The RAxML v8.0.0 Manual

by Alexandros Stamatakis
Heidelberg Institute for Theoretical Studies

I. About RAxML

RAxML (Randomized Axelerated Maximum Likelihood) is a program for sequential and parallel Maximum Likelihood based inference of large phylogenetic trees. It can also be used for post-analyses of sets of phylogenetic trees, analyses of alignments and, evolutionary placement of short reads.
It has originally been derived from fastDNAml which in turn was derived from Joe Felsentein's dnaml which is part of the PHYLIP package.

When using RAxML please cite:

A. Stamatakis. RAxML-VI-HPC: Maximum Likelihood-based Phylogenetic Analyses with Thousands of Taxa and Mixed Models. In *Bioinformatics* 22(21):2688-2690, 2006.

http://bioinformatics.oxfordjournals.org/content/22/21/2688.long

II. Getting help

RAxML support is provided via the RAxML Google group at: https://groups.google.com/forum/?hl=de#!forum/raxml Note that, Google groups have a search function!

Thus, before posting to the RAxML google group:

1. Search the group to see if your issue has not already been discussed!
2. If you don't want to get a rude reply, read this manual first!
3. Read a standard textbook about phylogenetics such as Ziheng Yang's excellent *Computational Molecular Evolution*. If you haven't read on you should not be using RAxML anyway.

Do never send emails with RAxML questions to A. Stamatakis directly!

A step by step tutorial with some basic commands is available at: http://sco.h-its.org/exelixis/web/software/raxml/hands_on.html

Additional help, links, little helper scripts and documentation is available at the RAxML software page: http://sco.h-its.org/exelixis/web/software/raxml/index.html

III. RAxML web-servers and Graphical User Interfaces

While there exist several web-servers that allow you to run RAxML, I am directly involved in running three of them.

1. The Cipres Portal web server: http://www.phylo.org/sub_sections/portal/
2. The web-server at vital IT in Switzerland: http://embnet.vital-it.ch/raxml-bb/
3. A dedicated server for the Evolutionary Placement Algorithm: http://epa.h-its.org/raxml

There is no official graphical user interface supported by me, but a GUI has been developed by researchers at the research museum in Frankfurt, which is available here: http://sourceforge.net/projects/raxmlgui/

Note that, I will not provide any sort of support for the GUI, you need to contact the original authors for this.

IV. Downloading RAxML

RAxML is open source under GNU GPL. It is distributed via Alexis github repository: https://github.com/stamatak/standard-RAxML where you can always download the most up to date version. Make sure to **watch** the github repository to remain up to date regarding code changes.

We do not provide any support whatsoever for previous versions of the code!

Version numbers follow the notation **x.y.z** where **x** changes with major code reorganizations, **y** changes when new features are added and **z** changes with bug fixes.

V. Compiling RAxML

RAxML comes in a lot of different flavors.

It can run sequentially, in parallel using either MPI (message passing interface) or PThreads for multi-core shared memory systems.

It also has a hybrid/combined PThreads ↔ MPI parallelization that uses MPI to distribute bootstrap replicates or independent tree searches to different shared memory nodes in a cluster while it uses PThreads to parallelize the likelihood calculations of single tree searches. We call this coarse grain (MPI) and fine-grain (PThreads) parallelism.

Thus, before compiling you need to know on what kind of system you intend to execute RAxML. Also note that, the MPI version only implements/offers a subset of the RAxML functionality, it can only distribute distinct tree searches to processors!

Another important thing to consider prior to compiling is what your target processor architecture is! Modern x86 processors are very powerful because the have so-called vector instructions!

Depending on how new your processor is it will support SSE3 vector instructions, or, if newer also the faster AVX or the even faster AVX2 vector instructions. These instructions are used by RAxML to substantially accelerate the likelihood and parsimony computations it conducts.

Thus, you should always try to compile the code in a way that best exploits the capabilities of your CPU(s). Note that, even most modern laptops have more than 1 CPU/core, hence you will probably always want to compile the PThreads version of RAxML.

Now let's have a look at the RAxML source code, when you download it it will be in a file called:

```
standard-RAxML-8.0.0.tar.gz
```

uncompress it by typing:

```
gunzip standard-RAxML-8.0.0.tar.gz
tar xf standard-RAxML-8.0.0.tar
```

and change into the directory that contains the source files and list the contents:

```
cd standard-RAxML-8.0.0/
ls
```

There is a subdirectory called `usefulScripts` that contains a couple of perl scripts for various little RAxML tasks.

Next, let's list the Makefiles:

```
ls Makefile.*
```

which will generate a listing looking like this:

Makefiles for sequential version, hybrid MPI/Pthreads version, sequential version for MACs using the clang compiler, MPI version, PThreads version, PThreads version for MACs using clang that all rely on the most recent AVX2 vector instructions:

```
Makefile.AVX2.gcc
Makefile.AVX2.HYBRID.gcc
Makefile.AVX2.mac
Makefile.AVX2.MPI.gcc
Makefile.AVX2.PTHREADS.gcc
Makefile.AVX2.PTHREADS.mac
```

Corresponding Makefiles using AVX instructions:

```
Makefile.AVX.gcc
Makefile.AVX.HYBRID.gcc
Makefile.AVX.mac
Makefile.AVX.MPI.gcc
Makefile.AVX.PTHREADS.gcc
Makefile.AVX.PTHREADS.mac
```

Corresponding Makefiles not using any vector instructions (only required when your CPU is more than 5-6 years old!). The file called: `Makefile.QuartetMPI.gcc` is a dedicate Makefile that implements a MPI parallelization of the quartet evaluation functionality, for details see the section describing the command line arguments!

```
Makefile.gcc
Makefile.HYBRID.gcc
Makefile.MPI.gcc
Makefile.PTHREADS.gcc
Makefile.PTHREADS.mac
Makefile.QuartetMPI.gcc
```

Corresponding Makefiles using SSE3 instructions:

```
Makefile.SSE3.gcc
Makefile.SSE3.HYBRID.gcc
Makefile.SSE3.mac
Makefile.SSE3.MPI.gcc
Makefile.SSE3.PTHREADS.gcc
Makefile.SSE3.PTHREADS.mac
Makefile.SSE3.QuartetMPI.gcc
```

Now assume that your computer supports AVX instructions, to compile the sequential version type:

```
make –f Makefile.AVX.gcc
```

this will generate a RAxML executable called:

```
raxmlHPC-AVX
```

Now, to also compile the PThreads version, first remove all object files generated by compiling the sequential version by typing:

```
rm *.o
```

and then type:

```
make -f Makefile.AVX.PTHREADS.gcc
```

which will generate an executable called

```
raxmlHPC-PTHREADS-AVX
```

Compiling all other program flavors is analogous, with the only difficulty that the MPI versions need mpicc, the MPI compiler. A common MPI compiler distribution is provided by OpenMPI and is easy to install on Ubuntu Linux, for instance.

When to use which Version?

The use of the sequential version is intended for small to medium datasets and for initial experiments to determine appropriate search parameters.
The Pthreads version will work well for very long alignments, but performance is extremely hardware-dependent!
However, even for relatively short alignments (1,900 taxa, 1,200bp, DNA data) we observed speedups of around factor 6.5 on an 8-core node.
For a long alignment (125 taxa, 20,000 base-pairs, DNA) we observed significant super-linear speedups of around 10-11 on an 8-core system.

Warning: Make sure to specify the exact number of CPUs available on your system via the `-T` option. If you start more threads than you have cores available, there will be a significant performance decrease!
The MPI-version is for executing really large production runs (i.e. 100 or 1,000 bootstraps) on a Linux cluster. You can also perform multiple inferences on larger datasets in parallel to find a best-known ML tree for your dataset.
Finally, the rapid BS algorithm and the associated ML search have also been parallelized with MPI.

Warning: Reduced functionality of MPI version! The current MPI-version only works properly if you specify the `-#` or `-N` option in the command line, since it has been designed to do multiple inferences or rapid/standard BS (bootstrap) searches in parallel!
For all remaining options, the usage of this type of coarse-grained parallelism does not make much sense!

Processor Affinity and Thread Pinning with the Pthreads Version

An important aspect if you want to use the Pthreads version of the program is to find out how your operating system/platform handles processor affinity of threads. Within the shared-memory or multi-core context processor affinity means that if you run, for instance, 4 threads on a 4-way CPU or 4 cores each individual thread should always run on the same CPU, that is, `thread0` on `CPU0`, `thread1` on `CPU1` etc.
This is important for efficiency, since cache entries can be continuously re-used if a thread, which works on the same part of the shared memory space, remains on the same CPU. If threads are moved around, for instance, `thread0` is initially executed on `CPU0` but then on `CPU4` etc. the cache memory of the CPU will have to be re-filled every time a thread is moved. With processor affinity enabled, performance improvements of 5% have been measured on sufficiently large and thus memory-intensive datasets.

There is a function that will automatically pin threads to CPUs, that is, enforce thread affinity, under LINUX/UNIX. Because this function might occasionally cause some error messages during compilation due to portability issues it is disabled by default. To enable it, you will need to comment out this flag:

```
#define _PORTABLE_PTHREADS
```

in the `axml.c` source file.

VI. RAxML Likelihood Values & Idiosyncrasies

It is very important to note that the likelihood values produced by RAxML can not be directly compared to likelihood values of other ML programs. However, the likelihood values of the current version are very similar to those obtained by other programs with respect to previous releases of RAxML (usually between +/− 1.0 log likelihood units of those obtained e.g. by PHYML or GARLI).

The above of course refers to evaluating the likelihood on identical trees, in terms of tree searches the programs will yield different tree topologies and hence different likelihoods in most cases anyway.

Note, that the deviations between PHYML/RAxML and GARLI likelihood values can sometimes be larger because GARLI uses a slightly different procedure to compute empirical base frequencies (Derrick Zwickl, personal communication, many years ago) while the method in RAxML is exactly the same as implemented in PHYML.

These deviations between RAxML/PHYML on the one side and GARLI on the other side appear to be larger on long multigene alignments. Also note that, even likelihood values obtained by different RAxML versions, especially should not be directly compared with each other either.

This is due to frequent code and data structure changes in the likelihood function implementation and model parameter optimization procedures!

Thus, if you want to compare topologies obtained by distinct ML programs with respect to their likelihood, make sure that you optimize branch lengths and model parameters of final topologies with one and the same program.
This can be done by either using the respective RAxML option (`-f e`) or, e.g., the corresponding option in PHYML (see http://www.atgc-montpellier.fr/phyml/).

Differences in Likelihood scores

In theory all ML programs implement the same mathematical function and should thus yield the same likelihood score for a fixed model and a given tree topology.
However, if we try to implement a numerical function on a finite machine we will unavoidably obtain rounding errors. Even if we change the sequence (or if it is changed by the compiler, which it usually is) of some operations applied to floating point or double precision arithmetics in our computer we will probably get different results

In my experiments I have observed differences among final likelihood values between GARLI, IQPNNI, PHYML, RAxML (every program showed a different value).

You can also experiment with this by removing the gcc optimization flag `-O2` in one of the RAxML Makfiles. This will yield much slower code, that is in theory mathematically equivalent to the optimized code, but will yield slightly different likelihood scores, due to re-ordered floating point operations.

My personal opinion is that the topological search (number of topologies analyzed) is much more important than exact likelihood scores to obtain *good* final ML trees.

