moving back to native mode (and the other way around as well).

TNT mode command 'help' (h) [+*[]

> Show documentation for the commands that are available in TNT mode and in

```
  imported TNT files.
> Options:
    +  not supported
    *  show complete documentation of all commands
    [  not supported

By default, only a brief description is provided for each command. Option '*'
gives more information. Use 'help xyz' for help on command 'xyz'.

--------------------------------------------------------------------------------
TNT mode command 'nstates' (n)  [*&/]

> Set the TNT default datatype.
> Options:
    *  not supported
    &  not supported
    /  not supported

Only 'nstates num n' and 'nstates n' are supported, with n an integer between 1
and 30. The number of available states depend on the value of n: for n up to 8,
ten states 0-9 will be available; for larger n, 30 states are available, coded
as 0-9 and a-t (case insensitive). This is not completely the same as in TNT,
but it does mostly ensure that sufficient states are available when importing
TNT files (in TNT, for n up to 8 there are 8 available states 0-7; for n from 9
up to 16 16 states, and from 17 onwards 32 states).

--------------------------------------------------------------------------------
TNT mode command 'program set tntmode' (pst)  [=-]

> Set/show if TNT mode is on.
> Options:
    =  switch to TNT mode
    -  switch to regular mode

--------------------------------------------------------------------------------
TNT mode command 'quit' (q)

> Leave TNT mode, go back to regular mode.

--------------------------------------------------------------------------------
TNT mode command 'tread' (t)

> Read trees in parenthetical notation (numbering of terminals starts from 0).

Read trees in parentetical notation. This command just reads the basic format of
parenthetical trees: None of TNT's special features are supported or even parsed
correctly (they will flag an error). A further restriction is that terminals
must be identified using their sequential number in the dataset (starting from
0), not by their names. When moving back to native mode, numbering shifts back
to counting from 1 onwards (and likewise the other way around).

--------------------------------------------------------------------------------
TNT mode command 'xread' (x)  [=*-[!/+>]

> Read alphanumeric or dna data (no support for interleaved data).
> Options:
    =  not supported
    *  not supported
    -  not supported
    [  not supported
    !  not supported
    /  not supported
    +  not supported
    >  not supported
```

TNT's xread options are recognized but then ignored. This enables anagallis to read basic character data from a TNT xread statement without flagging an error. These character data remain available when moving back to native mode (works the other way around as well). Interleaved input is not supported (an error will be triggered).