Especially on large trees with more than 1,000 sequences the differences in likelihood scores induced by the topology are usually so large, that a very rough parameter optimization with an epsilon (RAxML –e option) of 1 log likelihood unit, i.e., if the difference epsilon between two successive model parameter optimization iterations is smaller than 1.0 we stop the optimization, will already clearly show the differences.

Note that, if you perform a bootstrap analysis you don't need to worry too much about likelihood values anyway, since usually you are only interested in the bootstrapped topologies.

The CAT model of rate heterogeneity

The name of this model has caused a lot of confusion because there is a CAT model also implemented in PhyloBayes (see http://megasun.bch.umontreal.ca/People/lartillot/www/index.htm)

Unfortunately, I was not aware of this when I introduced the CAT model in RAxML, the CAT model in RAxML is something completely different! However, I decided not to change the name for backward compatibility such that new RAxML version keep working with wrapper scripts.

Warning: It is not a good idea to use the CAT approximiation of rate heterogeneity on datasets with less than 50 taxa. In general there will not be enough data per alignment column available to reliably estimate the per-site rate parameters.

CAT has been designed to accelerate the computations on large datasets with many taxa! Please read the respective paper [12] to understand how CAT works, what the rate categories are (they are conceptually different from the discrete rate categories of the GAMMA model), and what the limitations of this method are. Continue here

The GTRCAT approximation is a computational work–around for the widely used (see [14] for interesting usage statistics) General Time Reversible (GTR [15]) model of nucleotide substitution under the model of rate heterogeneity [16, 17]. CAT is used in an analogous way to accomodate searches with rate heterogeneity in the AA substitution models.
There is a paper available [12] which describes what GTRCAT is and why I don't like GTRGAMMA despite the fact that is a beautiful Greek letter. The main idea behind GTRCAT is to allow for integration of rate heterogeneity into phylogenetic analyses at a significantly lower computational cost (about 4 times faster) and memory consumption (4 times lower). Essentially, GTRCAT represents a rather un-mathematical "quick & dirty" approach to rapidly navigate into portions of the tree space, where the trees score well under GTRGAMMA . However, due to the way individual rates are optimized and assigned to rate categories in GTRCAT (for details on this please read the paper [12]), the likelihood values computed by GTRCAT are completely meaningless. This means: NEVER COMPARE ALTERNATIVE TREE TOPOLOGIES USING THEIR CATbased LIKELIHOOD VALUES! You will probably obtain a biased assessment of trees. This is the reason why GTRCAT is called approximation instead of model. The same applies to the CAT approximation when used with AA data.
Finally, note that, in the few real-world phylogenetic studies I have worked on so far in collaboration with Biologists, we never received "nasty" reviewer comments for using the CAT approximation. A very

recent
phylogenetic analysis with RAxML in Nature also used the CAT approximation [18].

VII. Alignment input File Formats

Alignment & Tree Input Formats

Alignments: The input alignment format of RAxML is relaxed interleaved or sequential PHYLIP or FASTA. *Relaxed* means that sequence names can be of variable length between 1 up to 256 characters.
If you need longer taxon names you can adapt the constant `#define nmlngth 256` in the source file `axml.h` appropriately.
Moreover, RAxML is less sensitive with respect to the PHYLIP formatting (tabs, insets, etc) of interleaved PHYLIP files.

Trees: The input tree format is Newick, the RAxML input trees must not always be comprehensive, i.e., need not contain all taxa of the alignment.
See http://evolution.genetics.washington.edu/phylip/newicktree.html for details on the Newick format.

Alignment Error Checking

Many alignments need to be checked for the following errors/insufficiencies before running an analysis with RAxML or any other phylogenetic inference program.

RAxML automatically analyzes the alignment and checks for the following errors:

1. Identical Sequence name(s) appearing multiple times in an alignment, this can easily happen when you export a standard PHYLIP-file from some tool which truncates the sequence names to 8 or 10 characters.
2. Identical Sequence(s) that have different names but are exactly identical. This mostly happens when you excluded some hard-to-align alignment regions from your alignment and does not make sense to use.
3. Undetermined Column(s) that contain only ambiguous characters that will be treated as missing data, i.e. columns that entirely consist of `X, ?, *, –` for AA data and `N, O, X, ?, –` for DNA data (analogous for other data types)
4. Undetermined Sequence(s) that contain only ambiguous characters (see above) that will be treated as missing data.

Prohibited Character(s) in taxon names are names that contain any form of whitespace character, like blanks, tabulators, and carriage returns, as well as one of the following prohibited characters: `:` or `()` or `[]`.

In case that RAxML detects identical sequences and/or undetermined columns and was executed, e.g., with `-n alignmentName` it will automatically write an alignment file called `alignmentName.reduced` with identical sequences and/or undetermined columns removed.

If this is detected for a partitioned model analysis a respective model file `modelFileName.reduced` will also be written. In case RAxML encounters identical sequence names or undetermined sequences or illegal characters in taxon names it will exit with an error and you will have to fix your alignment.

VIII. The RAxML options

The single by far most important command is the RAxML help option that displays all options.

I also frequently use it because I can not remember them all. In the following I will assume that we are just using the sequential unvectorized code, that is, the raxmlHPC executable.

I will discuss each option and provide a simple usage example for it.

Thus, to get on-line help type:

```
raxmlHPC -h
```

and you will get the following, very long listing, that will be discussed at length below:

```
raxmlHPC
            -s sequenceFileName -n outputFileName -m substitutionModel
            [-a weightFileName] [-A secondaryStructureSubstModel]
            [-b bootstrapRandomNumberSeed] [-B wcCriterionThreshold]
            [-c numberOfCategories] [-C] [-d] [-D]
            [-e likelihoodEpsilon] [-E excludeFileName]
            [-f a|A|b|B|c|C|d|D|e|E|F|g|G|h|H|i|I|j|J|m|n|N|o|p|q|r|R|s|S|t|T|u|v|
                V|w|W|x|y]
            [-F]
            [-g groupingFileName] [-G placementThreshold] [-h] [-H]
            [-i initialRearrangementSetting]
            [-I autoFC|autoMR|autoMRE|autoMRE_IGN]
            [-j] [-J MR|MR_DROP|MRE|STRICT|STRICT_DROP|T_<PERCENT>] [-k] [-K]
            [-L MR|MRE|T_<PERCENT>] [-M]
            [-o outGroupName1[,outGroupName2[,...]]][-O]
            [-p parsimonyRandomSeed] [-P proteinModel]
            [-q multipleModelFileName] [-r binaryConstraintTree]
            [-R binaryModelParamFile] [-S secondaryStructureFile]
            [-t userStartingTree]
            [-T numberOfThreads] [-u] [-U] [-v] [-V] [-w outputDirectory]
            [-W slidingWindowSize]
            [-x rapidBootstrapRandomNumberSeed][-X][-y]
            [-Y quartetGroupingFileName|ancestralSequenceCandidatesFileName]
            [-z multipleTreesFile]
            [-#|-N numberOfRuns|autoFC|autoMR|autoMRE|autoMRE_IGN]
```

```
-a    Specify a column weight file name to assign individual weights to each
      column of the alignment. Those weights must be integers separated by any
      type and number of whitespaces whithin a separate file.
```

In addition, there must of course be as many weights as there are columns in your alignment. The contents of an example weight file could look like this:

```
5 1 1 2 1 1 1 1 1 1 1 2 1 1 3 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 1 1 1 4 1 1
```

Example: `raxmlHPC -a wgtFile -s alg.phy -p 12345 -m GTRCAT -n TEST`

```
-A    Specify one of the secondary structure substitution models implemented in
      RAxML. The same nomenclature as in the PHASE manual is used, available
      models: S6A, S6B, S6C, S6D, S6E, S7A, S7B, S7C, S7D, S7E, S7F, S16, S16A,
      S16B

      DEFAULT: 16-state GTR model (S16)
```

Note that, partitioning does not work with secondary structure models. That is a secondary structure can only be superimposed to a single partition. Also note that, you need of course to also specify a file defining the secondary structure via the `-S` option.

Example: `raxmlHPC -S secondaryStructureFile -s alg.phy -A S7D -p 12345 -m GTRGAMMA -n TEST`

`-b` Specify an integer number (random seed) and turn on bootstrapping

`DEFAULT: OFF`

This option allows you to turn on non-parametric bootstrapping. To allow for reproducibility of runs in the sequential program, you have to specify a random number seed, e.g., `-b 123476`.
Note however, that parallel bootstraps with the parallel version raxmlHPC-MPI are not reproducible despite the fact that you specify a random number seed.

Example: `raxmlHPC -b 12345 -p 12345 -# 100 -s alg -m GTRCAT -n TEST`

`-B` specify a floating point number between 0.0 and 1.0 that will be used as cutoff threshold for the MR-based bootstopping criteria. The recommended setting is 0.03.

`DEFAULT: 0.03 (recommended empirically determined setting)`

This setting allows to specify a threshold for the so-called bootstopping criteria that will automatically determine if you have conducted enough bootstrap replicate searches for obtaining stable support values. Note that, this only has an effect if you use the bootstopping criteria that rely on building majority rule consensus trees for determining convergence. The option will not have an effect when the – not recommended – frequency stopping criterion is being used. Please also read and cite the corresponding paper: http://link.springer.com/chapter/10.1007/978-3-642-02008-7_13#page-1

Example: `raxmlHPC -B 0.02 -b 12345 -p 12345 -# AUTOMR -s alg -m GTRCAT -n TEST`

`-c` Specify number of distinct rate categories for RAxML when model of rate heterogeneity is set to CAT.
Individual per-site rates are categorized into numberOfCategories rate categories to accelerate computations.

`DEFAULT: 25`

Example: `raxmlHPC -c 40 -p 12345 -s alg -m GTRCAT -n TEST`

Warning: Note that the setting of -c has no effect whatsoever on the number of discrete rate categories that are used to approximate the Gamma distribution of rate heterogeneity! RAxML always uses 4 discrete rate categories to approximate Gamma!

`-C` Enable verbose output for the "-L" and "-f i" options. This will produce more, as well as more verbose output files

`DEFAULT: OFF`

The above option will generate verbose output and additional output files when RAxML is used to compute the TC and IC measures as introduced by Salichos and Rokas: http://www.nature.com/nature/journal/v497/n7449/full/nature12130.html

Example: `raxmlHPC -L -m GTRCAT -L MRE -z treeSet -n TEST`

-d start ML optimization from random starting tree

DEFAULT: OFF

This option allows you to start the RAxML search with a complete random starting tree instead of the default randomized stepwise addition Maximum Parsimony starting tree. On smaller datasets (around 100–200 taxa) it has been observed that this might sometimes yield topologies of distinct local likelihood maxima which better correspond to empirical expectations.
It has also been observed that, this sometimes yield better – more diverse – starting trees for the analysis of broad phylogenomic alignments that have a very strong phylogenetic signal!

Example: `raxmlHPC -d -p 12345 -s alg -m GTRGAMMA -n TEST`

-D ML search convergence criterion. This will break off ML searches if the relative Robinson-Foulds distance between the trees obtained from two consecutive lazy SPR cycles is smaller or equal to 1%. Usage recommended for very large datasets in terms of taxa.
On trees with more than 500 taxa this will yield execution time improvements of approximately 50% while yielding only slightly worse trees.

DEFAULT: OFF

When enabling this option, RAxML will stop ML and standard, slow bootstrap searches early, when the RF distance between the trees generated by two consecutive cycles of SPR (Subtree Pruning Re-Grafting) moves is smaller than 1%. This leads to substantial speed improvements while the decrease in log likelihood scores is only very small. The option has been tested on several datasets and the results have been included in the following book chapter:
A. Stamatakis: "Phylogenetic Search Algorithms for Maximum Likelihood". In M. Elloumi, A.Y. Zomaya, editors. Algorithms in Computational Biology: techniques, Approaches and Applications, 547-577, John Wiley and Sons, 2011.

Example: `raxmlHPC -D -p 12345 -s alg -m GTRCAT -n TEST`

-e set model optimization precision in log likelihood units for final optimization of tree topology

```
DEFAULT: 0.1      for models not using proportion of invariant sites estimate
         0.001    for models using proportion of invariant sites estimate
```

This allows you to specify up to which likelihood difference, the model parameters will be optimized when RAxML uses the GAMMA model of rate or when you just evaluate a trees with the `-f e` option or a bunch of trees with the `-f n` option or similar options. This has shown to be useful to quickly evaluate the likelihood of a bunch of large final trees of more than 1,000 taxa because it will run much faster.
I typically use e.g. `-e 1.0` or `-e 2.0` in order to rapidly compare distinct final tree topologies based on their likelihood values.
Note that, topology-dependent likelihood-differences are typically far larger than 1.0 or 2.0 log likelihood units. The default setting is 0.1 log likelihood units which proves to be sufficient in most practical cases.

Example: `raxmlHPC -f e -e 0.00001 -s alg -t tree -m GTRGAMMA -n TEST`

`-E` specify an exclude file name, that contains a specification of alignment positions you wish to exclude.
Format is similar to Nexus, the file shall contain entries like "100-200 300-400", to exclude a single column write, e.g., "100-100", if you use a mixed model, an appropriately adapted model file will be written.

This option will just make RAxML write a reduced alignment file without the excluded columns that can then be used for the real analysis. If you use a partitioned model, an appropriately adapted model file will also be written.

Example: `raxmlHPC -E excludeFile -s alg -m GTRCAT -q part -n TEST`

In this case the alignment file with columns excluded will be named `alg.excludeFile` and the partition file with the specified columns excluded is called `part.excludeFile`

`-f` select algorithm:

The -f option is a very powerful option because, in many cases, it allows you to select what kind of algorithm RAxML shall execute. If you don't specify `-f` at all RAxML will execute the standard hill climbing algorithm (`-f d` option)The individual options are listed below.

`-f a` rapid Bootstrap analysis and search for best-scoring ML tree in one program run

Tell RAxML to conduct a rapid Bootstrap analysis and search for the best-scoring ML tree in one single program run.

Example: `raxmlHPC -f a -p 12345 -s alg -x 12345 -# 100 -m GTRCAT -n TEST`

`-f A` compute marginal ancestral states on a ROOTED reference tree provided via -t

This option allows you to compute marginal ancestral states/sequences on a given, fixed, and rooted reference tree. If you don't know what marginal ancestral states are please read Ziheng Yang's book on Computational Molecular Evolution.

Example: `raxmlHPC -f A -t testTree -s testData -m GTRGAMMA -n TEST`

A small phylip-formatted test alignment would look like this:

```
4 4
t1 ACGT
t2 AATT
t3 ATAT
t4 CCGT
```

and a test tree:

```
((t1,t2),(t3,t4));
```

RAxML will then output the rooted binary tree again, but with inner node labels that must be used to associate ancestral sequences with their corresponding positions in the tree.

It also writes two files, one containing the marginal probabilities for each inner node label as well as one containing guesses (by taking the maximum probability) for the actual ancestral sequences.

`-f b` draw bipartition information on a tree provided with -t (typically the best-knwon ML tree) based on multiple trees (e.g., from a bootstrap) in a file specifed by -z

Example: `raxmlHPC -f b -t ref -z trees -m GTRCAT -n TEST`

`-f B` optimize br-len scaler and other model parameters (GTR, alpha, etc.) on a tree provided with -t. The input tree passed via -t needs to contain branch lengths. The branch lengths will not be optimized, just scaled by a single common value.

This is a slightly idiosyncratic option that was mainly developed to work on conjunction with the PartitionFinder tool (see: http://www.robertlanfear.com/partitionfinder/).
The concept of branch length scalers is similar to the one implemented in MrBayes. That is, we assume that all partitions have the same branch lengths proportionally to each other. However, for each partition *p* we do a maximum likelihood estimate of a single scaling factor *s[p]* by which we multiply the branch lengths. Essentially this represents a parametrization trade-off between estimating a separate set of branch lengths for each partition (heavy increase in the number of parameters → danger of over-parametrization) and doing one common branch length estimate across all partitions (possible under-parametrization). Note that, this works only for evaluating a given, fixed tree with branch lengths and not for tree searches!

Example: `raxmlHPC -s alg.phy -t tree -m GTRGAMMA -f B -q part -n TEST`

`-f c` check if the alignment can be properly read by RAxML

Example: `raxmlHPC -f c -m GTRCAT -s alg -n TEST`

`-f C` ancestral sequence test for my PhD student Jiajie Zhang, users will also need to provide a list of taxon names via -Y separated by whitespaces

This is an experimental option right now, not ready for general use.

`-f d` new rapid hill-climbing

`DEFAULT: ON`

This is the default RAxML tree search algorithm and is substantially faster than the original search algorithm. It takes some shortcuts, but yields trees that are almost as good as the ones obtained from the full search algorithm. The algorithm is described and assessed in this paper here: http://link.springer.com/article/10.1007/s11265-007-0067-4#page-1

Example: `raxmlHPC -f d -m GTRCAT -p 12345 -s alg -n TEST`

`-f D` execute one or more rapid hill-climbing searches that will also generate RELL bootstraps

This is an implementation of the techniques introduced in the following paper: http://www.ncbi.nlm.nih.gov/pubmed/23418397 Some initial tests have shown that the RAxML implementation yields results that are well correlated with standard bootstrap values. Using this option is recommended when you have very large trees and don't have the resources/time for computing bootstrap replicates.

Example: `raxmlHPC -f D -m GTRCAT -p 12345 -s alg -n TEST`

`-f e` optimize model parameters+branch lengths for given input tree

You need to provide an input tree via the `-t` option.

Example: `raxmlHPC -f e -t ref -m GTRGAMMA -s alg -n TEST`

`-f E` execute very fast experimental tree search, at present only for testing

This option will execute a very fast tree search algorithm that will not try as hard to optimize the likelihood. It is intended for very large trees and follows a similar logic as the FastTree program: http://meta.microbesonline.org/fasttree/

Example: `raxmlHPC -f E -m GTRCAT -p 12345 -s alg -n TEST`

`-f F` execute fast experimental tree search, at present only for testing

This option will execute a fast tree search algorithm that will not try as hard to optimize the likelihood. It is intended for very large trees and follows a similar logic as the FastTree program: http://meta.microbesonline.org/fasttree/

Example: `raxmlHPC -f F -m GTRCAT -p 12345 -s alg -n TEST`

`-f g` compute per site log Likelihoods for one ore more trees passed via -z and write them to a file in treepuzzle format that can be read by CONSEL
The model parameters will be estimated on the first tree only!

Example: `raxmlHPC -f g -s alg -m GTRGAMMA -z trees -n TEST`

`-f G` compute per site log Likelihoods for one ore more trees passed via -z and write them to a file in treepuzzle format that can be read by CONSEL.
The model parameters will be re-estimated for each tree!

Example: `raxmlHPC -f G -s alg -m GTRGAMMA -z trees -n TEST`

`-f h` compute log likelihood test (SH-test) between best tree passed via -t and a bunch of other trees passed via -z
The model parameters will be estimated on the first tree only!

Example: `raxmlHPC -f h -t ref -z trees -s alg -m GTRGAMMA -n TEST`

`-f H` compute log likelihood test (SH-test) between best tree passed via -t and a bunch of other trees passed via -z
The model parameters will be re-estimated for each tree !

Example: `raxmlHPC -f H -t ref -z trees -s alg -m GTRGAMMA -n TEST`

`-f i` calculate IC and TC scores (Salichos and Rokas 2013 http://www.nature.com/nature/journal/v497/n7449/full/nature12130.html) on a reference tree provided with -t based on multiple trees (e.g., from a bootstrap) in a file specifed by -z

Example: `raxmlHPC -f i -m GTRCAT -t referenceTree -z bootstrapTrees -n TEST`

-f I a simple tree rooting algorithm for unrooted trees. It roots the tree by rooting it at the branch that best balances the subtree lengths (sum over branches in the subtrees) of the left and right subtree. A branch with an optimal balance does not always exist! You need to specify the tree you want to root via -t.

Example: `raxmlHPC -f I -m GTRCAT -t unrootedTree -n TEST`

-f j generate a bunch of bootstrapped alignment files from an original alignemnt file. You need to specify a seed with -b and the number of replicates with -#

Example: `raxmlHPC -f j -b 12345 -# 100 -s alg -m GTRCAT -n TEST`

Warning: Note that, if you are generating those BS replicate alignments for a dataset for which you subsequently intend to conduct a partitioned analysis you will also need to specify the partition file here via `-q` because RAxML re-samples sites on a per-partition basis!

-f J Compute SH-like support values on a given tree passed via -t.

This option will compute sh-like support values as described here http://www.ncbi.nlm.nih.gov/pubmed/20525638 on a given tree. The input tree is typically the best-known ML tree found by a RAxML analysis. Keep in mind that for applying the SH-like test the tree needs to be NNI (Nearest Neighbor Interchange) optimal. Thus, RAxML will initially try to apply NNI moves to further improve the tree and then compute the SH test for each inner branch of the tree.

Example: `raxmlHPC -f J -m GTRGAMMA -s alg -t tree -n TEST`

-f m compare bipartitions between two bunches of trees passed via "-t" and "-z" respectively. This will return the Pearson correlation between all bipartitions found in the two tree files. A file called RAxML_bipartitionFrequencies.outpuFileName will be printed that contains the pair-wise bipartition frequencies of the two sets

Example: `raxmlHPC -f m -t trees1 -z trees2 -s alg -m GTRCAT -n TEST`

-f n compute the log likelihood score of all trees contained in a tree file provided by -z
The model parameters will be estimated on the first tree only!

Example: `raxmlHPC -f n -z trees -s alg -m GTRGAMMA -n TEST`

-f N compute the log likelihood score of all trees contained in a tree file provided by -z
The model parameters will be re-estimated for each tree!

Example: `raxmlHPC -f N -z trees -s alg -m GTRGAMMA -n TEST`

-f o old and slower rapid hill-climbing without the heuristic cutoff described in http://link.springer.com/article/10.1007/s11265-007-0067-4#page-1

If you use this option you will typically get slightly better likelihood scores while the run times are expected to increase by factor 2 to 3.

Example: `raxmlHPC -f o -p 12345 -m GTRCAT -s alg -n TEST`

`-f p` `perform pure stepwise MP addition of new sequences to an incomplete starting tree and exit`

Example: `raxmlHPC -f p -t ref -p 12345 -s alg -m GTRCAT -n TEST`

`-f q` `fast quartet calculator`

This option will calculate the likelihood scores of all quartets for a given input alignment. Initially RAxML will construct a randomized stepwise addition order parsimony starting tree or you can also pass a given tree (e.g., the best-known ML tree) to RAxML via `-t`. This tree is used to estimate model parameters that will then remain fixed during the quartet calculations. For each quartet that is evaluated, only the branch lengths will be optimized.

The algorithm itself has three flavors: (i) it can randomly evaluate a specific number of quartets, (ii) it can evaluate all quartets, or (iii) it can evaluate quartets from a pre-specified quartet constraint tree file that must contain exactly 4 monophyletic and multi-furcating groups.

For details on the input format for the quartet constraint file, see the `-Y` option!

To not evaluate all possible quartets (keep in mind that this number grows quickly as a function of the taxa in the alignment/tree) you have to use either `-#` or `-N` to specify how many quartets you want to randomly evaluate.

Example 1: `raxmlHPC -m GTRGAMMA -t tree -s alg -f q -p 12345 -n TEST`

The above example will evaluate all possible quartets.

Example 2: `raxmlHPC -m GTRGAMMA -t tree -s alg -f q -p 12345 -N 100 -n TEST`

The above example will evaluate 100 randomly chosen quartets.

The output is written to a file called: `RaxML_quartets.TEST`

It looks like this:

```
Taxon names and indices:

Cow 1
Carp 2
Chicken 3
Human 4
Loach 5
Mouse 6
Rat 7
Seal 8
Whale 9
Frog 10

1 2 | 3 4: -2640.417355
1 3 | 2 4: -2640.810748
1 4 | 2 3: -2638.359608
```

```
1 2 | 3 5: -2552.181311
1 3 | 2 5: -2543.674965
...
```

First, each taxon name is assigned a number, then the quartets are represented as bipartitions, for instance, `1 2 | 3 4: -2640.417355` is the quartet `((Cow,Carp),(Chicken, Human))` and has a likelihood of `-2640.417355` .

The MPI version for this specific option that can be compiled with the corresponding Makefiles will distribute quartet evaluations across processors with MPI.

`-f r` compute pairwise Robinson-Foulds (RF) distances between all pairs of trees in a tree file passed via -z if the trees have node labels represented as integer support values the program will also compute two flavors of the weighted Robinson-Foulds (WRF) distance

Example: `raxmlHPC -m GTRCAT -z trees -f r -n TEST`

The two flavors of the Weighted RF distance that are compute are:

1. just add the support of those bipartitions contained in one tree, but not the other
2. add the support of those bipartitions contained in one tree, but not the other and also add the difference in support for each shared bipartition

`-f R` compute all pairwise Robinson-Foulds (RF) distances between a large reference tree passed via -t and many smaller trees (that must have a subset of the taxa of the large tree) passed via -z.
This option is intended for checking the plausibility of very large phylogenies that can not be inspected visually any more.

Example: `raxmlHPC -f R -m GTRCAT -t hugeTree -z manySmallTrees`

`-f s` split up a multi-gene partitioned alignment into the respective subalignments

Example: `raxmlHPC -f s -q part -s alg -m GTRCAT -n TEST`

Warning: Note that the subalignments will be named by the names you have given to the individual partitions. If your partition file contains two partitions called `part1` and `part2`, this command will generate two alignment files called `part1.phy` and `part2.phy`

`-f S` compute site-specific placement bias using a leave one out test inspired by the evolutionary placement algorithm

TODO

Example: `raxmlHPC -f S -s alg -t tree -m GTRGAMMA -N 100 -n TEST`

`-f t` do randomized tree searches on one fixed starting tree

This command will execute a specified number (by `-N` or `-#`) of randomized tree searches. The difference to the standard search algorithm is, that, given a starting tree it will apply SPR (subtree pruning re-grafting) moves in a randomized order and thus may fiend better trees.

Example: `raxmlHPC -f t -t -m GTRCAT -s alg -p 12345 -n TEST`

`-f T` do final thorough optimization of ML tree from rapid bootstrap search in stand-alone mode

This option allows to do a more thorough tree search that uses less lazy, i.e., more exhaustive SPR moves, in stand-alone mode. This algorithm is typically executed in the very end of a search done via `-f a`.

Example: `raxmlHPC -p 12345 -m GTRGAMMA -s alg -f T -t tree -n TEST`

`-f u` execute morphological weight calibration using maximum likelihood, this will return a weight vector. you need to provide a morphological alignment and a reference tree via -t

This option will determine to which degree the sites of a morphological alignment are congruent with a given molecular reference tree. As output it will generate a RAxML weight file that reflects the degree of congruence and that can be subsequently read via the `-a` option to conduct a more informed evolutionary placement of fossils using the evolutionary placement algorithm (`-f v` option). For more details, see the corresponding papers:
Evolutionary placement http://sysbio.oxfordjournals.org/content/60/3/291.short
Fossil calibration http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5586939&url=http%3A%2%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5586939

Example:
```
raxmlHPC -f u -m BINGAMMA -t molecularTree -s
morphologicalAlignment
```

`-f v` classify a bunch of environmental sequences into a reference tree using thorough read insertions you will need to start RAxML with a non-comprehensive reference tree and an alignment containing all sequences (reference + query)

For details on the design and performance of this algorithm, see: http://sysbio.oxfordjournals.org/content/60/3/291.short
RAxML will produce a couple of idiosyncratic output files for the placements, but also an output file according to the common file standard defined by Erick Matsen for his `pplacer` (see http://matsen.fhcrc.org/pplacer/) program that is similar to the EPA (Evolutionary Placement Algorithm) and myself.
The common file format is described in this paper here:
http://www.plosone.org/article/info_%3Adoi%2F10.1371%2Fjournal.pone.0031009

Example: `raxmlHPC -f v -s alg -t referenceTree -m GTRCAT -n TEST`

If you frequently want to insert different sequence into the same reference tree, there is a shortcut to make this more efficient, because the model parameters and branch lengths are estimated from scratch for the reference tree each time when you invoke the program. If you want to avoid this you proceed as follows.

Generate a binary file containing the model parameters for the reference tree

Example:
```
raxmlHPC -f e -m GTRGAMMA -s referenceAlignment -t referenceTree
        -n PARAMS
```

This will generate a file called: `RAxML_binaryModelParameters.PARAMS` which can then be read by the EPA to avoid re-optimizing parameters:

Example:
```
raxmlHPC -f v -R RAxML_binaryModelParameters.PARAMS
      -t RAxML_result.PARAMS -s alg -m GTRCAT -n TEST2
```

`-f V` classify a bunch of environmental sequences into a reference tree using thorough read insertions you will need to start RAxML with a non-comprehensive reference tree and an alignment containing all sequences (reference + query)

This is an experimental extension of the EPA for placing short reads in multi-gene datasets. It uses some computational shortcuts to make the placement faster. It has been used for the following paper: http://www.nature.com/nmeth/journal/v10/n12/full/nmeth.2693.html

Example:
```
raxmlHPC -f V -q part -s alg -t referenceTree -m GTRCAT -n TEST
```

Warning: this is a test implementation for more efficient handling of multi-gene & whole- genome datasets!

`-f w` compute ELW test on a bunch of trees passed via -z
`The model parameters will be estimated on the first tree only!`

Computes the Extended Likelihood Weights test by Strimmer and Rambaut, see http://www.ncbi.nlm.nih.gov/pubmed/11798428, on a set of trees.

Example:
```
raxmlHPC -f w -m GTRGAMMA -s alg -z trees -n TEST
```

`-f W` compute ELW test on a bunch of trees passed via -z
`The model parameters will be re-estimated for each tree`

Example:
```
raxmlHPC -f W -m GTRGAMMA -s alg -z trees -n TEST
```

`-f x` compute pair-wise ML distances, ML model parameters will be estimated on an MP starting tree or a user-defined tree passed via -t, only allowed for GAMMA-based models of rate heterogeneity

This option will just compute the pair-wise distances (branch lengths) between the sequences of an alignment using a maximum likelihood estimate. To obtain a model parameter estimate (GTR, alpha shape parameter etc.) it will initially either read in a user specified tree (e.g., the best-known ML tree) or generate a randomized stepwise addition parsimony starting tree if no input tree is provided. It then optimizes model parameters on this tree and then computes all pair-wises distances.

Example:
```
raxmlHPC -f x -p 12345 -s alg -m GTRGAMMA -n TEST
```

`-f y` classify a bunch of environmental sequences into a reference tree using parsimony you will need to start RAxML with a non-comprehensive reference tree and an alignment containing all sequences (reference + query)

This command is analogous to the `-f v` command with the only difference that it uses parsimony for placing reads into the reference tree. Because it uses parsimony it is orders of magnitude faster than the likelihood-based placement. You can use it for placing millions of reads into extremely large reference trees.

Example: `raxmlHPC -f y -m GTRCAT -s alg -t referenceTree -n TEST`

Warning: This algorithm is based on parsimony. Thus, a read may be placed into more than one, equally parsimonious position/branch of the reference tree. Be careful to take all equally parsimonious placements into account when analyzing the results!

```
-F    enable ML tree searches under CAT model for very large trees without
      switching to GAMMA in the end (saves memory).
      This option can also be used with the GAMMA models in order to avoid the
      thorough optimization of the best-scoring ML tree in the end.

      DEFAULT: OFF
```

When conducting analyses under the CAT model of rate heterogeneity in RAxML, the program will in most cases try to evaluate the GAMMA-based score and do some additional optimization of the tree under GAMMA in the very end of the search. If you want to avoid this because of time and/or memory constraints (GAMMA needs four times more memory than CAT) you can use this option.

Example: `raxmlHPC -F -m GTRCAT -p 12345 -s alg -N 10 -n TEST`

```
-g    specify the file name of a multifurcating constraint tree this tree does not
      need to be comprehensive, i.e. must not contain all taxa
```

This option frequently causes confusion among users for two main reasons: (i) if the constraint tree you provide is not comprehensive, that is, does not contain all taxa in the alignment, the taxa not included in the constraint are free, that is, they are allows to fall into any part of the tree! (ii) users often state that the RAxML tree does not correspond to their constraint tree. This is mostly due to the tree visualization tools that are used, depending at which top level trifurcation (remember that maximum likelihood trees are always unrooted!) the tree is rooted it may look as if it does not comply with the constraint. Please double-check twice and thrice before posting to the RAxML google group!
Finally also note that, any multi-furcations in the input tree will be resolved via a maximum likelihood search.

Example: `raxmlHPC -g constraintTree -m GTRGAMMA -p 12345 -s alg -n TEST`

```
-G    enable the ML-based evolutionary placement algorithm heuristics by
      specifying a threshold value (fraction of insertion branches to be evaluated
      using thorough insertions under ML).
```

This option can be used in conjunction with the -f v and -f V options to speed up evolutionary placements of short reads. A setting of 0.1 (10% of branches considered for thorough insertions) yields a good accuracy/speed trade-off. For more details you may actually consider to read the paper: http://sysbio.oxfordjournals.org/content/60/3/291.short

Example: `raxmlHPC -f v -G 0.1 -s alg -t referenceTree -m GTRCAT -n TEST`

```
-h    Display this help message.
```

Example: `raxmlHPC -h`

`-H` `Disable pattern compression.`

`DEFAULT: ON`

This option allows to disable alignment site compression in the input alignment. If alignments contain identical sites they can be compressed to become shorter and thus speed up the likelihood calculations. This option has been developed mostly for internal use in my lab.

Example: `raxmlHPC -H -s alg -p 12345 -m GTRCAT -n TEST`

Warning: using this option may considerably slow down RAxML

`-i` `Initial rearrangement setting for the subsequent application of topological changes phase`

This option allows to specify the radius (number of nodes away from the original pruning position) up to which pruned subtrees will be re-inserted in the course of SPR (subtree pruning re-grafting moves) during the tree search. By default this setting will be determined automatically by RAxML. The example below specifies a radius of 10, that is, subtrees will be inserted into all branches that are between 1 and 10 nodes away from their pruning position.

Example: `raxmlHPC -i 10 -s alg -m GTRCAT -p 12345 -n TEST`

`-I` `a posteriori bootstopping analysis. Use:`

```
-I autoFC for the frequency-based criterion
-I autoMR for the majority-rule consensus tree criterion
-I autoMRE for the extended majority-rule consensus tree criterion
-I autoMRE_IGN for metrics similar to MRE, but include bipartitions under
      the threshold whether they are compatible or not. This emulates MRE but
      is faster to compute.
```

`You also need to pass a tree file containing several bootstrap replicates via -z`

This option allows to carry out the bootstrap convergence test, that is, the test that determines if you have computed sufficient BS replicates for getting stable support values, a posteriori, that is after you have already computed, for instance, 100 bootstrap replicates. This option is particularly useful when you are doing large-scale tree searches on supercomputers. Here you would initially run 50 replicates and check if they are sufficient. Then you'd run another 50 and asses if the 100 trees you have now are sufficient, etc.
My favorite flavors in terms of convergence criteria are: `autoMRE` and if the tree is very large in terms of number of taxa (more than 1000) `autoMRE_IGN` to save time. Before using this option you should read the corresponding paper: http://link.springer.com/chapter/10.1007/978-3-642-02008-7_13#page-1

Example: `raxmlHPC -m GTRCAT -z RAxML_bootstrap.BS -I autoMRE -n TEST`

Note that, if you have several files containing bootstrap trees they can easily be concatenated by using the Linux `cat` command.

Example: `cat  RAxML_bootstrap.BS_1 RAxML_bootstrap.BS_2 > allBootstraps`

-j Specifies that intermediate tree files shall be written to file during the standard ML and BS tree searches.

DEFAULT: OFF

This will simply print out a couple of intermediate trees during the tree search and not the final tree only. The intermediate trees are written to files called: `RAxML_checkpoint.TEST.0`, `RAxML_checkpoint.TEST.1`, etc.

Example: `raxmlHPC -j -m GTRGAMMA -s alg -p 12345 -n TEST`

-J Compute majority rule consensus tree with "-J MR" or extended majority rule consensus tree with "-J MRE" or strict consensus tree with "-J STRICT". For a custom consensus treshold >= 50%, specify T_<NUM>, where 100 >= NUM >= 50. Options "-J STRICT_DROP" and "-J MR_DROP" will execute an algorithm that identifies dropsets which contain rogue taxa as proposed by Pattengale et al. in the paper "Uncovering hidden phylogenetic consensus".
You will also need to provide a tree file containing several UNROOTED trees via "-z"

This option actually triggers two different sets of algorithms, one set for building consensus trees and one set for finding rogue taxa. Also note that, these algorithms have been parallelized, that is, if you use the PThreads version they will run faster!

You should read and cite the following papers when using these options!

For consensus trees:
http://www.sciencedirect.com/science/article/pii/S1877750310000086

For identifying rogue taxa:
http://ieeexplore.ieee.org/xpl/login.jsptp=&arnumber=5710874&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5710874

The two examples below compute an extended majority rule consensus tree and a 75% majority rule consensus tree.

Example: `raxmlHPC -J MRE -z trees -m GTRCAT -n TEST`

Example: `raxmlHPC -J T_75 -z trees -m GTRCAT -n TEST`

The example below identifies rogues using the aforementioned dropset method and using a majority rule threshold. For details, please read the paper.

Example: `raxmlHPC -J MR_DROP -z trees -m GTRCAT -n TEST`

-k Specifies that bootstrapped trees should be printed with branch lengths.
The bootstraps will run a bit longer, because model parameters will be optimized at the end of each replicate under GAMMA or GAMMA+P-Invar respectively.

DEFAULT: OFF

Example: `raxmlHPC -b 12345 -p 12345 -# 100 -k -s alg -m GTRGAMMA -n TEST`

-K Specify one of the multi-state substitution models (max 32 states) implemented in RAxML.

Available models are: ORDERED, MK, GTR

DEFAULT: GTR model

Evidently, for this option to have an effect you need to have an alignment containing multi-state characters. If you have several partitions that consist of multi-state characters the model specified via -K will be applied to all models. Thus, it is not possible to assign different models to distinct multi-state partitions!

Example: `raxmlHPC -p 12345 -m MULTIGAMMA -s multiStateAlignment -n TEST`

-L Compute consensus trees labelled by IC supports and the overall TC value as proposed in Salichos and Rokas 2013.
Compute a majority rule consensus tree with -L MR or an extended majority rule consensus tree with -L MRE.
For a custom consensus treshold >= 50%, specify -L T_<NUM>, where 100 >= NUM >= 50.
You will of course also need to provide a tree file containing several UNROOTED trees via -z!

This option allows to compute the TC and IC metrics on a consensus tree, as described by Salichos and Rokas in this paper here:
http://www.nature.com/nature/journal/v497/n7449/full/nature12130.html

There is a dedicated section further down in this manual with detailed instructions.

Example: `raxmlHPC -m GTRCAT -L MRE -z trees -n TEST`

-m Model of Binary (Morphological), Nucleotide, Multi-State, or Amino Acid Substitution:

This is probably the most complex and confusing command line option because it can be used to specify a lot of things. A very important thing to be aware of is that, in case you use a partitioned alignment the data type in the partitions and the specific models of substitution to be used are actually specified in the partition file. Thus, if you use the `-q` option the only information that will be extracted from this string here that is passed via `-m` is which model of rate heterogeneity is going to be used. Note that, the rate heterogeneity model is always applied to all data partitions, hence you can not have one partition be analyzed under CAT and another one be analyzed under GAMMA.

In general, the term CAT in the strings always indicates that you want to use the CAT model of rate heterogeneity. The string CATI indicates

TODO comment about ASC and X

BINARY:

```
-m BINCAT[X]       Optimization of site-specific evolutionary rates which are
                   categorized into numberOfCategories distinct rate
                   categories for greater computational efficiency. Final tree
                   might be evaluated automatically under BINGAMMA, depending
                   on the tree search option.
                   With the optional "X" appendix you can specify a ML
                   estimate of base frequencies.

-m BINCATI[X]      Optimization of site-specific evolutionary rates which are
```

```
                  categorized into numberOfCategories distinct rate
                  categories for greater computational efficiency. Final tree
                  might be evaluated automatically under BINGAMMAI, depending
                  on the tree search option.
                  With the optional "X" appendix you can specify a ML
                  estimate of base frequencies.

-m ASC_BINCAT[X]  Optimization of site-specific evolutionary rates which are
                  categorized into numberOfCategories distinct rate
                  categories for greater computational efficiency. Final tree
                  might be evaluated automatically under BINGAMMA, depending
                  on the tree search option.
                  With the optional "X" appendix you can specify a ML
                  estimate of base frequencies.
                  The ASC prefix will correct the likelihood for
                  ascertainment bias.

-m BINGAMMA[X]    GAMMA model of rate heterogeneity (alpha parameter will be
                  estimated). With the optional "X" appendix you can specify
                  a ML estimate of base frequencies.
-m ASC_BINGAMMA[X] GAMMA model of rate heterogeneity (alpha parameter will be
                  estimated). The ASC prefix will correct the likelihood for
                  ascertainment bias.
                  With the optional "X" appendix you can specify a ML
                  estimate of base frequencies.
-m BINGAMMAI[X]   Same as BINGAMMA, but with estimate of proportion of
                  invariable sites. With the optional "X" appendix you can
                  specify a ML estimate of base frequencies.
```

NUCLEOTIDES:

```
-m GTRCAT[X]      GTR + Optimization of substitution rates + Optimization of
                  site-specific evolutionary rates which are categorized into
                  numberOfCategories distinct rate categories for greater
                  computational efficiency.  Final tree might be evaluated
                  under GTRGAMMA, depending on the tree search option.
                  With the optional "X" appendix you can specify a ML
                  estimate of base frequencies.

-m GTRCATI[X]     GTR + Optimization of substitution rates + Optimization of
                  site-specific evolutionary rates which are categorized into
                  numberOfCategories distinct rate categories for greater
                  computational efficiency.  Final tree might be evaluated
                  under GTRGAMMAI, depending on the tree search option.
                  With the optional "X" appendix you can specify a ML
                  estimate of base frequencies.

-m ASC_GTRCAT[X]  GTR + Optimization of substitution rates + Optimization of
                  site-specific evolutionary rates which are categorized into
                  numberOfCategories distinct rate categories for greater
                  computational efficiency.  Final tree might be evaluated
                  under GTRGAMMA, depending on the tree search option.
                  With the optional "X" appendix you can specify a ML
                  estimate of base frequencies.
                  The ASC prefix will correct the likelihood for
                  ascertainment bias.

-m GTRGAMMA[X]    GTR + Optimization of substitution rates + GAMMA model of
```

	rate heterogeneity (alpha parameter will be estimated). With the optional "X" appendix you can specify a ML estimate of base frequencies.
-m ASC_GTRGAMMA[X]	GTR + Optimization of substitution rates + GAMMA model of rate heterogeneity (alpha parameter will be estimated). The ASC prefix will correct the likelihood for ascertainment bias. With the optional "X" appendix you can specify a ML estimate of base frequencies.
-m GTRGAMMAI[X]	Same as GTRGAMMA, but with estimate of proportion of invariable sites. With the optional "X" appendix you can specify a ML estimate of base frequencies.

MULTI-STATE:

-m MULTICAT[X]	Optimization of site-specific evolutionary rates which are categorized into numberOfCategories distinct rate categories for greater computational efficiency. Final tree might be evaluated automatically under MULTIGAMMA, depending on the tree search option. With the optional "X" appendix you can specify a ML estimate of base frequencies.
-m MULTICATI[X]	Optimization of site-specific evolutionary rates which are categorized into numberOfCategories distinct rate categories for greater computational efficiency. Final tree might be evaluated automatically under MULTIGAMMAI, depending on the tree search option. With the optional "X" appendix you can specify a ML estimate of base frequencies.
-m ASC_MULTICAT[X]	Optimization of site-specific evolutionary rates which are categorized into numberOfCategories distinct rate categories for greater computational efficiency. Final tree might be evaluated automatically under MULTIGAMMA, depending on the tree search option. With the optional "X" appendix you can specify a ML estimate of base frequencies. The ASC prefix willl correct the likelihood for ascertainment bias.
-m MULTIGAMMA[X]	GAMMA model of rate heterogeneity (alpha parameter will be estimated). With the optional "X" appendix you can specify a ML estimate of base frequencies.
-m ASC_MULTIGAMMA[X]	GAMMA model of rate heterogeneity (alpha parameter will be estimated). The ASC prefix willl correct the likelihood for ascertainment bias. With the optional "X" appendix you can specify a ML estimate of base frequencies.
-m MULTIGAMMAI[X]	Same as MULTIGAMMA, but with estimate of proportion of invariable sites. With the optional "X" appendix you can specify a ML estimate of base frequencies. You can use up to 32 distinct character states to encode

multi-state regions, they must be used in the following order:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V
i.e., if you have 6 distinct character states you would use 0, 1, 2, 3, 4, 5 to encode these.
The substitution model for the multi-state regions can be selected via the "-K" option

AMINO ACIDS:

-m PROTCATmatrixName[F|X]	specified AA matrix + Optimization of substitution rates + Optimization of site-specific evolutionary rates which are categorized into numberOfCategories distinct rate categories for greater computational efficiency. Final tree might be evaluated automatically under PROTGAMMAmatrixName[F|X], depending on the tree search option. With the optional "X" appendix you can specify a ML estimate of base frequencies.
-m PROTCATImatrixName[F|X]	specified AA matrix + Optimization of substitution rates + Optimization of site-specific evolutionary rates which are categorized into numberOfCategories distinct rate categories for greater computational efficiency. Final tree might be evaluated automatically under PROTGAMMAImatrixName[F|X], depending on the tree search option. With the optional "X" appendix you can specify a ML estimate of base frequencies.
-m ASC_PROTCATmatrixName[F|X]	specified AA matrix + Optimization of substitution rates + Optimization of site-specific evolutionary rates which are categorized into numberOfCategories distinct rate categories for greater computational efficiency. Final tree might be evaluated automatically under PROTGAMMAmatrixName[F|X], depending on the tree search option. With the optional "X" appendix you can specify a ML estimate of base frequencies. The ASC prefix will correct the likelihood for ascertainment bias.
-m PROTGAMMAmatrixName[F|X]	specified AA matrix + Optimization of substitution rates + GAMMA model of rate heterogeneity (alpha parameter will be estimated). With the optional "X" appendix you can specify a ML estimate of base frequencies.
-m ASC_PROTGAMMAmatrixName[F|X]	specified AA matrix + Optimization of substitution rates + GAMMA model of rate heterogeneity (alpha parameter will be estimated). The ASC prefix will correct the likelihood for ascertainment bias. With the optional "X" appendix you can specify a ML estimate of base frequencies.
-m PROTGAMMAImatrixName[F|X]	Same as PROTGAMMAmatrixName[F|X], but with estimate of proportion of invariable sites. With the optional "X" appendix you can specify a ML estimate of base frequencies.

Available AA substitution models:

DAYHOFF, DCMUT, JTT, MTREV, WAG, RTREV, CPREV, VT, BLOSUM62, MTMAM, LG, MTART, MTZOA, PMB, HIVB, HIVW, JTTDCMUT, FLU, DUMMY, DUMMY2, AUTO, LG4M, LG4X, PROT_FILE, GTR_UNLINKED, GTR

With the optional "F" appendix you can specify if you want to use empirical base frequencies
Please note that for mixed models you can in addition specify the per-gene AA model in the mixed model file (see manual for details). Also note that if you estimate AA GTR parameters on a partitioned dataset, they will be linked (estimated jointly) across all partitions to avoid over-parametrization

TODO examples

-M Switch on estimation of individual per-partition branch lengths. Only has effect when used in combination with "-q"
Branch lengths for individual partitions will be printed to separate files
A weighted average of the branch lengths is computed by using the respective partition lengths

DEFAULT: OFF

This will enable an independent per-partition estimate of branch lengths. Note that this increases dramatically the number of free parameters in your model. An unrooted binary tree has *2n-3* branch lengths were n is the number of taxa. If you use this option, instead of *2n-3* parameters you will get *p(2n-3)* parameters where *p* is the number of partitions. RAxML will also run notably slower if you use this option.
Apart from the normal output tree, RAxML will also generate output files for each partition individually that allow to recover the per-partition branch lengths if needed.

Example: `raxmlHPC -p 12345 -M -m GTRGAMMA -s alg -q part -n TEST`

-n Specifies the name of the output file.

This option has to be always specified. The arbitrary name passed via `-n` will be appended to all RAxML output files such that you know which files have been generated by which invocation.
If you intend to do two runs that write files into the same directory with the same name specified by -n the program will exit with an error to prevent you over-writing output files from a previous run.

Example:
```
raxmlHPC -p 12345 -m GTRGAMMA -s alg
-n MySuperDuperRAxMLOutputFileName
```

-o Specify the name of a single outgrpoup or a comma-separated list of outgroups, e.g., -o Rat or -o Rat,Mouse, in case that multiple outgroups are not monophyletic the first name in the list will be selected as outgroup, don't leave spaces between taxon names!

If there is more than one outgroup a check for monophyly of the outgrpoup in the respective output/final tree(s) will be performed. If the outgroup is not monophyletic the tree will be rooted at the first outgroup in the list and a respective
warning will be printed.
Keep in mind that outgroups are just a tree drawing option, the trees remain, essentially unrooted trees.

Example: `raxmlHPC -s alg -p 12345 -m GTRGAMMA -o Rat,Mouse -n TEST`

`-O` Disable check for completely undetermined sequence in alignment.
The program will not exit with an error message when "-O" is specified.

DEFAULT: check enabled

In general it does not make sense to analyze alignments where one or more sequences consist of completely undetermined characters (e.g., `-,N,?,X` etc.) because you don't have any information about these sequences and they will simply randomly scatter throughout the tree. Normally RAxML will exit with an error message when it detects such sequences. This option here can disable this behavior, but be warned that you really need to know what you are doing when using this option.

Example: `raxmlHPC -s alg -p 12345 -m GTRGAMMA -O -n TEST`

`-p` Specify a random number seed for the parsimony inferences. This allows you to reproduce your results and will help me debug the program.

For all options/algorithms in RAxML that require some sort of randomization, this option must be specified. Make sure to pass different random number seeds to RAxML and not only 12345 as I have done in the examples. When not specifying -p when it is required by RAxML, the program will exit with a respective error message.
In the example below (a simple tree search for the ML tree) the random number seed is required for randomized stepwise addition order parsimony starting tree that is computed prior to the actual ML optimization.

Example: `raxmlHPC -s alg -p 2352890 -m GTRGAMMA -n TEST`

`-P` Specify the file name of a user-defined AA (Protein) substitution model. This file must contain 420 entries, the first 400 being the AA substitution rates (this must be a symmetric matrix) and the last 20 are the empirical base frequencies

Note that, the substitution model you specify via `-m PROTGAMMAWAG` in the example below will be ignored here. RAxML will only extract the information that this is a protein data alignment and that you want to use the GAMMA model of rate heterogeneity from the string and will ignore `WAG`.
If you want to use your own protein substitution models for partitioned datasets, the substitution model files that have the same format as described here need to be specified in the partition file.

Example:
```
raxmlHPC -s alg -p 12345 -P myProteinModel -m PROTGAMMAWAG
-n TEST
```

`-q` Specify the file name which contains the assignment of models to alignment partitions for multiple models of substitution. For the syntax of this file please consult the manual.

This option allows you to specify the regions of your alignment for which an individual model of nucleotide substitution should be estimated. They can also contain different types of data.
This will typically be useful to infer trees for long (in terms of base–pairs) multigene alignments. If, e.g.,-m GTRGAMMA is used, individual alpha-shape parameters, GTR-rates, and empirical base frequencies will be estimated and optimized for each partition.
Since RAxML can handle alignments consisting of different data types (binary data, DNA data, protein data etc.) you must specify the type of datafor each partition in the partition file, before the partition name.
For DNA data this just means that you have to add `DNA` to each line in the partition file, for AA data this is done by specifying the respective AA substitution matrix (e.g., `WAG` or `LG`) you want to use for a partition. For binary data you'd specify `BIN` and for multi-state data `MULTI`.
Now, if you want to analyze a specific partition using an ascertainment bias correction you need to prepend `ASC_` to the data type, e.g., `ASC_WAG` or `ASC_DNA`.
If for a specific partition you want to use a maximum likelihood estimate for the base frequencies, you'd append `X` to the data type name, e.g., `DNAX` or `LGX`.
If you want to use empirical base frequencies (instead of the default pre-defined ones that ship with the models) you'd write, e.g., `WAGF` or `JTTF`.
If you want to use a protein substitution model of your own for a specific partition, it must be in the same format as specified for the `-P` option. Instead of specifying the data type with `WAG` for instance, you will need to specify the protein substitution model file name in square brackets, e.g., `[myProtenSubstitutionModelFileName]`.

If you want to do a partitioned analysis of concatenated AA and DNA partitions you can either specify `-m GTRGAMMA` or, e.g., `-m PROTGAMMAWAG`. The only thing that will be extracted from the string passed via -m is the model of rate heterogeneity you want to use.

If you have a pure DNA alignment with 1,000bp from two genes gene1 (positions 1–500) and gene2(positions 501–1,000) the information in the partition file should look as follows:

```
DNA, gene1 = 1-500
DNA, gene2 = 501-1000
```

If you want an ML estimate of frequencies for the first partition it will look like this:

```
DNAX, gene1 = 1-500
DNA, gene2 = 501-1000
```

To analyze the first partition with ascertainment bias correction you need to write:

```
ASC_DNA, gene1 = 1-500
DNA, gene2 = 501-1000
```

If gene1 is scattered through the alignment, e.g. positions 1–200, and 800–1,000 you specify this with:

```
DNA, gene1 = 1-200, 800-1,000
DNA, gene2 = 201-799
```

You can also assign distinct models to the codon positions, i.e. if you want a distinct model to be estimated for each codon position in gene1 you can specify:

```
DNA, gene1codon1 = 1-500\3
DNA, gene1codon2 = 2-500\3
```

```
DNA, gene1codon3 = 3-500\3
DNA, gene2 = 501-1000
```

If you only need a distinct model for the 3rd codon position you can write:

```
DNA, gene1codon1andcodon2 = 1-500\3, 2-500\3
DNA, gene1codon3 = 3-500\3
DNA, gene2 = 501-1000
```

As already mentioned, for AA data you must specify the transition matrices for each partition:

```
JTT, gene1 = 1-500
WAGF, gene2 = 501-800
WAG, gene3 = 801-1000
```

The AA substitution model must be the first entry in each line and must be separated by a comma from the gene/partition name, just like the DNA token above. You can not assign different models of rate heterogeneity to different partitions, i.e., it will be either CAT , GAMMA , GAMMAI etc. for all partitions, as specified with `-m` .

If you want to use a protein model of your own, for partition one, you'd write:

```
[myProtenSubstitutionModelFileName], gene1 = 1-500
WAGF, gene2 = 501-800
WAG, gene3 = 801-1000
```

To use a maximum likelihoo estimate of the base frequencies for all partitions you'd write:

```
JTTX, gene1 = 1-500
WAGX, gene2 = 501-800
WAGX, gene3 = 801-1000
```

Finally, if you have a concatenated DNA and AA alignment, with DNA data at positions 1–500 and AA data at 501-1,000 with the WAG model the partition file should look as follows:

```
DNA, gene1 = 1-500
WAG, gene2 = 501-1000
```

A partition file for binary data would look like this:

```
BIN, gene1 = 1-500
BIN, gene2 = 501-1000
```

and for multi-state data

```
MULTI, gene1 = 1-500
MULTI, gene2 = 501-1000
```

Example: `raxmlHPC -s alg -m GTRGAMMA -q part -p 12345 -n TEST`

```
-r    Specify the file name of a binary constraint tree.
      This tree does not need to be comprehensive, i.e. must not contain all taxa
```

This option allows you to pass a binary/bifurcating constraint/backbone tree in NEWICK format to RAxML.
Note, that using this option only makes sense if this tree contains less taxa than the input

alignment. The remaining taxa will initially be added by using parsimony criterion. Once a comprehensive tree with all taxa has been obtained it will be optimized under ML respecting the restrictions of the binary constraint tree.

Thus, option will not change the structure of the binary backbone tree you provide as input at all. What it will do is to just add the taxa not contained in the binary backbone tree to the best positions based on their likelihood. The result is a fully resolved binary tree.

Example: `raxmlHPC -s alg -m GTRGAMMA -r constr -p 12345 -n TEST`

`-R` `Specify the file name of a binary model parameter file that has previously been generated with RAxML using the -f e tree evaluation option. The file name should be: RAxML_binaryModelParameters.runID`

This option can be used to save time for some other RAxML options, by not re-estimating the model parameters of a fixed tree again, but only doing this once. The binary model parameter input file is always automatically generated by the `-f e` tree evaluation function.

It is useful with the `-f v` option when you want to place different reads into the same reference tree and with the quartet evaluation function `-f q`, in particular the parallel version of it!

Example:

```
raxmlHPC -f v -R RAxML_binaryModelParameters.PARAMS
-t RAxML_result.PARAMS -s alg -m GTRCAT -n TEST2
```

`-s` `Specify the name of the alignment data file in PHYLIP or FASTA format`

Specify the name of the alignment data file which can be in relaxed PHYLIP format. Relaxed means that you don't have to worry if the sequence file is interleaved or sequential and that the taxon names are too long. What you do need to worry about though is that there always needs to be a space between the taxon name and the sequence.
Sequence names can be of variable length between 1 up to 256 characters. If you need longer taxon names you can adapt the constant `#define nmlngth 256` in file `axml.h` appropriately. Moreover, RAxML is not sensitive with respect to the formatting (tabs, insets, etc) of interleaved PHYLIP files.

RAxML can now also parse FASTA format. If RAxML notices that it can't parse a phylip format it will try to parse the alignment file as FASTA format. So far it has been able to parse all FASTA files.

A plethora of small example input files for different data types can be found in the step-by-step on-line tutorial: http://sco.h-its.org/exelixis/web/software/raxml/hands_on.html

Example: `raxmlHPC -s alignment.fasta -m GTRGAMMA -p 12345 -n TEST`

`-S` `Specify the name of a secondary structure file. The file can contain "." for alignment columns that do not form part of a stem and characters "()<>[]{}" to define stem regions and pseudoknots`

Specifying secondary structure models for an RNA alignment works slightly differently than passing other datatypes to RAxML because we read in a plain RNA alignment and then need to tell RAxML by an additional text file that is passed via `-S` which RNA alignment sites need to be grouped/evolve together.
We do this in a standard bracket notation written into a plain text file, e.g., our DNA test alignment has 60 sites, thus our secondary structure file needs to contain a string of 60 characters like this one:

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

The '.' symbol indicates that this is just a normal RNA site while the brackets indicate stems. Evidently, the number of opening and closing brackets mus match. In addition, it is also possible to specify pseudo knots with additional symbols: `<>[]{}` for instance:

```
..................((.......)).......{....(....}....)........
```

In terms of models there are 6-state, 7-state and 16-state models for accommodating secondary structure that are specified via `-A`.
Available models are: `S6A, S6B, S6C, S6D, S6E, S7A, S7B, S7C, S7D, S7E, S7F, S16, S16A, S16B`. The default is the GTR 16-state model (`-A S16`). In RAxML the same nomenclature as in PHASE is used, so please consult the phase manual at http://intranet.cs.man.ac.uk/ai//Software/phase/phase-2.0-manual.pdf for a nice and detailed description of these models.

Example: `raxmlHPC -m GTRGAMMA -p 12345 -S secondaryStructure.txt -s rna.phy -n TEST`

A common question is whether secondary structure models can also be partitioned. This is presently not possible. However, you can partition the underlying RNA data, e.g., use two partitions for our DNA dataset as before. What RAxML will do internally though is to generate a third partition for secondary structure that does not take into account that distinct secondary structure site pairs may be part of different partitions of the alignment.

`-t` `Specify a user starting tree file name in Newick format`

Specifies a user starting tree file name which must be in Newick format. The branch lengths of that tree will generally be ignored for most options and re-estimated instead by RAxML. Note, that you can also specify a non-comprehensive (not containing all taxa in the alignment) starting tree. This might be useful if newly aligned/sequenced taxa have been added to your alignment and you want to extend the current tree.
Initially, taxa will be added to the tree using parsimony. The comprehensive tree will then be optimized under ML.

Example: `raxmlHPC -s alg -m GTRGAMMA -t tree -p 12345 -n TEST`

`-T` `PTHREADS VERSION ONLY! Specify the number of threads you want to run. Make sure to set "-T" to at most the number of CPUs you have on your machine, otherwise, there will be a huge performance decrease!`

`T` is set to 0 by default, the Pthreads version will produce an error if you do not set `-T` to at least 2.

Example: `raxmlHPC-PTHREADS -T 4 -s alg -m GTRGAMMA -p 12345 -n TEST`

`-u` `use the median for the discrete approximation of the GAMMA model of rate heterogeneity`

`DEFAULT: OFF`

If you specify this option, RAxML will use the median instead of the mean for the discrete Gamma model of rate heterogeneity. Typically, using the median will yield slightly better likelihood scores.

Example: `raxmlHPC -p 12345 -u -s alg -m GTRGAMMA -n TEST`

`-U` Try to save memory by using SEV-based implementation for gap columns on large gappy alignments. The technique is described here: http://www.biomedcentral.com/1471-2105/12/470
This will only work for DNA and/or PROTEIN data and only with the SSE3 or AVX-vextorized version of the code.

This option can help you to save memory and potentially also time on large gappy multi-gene alignemnts with missing data, in particular in cases where you have a typical gappy partitioned alignment (data for certain taxa missing for certain genes/partitions). The amount of saved memory is roughly proportional to the proportion of missing data.

Example: `raxmlHPC -p 12345 -U -s alg -m GTRGAMMA -n TEST`

`-v` Display version information

Displays the RAxML version you are using.

Example: `raxmlHPC -v`

`-V` Disable rate heterogeneity among sites model and use one without rate heterogeneity instead. Only works if you specify the CAT model of rate heterogeneity.

`DEFAULT: use rate heterogeneity`

This option can be used if your dataset better fits a model without rate heterogeneity. This is rare, but such datasets do exist.

Example: `raxmlHPC -m GTRCAT -s alg -p 12345 -V -n TEST`

`-w` FULL (!) path to the directory into which RAxML shall write its output files

`DEFAULT: current directory`

Name of the working directory where RAxML shall write its output files to. Note that you need to specify the full path and not the relative path!

Example:
```
raxmlHPC -m GTRCAT -p 12345 -s alg
-w /home/stamatak/Desktop/myAnalysys -n TEST
```

`-W` Sliding window size for leave-one-out site-specific placement bias algorithm only effective when used in combination with -f S

`DEFAULT: 100 sites`

Defines the sliding window size for the -f S option, 100 usually yields relatively smooth plots.

Example: `raxmlHPC -f S -s alg -t tree -m GTRGAMMA -N 100 -W 200 -n TEST`

`-x` Specify an integer number (random seed) and turn on rapid bootstrapping
CAUTION: unlike in previous versions of RAxML will conduct rapid BS replicates under the model of rate heterogeneity you specified via -m and not by default under CAT

This will invoke the rapid bootstrapping algorithm described in http://sysbio.oxfordjournals.org/content/57/5/758.short

Example: `raxmlHPC -x 12345 -p 12345 -# 100 -m GTRCAT -s alg -n TEST`

You can also combine this with the bootstopping option (bootstrap convergence criterion), e.g.:

Example: `raxmlHPC -x 12345 -p 12345 -# AUTO_MRE -m GTRCAT -s alg -n TEST`

or also have RAxML search for the best-scoring ML tree after the bootstrap searches using the `-f a` option:

Example: `raxmlHPC -x 12345 -p 12345 -# AUTO_MRE -m GTRCAT -s alg -f a -n TEST`

```
-X    Same as the "-y" option below, however the parsimony search is more
      superficial.
      RAxML will only do a randomized stepwise addition order parsimony tree
      reconstruction without performing any additional SPRs.
      This may be helpful for very broad whole-genome datasets, since this can
      generate topologically more different starting trees.

      DEFAULT: OFF
```

Example: `raxmlHPC -X -m GTRCAT -s alg -p 12345 -n TEST`

```
-y    If you want to only compute a parsimony starting tree with RAxML specify -y,
      the program will exit after computation of the starting tree

      DEFAULT: OFF
```

If you want to only compute a randomized parsimony starting tree with RAxML and not execute an ML analysis of the tree specify `-y`. The program will exit after computation of the starting tree.

Example: `raxmlHPC -y -m GTRCAT -s alg -p 12345 -n TEST`

```
-Y    Pass a quartet grouping file name defining four groups from which to draw
      quartets
      The file input format must contain 4 groups in the following form:
      (Chicken, Human, Loach), (Cow, Carp), (Mouse, Rat, Seal), (Whale, Frog);
```
`Only works in combination with -f q !`

The example below, will randomly draw 100 quartets from the predifined quartet grouping above and evaluate their likelihood.

Example: `raxmlHPC -f q -m GTRGAMMA -t tree -Y quartetFile -n 100 -n TEST`

```
-z    Specify the file name of a file containing multiple trees e.g. from a
      bootstrap that shall be used to draw bipartition values onto a tree provided
      with -t.
      It can also be used, for instance to compute per site log likelihoods in
      combination with -f g and to read a bunch of trees for a couple of other
      options (-f h, -f m, -f n).
```

This option is required in combination with a lot of other RAxML options, in essence every time you need to read in a bunch of trees. Below is an example where you use the option to draw BS support values on a best-knwon ML tree.

Example: `raxmlHPC -f b -m GTRCAT -t mlTree -z bootstrapTrees -n TEST`

```
-#|-N Specify the number of alternative runs on distinct starting trees
      In combination with the "-b" option, this will invoke a multiple boostrap
      analysis
      Note that "-N" has been added as an alternative since -# sometimes caused
      problems with certain MPI job submission systems, since -# is often used
      to start comments.
      If you want to use the bootstopping criteria specify -# autoMR or -#
      autoMRE or -# autoMRE_IGN for the majority-rule tree based criteria (see -I
      option) or -# autoFC for the frequency-based criterion.
      Bootstopping will only work in combination with -x or -b

      DEFAULT: 1 single analysis
```

Specifies the number of alternative runs on distinct starting trees, e.g., if `-# 10` or `-N 10` is specified, RAxML will compute 10 distinct ML trees starting from 10 distinct randomized maximum parsimony starting trees.
In combination with the `-b` option, this will invoke a multiple bootstrap analysis. In combination with `-x` this will invoke a rapid BS analysis and combined with `-f a -x` a rapid BS search and thereafter a thorough ML search on the original alignment.
We introduced `-N` as an alternative to `-#` since the special character `#` seems to sometimes cause problems with certain batch job submission systems.
In combination with `-f j` this will generate numberOfRuns bootstrapped alignment files.

Note that, several other program options such as the quartet evaluation algorithm (`-f q` option) or the site-specific placement bias algorithm (`-f S` option) require this option to be specified as well. If you forgot to specify this option, RAxML will tell you that you need to do so.

The example below will do 20 independent ML tree searches on 20 randomized stepwise addition order parsimony tree.

Example: `raxmlHPC -p 12345 -s alg -n TEST -m GTRGAMMA -# 20`

Computing TC and IC values

In the following I provide some more detailed examples for computing the IC and TC metrics from Salichos and Rokas 2013 http://www.ncbi.nlm.nih.gov/pubmed/23657258

Given a set of gene trees, RAxML can directly calculate a majority rule consensus (MR in RAxML terminology) as well as an extended majority rule consensus tree (MRE in RAxML terminology) on this set that has every internode (that is, internal branch) annotated by their respective IC and ICA scores.

For instance, to compute the IC, ICA, TC, and TCA scores for a given set of gene trees on a MRC tree you would type:

```
raxmlHPC -L MR -z 1070_yeast_genetrees.tre -m GTRCAT -n T1
```

where
`-L MR` specifies that the scores will be displayed on a MR tree that is computed by RAxML
`-z 1070_yeast_genetrees.tre` specifies the filename that contains the set of gene trees (which are the maximum likelihood trees from the 1,070 yeast genes analyzed by Salichos, and

Rokas 2013, and which are provided as supplementary data to this manuscript)
`-m GTRCAT` is an arbitrary substitution model (this will have no effect whatsoever, but is required as input to RAxML)
`-n T1` is the run ID that is appended to output files.

RAxML will automatically build the MR tree, annotate it with the IC and ICA scores, and report both in an output file named `RAxML_MajorityRuleConsensusTree_IC.T1`, which will look like this:

```
(Scer,Spar,(Smik,(Skud,(Sbay,(Scas,(Cgla,(Kpol,(Zrou,((Clus,((Psti,((Ctro,
(Calb,Cdub):1.0[0.95,0.95]):1.0[0.77,0.77],
(Cpar,Lelo):1.0[0.76,0.76]):1.0[0.75,0.75]):1.0[0.11,0.11],
(Cgui,Dhan):1.0[0.02,0.07]):1.0[0.02,0.08]):1.0[0.97,0.97],((Sklu,
(Kwal,Kthe):1.0[0.97,0.97]):1.0[0.32,0.23],
(Agos,Klac):1.0[0.08,0.08]):1.0[0.04,0.10]):1.0[0.59,0.47]):1.0[0.02,0.02]):1.0[0.
11,0.11]):1.0[0.02,0.02]):1.0[0.97,0.97]):1.0[0.05,0.14]):1.0[0.30,0.27]):1.0[0.54
,0.54]);
```

For each internode or internal branch of the constructed MR tree, RAxML will assign an `length[x,y]` branch label, where `length` corresponds to the branch length (because this is a MRE tree, all internal branch lengths have been arbitrarily set to `1.0` by default), `x` corresponds to the IC score and `y` to the ICA score.

RAxML will also calculate the TC and TCA scores for the MR tree, as well as the relative TC and TCA scores that are normalized by the maximum possible TC and TCA scores for a fully bifurcating tree from the same number of taxa.

The scores are displayed in the terminal output and in the `RAxML_info.runID` standard output file associated with the run (in this case `RAxML_info.T1`) and will look like this:

```
Tree certainty for this tree: 7.642240
Relative tree certainty for this tree: 0.382112

Tree certainty including all conflicting bipartitions (TCA) for this tree: 7.580023
Relative tree certainty including all conflicting bipartitions (TCA) for this tree: 0.379001
```

Given a set of gene trees, RAxML can also directly calculate an extended MR tree on this set that has every internode (that is, internal branch) annotated by their respective IC and ICA scores.
The particularly compute-intensive inference of extended MRC trees (finding the optimal extended MRC tree is, in fact, NP-hard; Phillips, and Warnow 1996) relies on RAxML's fast parallel implementation. Thus if you use the PThreads version of RAxML, this part will run in parallel.
To compute IC, ICA, TC, and TCA scores on an extended MR tree you would type:

```
raxmlHPC -L MRE -z 1070_yeast_genetrees.tre -m GTRCAT -n T2
```

RAxML can compute MR and extended MR trees, using both fully bifurcating and partially resolved/multifurcating trees as an input. RAxML can also compute stricter MR trees with arbitrary threshold settings that range between 51 and 100%. For instance, by typing

```
raxmlHPC -L T_75 -z 1070_yeast_genetrees.tre -m GTRCAT -n T3
```

RAxML will display IC, ICA, TC and TCA scores on a MR tree that only includes those bipartitions that have ≥ 75% support.

We have also implemented an option (`-f i`) that allows the user to calculate and display IC, ICA, TC, and TCA scores onto a given, strictly bifurcating reference tree (for example, the best-known ML tree).
This is analogous to the standard `-f b` option in RAxML that draws bootstrap support values from a set of bootstrap trees onto a reference phylogeny. The option can be invoked by typing

```
raxmlHPC -f i -t yeast_concatenationtree.tre -z 1070_yeast_genetrees.tre -m GTRCAT -n T4
```

Note that, the tree contained in file `yeast_concatenationtree.tre` needs to be strictly bifurcating and contain branch lengths. In this example, the `yeast_concatenationtree.tre` file is the best-known maximum likelihood tree recovered by concatenation analysis of the 1,070 yeast genes from the paper.
Using this command, RAxML will annotate the tree in `yeast_concatenationtree.tre` with the IC and ICA scores, and report both in an output file named `RAxML_IC_Score_BranchLabels.T4`, which will look like this:

```
(((((((Clus:0.47168135428609103688,
((((Lelo:0.30356174702769450624,Cpar:0.25490874239480920682):0.1302317827585764975
5[0.76,0.76],(Ctro:0.18383414558272206940,
(Calb:0.04124660275465741321,Cdub:0.04290801588396832289):0.14526604486383792869[0
.95,0.95]):0.12355825028654655873[0.77,0.77]):0.17335821030783615804[0.75,0.75],Ps
ti:0.42255112174261910685):0.07862882822310976461[0.11,0.11],
(Cgui:0.45961028886034632768,Dhan:0.28259245937168109286):0.05586015476156453580[0
.02,0.07]):0.08116340505230199009[0.02,0.08]):1.03598510402913923656[0.97,0.97],
((Agos:0.53332956655591512440,Klac:0.47072785596320687596):0.08132006357704427146[
0.08,0.08],
((Kthe:0.17123899487739652203,Kwal:0.17320923240031221857):0.25620117495110567019[
0.97,0.97],Sklu:0.24833228915799765435):0.05646992617871094550[0.32,0.23]):0.05236
306187235122145[0.04,0.10]):0.10686517691208799463[0.59,0.47],Zrou:0.4130783368556
3782877):0.03792570537296727218[0.02,0.02],Kpol:0.43287284049576529865):0.04560341
693136910068[0.11,0.11],Cgla:0.49584136365135367264):0.04363310339731014259[0.02,0
.02],Scas:0.37212829744050218705):0.29362133996280515014[0.97,0.97],
(Skud:0.06926467973344750673,(Smik:0.06535810850036427588,
(Scer:0.04285848856634000975,Spar:0.03030513540244994877):0.02506719066056842596[0
.54,0.54]):0.02459323291555862850[0.30,0.27]):0.02524223867026276907[0.05,0.14],Sb
ay:0.06506923220637816918);
```

For each internode or internal branch of this output tree RAxML will assign a `length[x,y]` branch label, where `length` corresponds to the original branch length in the input tree passed via `-t`, `x` corresponds to the IC score and `y` to the ICA score. RAxML will also display the TC and TCA scores of this tree both in the terminal output and in the `RAxML_info.T4` output file associated with the run.

It should further be noted that the IC and ICA scores are represented as branch labels, since, as is the case for bootstrap support values, information associated to internodes/splits/bipartitions of a tree always refers to branches and not nodes.
Each tree viewer (e.g., Dendroscope http://ab.inf.uni-tuebingen.de/software/dendroscope/) that can properly parse the Newick tree format is able to display these branch labels.
The rationale for not providing IC and ICA scores as node labels is that, some viewers may not properly rotate the node labels when the tree is re-rooted by the user, which will lead to an erroneous branch- to-IC- and branch-to-ICA score association.

When calculating IC and ICA scores on extended MR trees or when drawing IC and ICA scores onto a given reference tree it may occur that the bipartition that has been included in the tree has lower support than one or more conflicting bipartitions. In this case, RAxML will report IC and ICA scores on the inferred tree with negative signs.

Finally, we have implemented a verbose output option that allows users to further scrutinize particularly interesting conflicting bipartitions.
Verbose mode is activated by adding the `-C` command line switch to any of the above examples. In verbose mode RAxML will generate two additional types of output files: One set of files containing one included bipartition and the corresponding conflicting bipartitions in Newick format

(called `RAxML_verboseIC.runID.0 … RAxML_verboseIC.runID.N-1`, where `N` is the number of bipartitions in the tree) and an output file that lists all bipartitions (included and conflicting) in a PHYLIP-like format (called `RAxML_verboseSplits.runID`).

For example, by adding `-C` to the previous command

```
raxmlHPC -f i -t yeast_concatenationtree.tre -z 1070_yeast_genetrees.tre -m GTRCAT -n T5 -C
```

will produce 20 files (one for each of the 20 bipartitions present in the `yeast_concatenationtree.tre`) named `RAxML_verboseIC.T5.0, RAxML_verboseIC.T5.1, …, RAxML_verboseIC.T5.19`

For example, the `RAxML_verboseIC.T5.0` file will look like this:

```
((Cpar, Lelo),(Scer, Smik, Skud, Cgla, Kpol, Zrou, Kwal, Kthe, Agos, Klac, Clus,
Cgui, Psti, Ctro, Calb, Cdub, Dhan, Sklu, Scas, Sbay, Spar));
((Cpar, Ctro, Calb, Cdub),(Scer, Smik, Skud, Cgla, Kpol, Zrou, Kwal, Kthe, Agos,
Klac, Clus, Cgui, Psti, Lelo, Dhan, Sklu, Scas, Sbay, Spar));
```

where the first Newick string represents the bipartition that was included in the `yeast_concatenationtree.tre` and all following Newick strings represent the corresponding conflicting bipartitions in descending order of their frequency of occurrence In the case of the `RAxML_verboseIC.T5.0` file the first bipartition, which is included in the `yeast_concatenationtree.tre` conflicts with only one other bipartition, which is listed as the second bipartition.

Analogously, the output file that lists all bipartitions (included and conflicting) in a PHYLIP-like format (`RAxML_verboseSplits.T5`), looks like this:

```
1. Scer
2. Smik
3. Skud
4. Cgla
5. Kpol
6. Zrou
7. Kwal
8. Kthe
9. Agos
10. Klac
11. Clus
12. Cgui
13. Psti
14. Cpar
15. Lelo
16. Ctro
17. Calb
18. Cdub
19. Dhan
20. Sklu
21. Scas
22. Sbay
23. Spar
partition:
----- ----- ---** ----- ---     956/89.345794/0.761406
----- ----- ---*- ***-- ---     39/3.644860/0.761406

partition:
----- ----- ----- -**-- ---     1051/98.224299/0.949483
```

```
----- ----- ----- **--- ---     6/0.560748/0.949483

.
.
.
partition:
--*** ***** ***** ***** **-     641/59.906542/0.303620
-**-- ----- ----- ----- -*-     148/13.831776/0.303620
-*-** ***** ***** ***** *--     114/10.654206/0.303620

partition:
-**** ***** ***** ***** **-     825/77.102804/0.545775
-**-- ----- ----- ----- -**     87/8.130841/0.545775
```

Here each block that starts with the partition keyword contains a specific bipartition and all corresponding conflicting bipartitions in descending order.
The `x/y/z` scores correspond to the frequency of the bipartition (`x`), the support percentage (also known as gene support frequency; `y`), and the IC score (`z`).

IV. Citing RAxML

When using RAxML please cite: TODO