July 2023 Current version: 4.1

# kSNP4.1 Users Guide

Barry G. Hall, Bellingham Research Institute
Jeremiah Nisbet, Bellingham Research Institute

Download either the **kSNP4.1 Mac Package** zip file or the **kSNP4.1 Linux Package** zip file from SourceForge https://sourceforge.net/projects/ksnp/files/. Each contains a **kSNP4.1pkg** directory (folder) and a Documentation directory that includes this User Guide. Also, separately, download the Example.zip file, which contains input files for the examples discussed in section VIII. Installation of ***kSNP4.1*** is discussed in section II.

## Table of Contents

kSNP4.1 is a command-line program to compare genomes sequences without aligning them. If you are unfamiliar with the command-line interface please read the document "A command-line primer.pdf" that is in the Documentation directory.

# I. The kSNP4.1 program

Single nucleotide polymorphisms (SNPs) are important for phylogenetic analysis, for tracking viral and bacterial pathogens during outbreaks, and for correlating genotype with phenotype. When dealing with gene *sequences* the first step in identifying SNPs is multiple alignment of the gene sequences.

Bacterial and viral genomes are characterized by multiple and massive insertion-deletions (indels), inversions, and transpositions of blocks of DNA from one part to another part of the genome. The term "pan-genome" has been coined to describe the collective content of the genomes of a bacterial species. The pan-genome consists of core genes, those genes that are present in all members of a species, and accessory genes, those that are present in some, but not all, members of the species. Indeed, there is at least as much variation in the presence/absence of DNA sequences as there is variation in SNPs. That variation makes multiple sequence alignment of complete genomes impractical for large numbers of genomes.

kSNP4.1 is a program that identifies the pan-genome SNPs in a set of genome sequences, and estimates phylogenetic trees based upon those SNPs. Because there are many potential downstream applications of SNP information, kSNP4.1.1 also provides output files that include a multiple alignment of all of the SNP positions and files that provide information about the positions of each SNP in each genome and annotations. kSNP4.1 can analyze both complete (finished) genomes and unfinished genomes in assembled contigs or raw, unassembled reads. Finished and unfinished genomes can be analyzed together, and kSNP4.1 can automatically download Genbank files of the finished genomes (and annotated genome assemblies) and incorporate the information in those files into the SNP annotation. Core SNPs, those present in all of the genomes, can optionally be analyzed separately to generate a multiple SNP alignment and trees based on only the core SNPs. kSNP4.1.1 also can provide a separate analysis of those SNPS that occur in at least a user-determined fraction of the genomes.

kSNP4.1 is provided as both a Linux package for 64-bit CPUs and as a Mac OS package for 64-bit CPUs. The packages contain the kSNP4.1 program and this documentation. In addition there are two example input data sets that are used as the basis for the discussion of the output files in Section VIII. Those data sets must be downloaded separately from SourceForge.

kSNP4.1 incorporates five third-party programs. Please see Section XII, Citations, for the list of publications describing these software packages.

kSNP4.1 is a command-line program that must be run from the Terminal application. If you are unfamiliar with the command-line interface please read the document "A command-line primer.pdf " before attempting to install or use kSNP4.1.

# II. Installing kSNP4.1

A note about terminology: Linux users use the term "directory" while Mac users use the term "folder" to mean exactly the same thing. Throughout this User Guide we will use the term "directory". Mac users, when you see "directory" think "folder".

kSNP4.1 consists of a Bash shell script (kSNP4) and a set of executables including five separate third-party programs to accomplish the kSNP4.1 goals. kSNP4.1 is provided as separate packages for Mac OS X and for Linux operating systems. Whichever version you chose was downloaded as a zip archive and should automatically self-extract. If it does not, double-click the file to extract it. In the resulting package directory you will see a directory named kSNP4.1pkg that contains 45 files. The kSNP4.1pkg directory contains a bash script named kSNP4 that directs the execution of a set of Unix executable files built from perl and python scripts, including the 3rd party programs jellyfish, FastTreeMP, parsimonator, mummer and consense. You do not have to deal with most of those files directly, they are called by kSNP4 and the executables that it controls.

This does mean that although you will have installed the kSNP4.1pkg directory (folder) when you actually run kSNP4.1 the command line will always begin with `kSNP4`.

You will have downloaded either the kSNP4.1 Linux package or the kSNP4.1 Mac package from SourceForge. Each package contains a Documentation directory and a kSNP4pkg directory.

In Terminal navigate to the package directory that you downloaded. You will install the kSNP4.1pkg directory into the /usr/local directory.

kSNP4.1pkg should be installed into the /usr/local directory, and once installed the kSNP4.1pkg directory should not be moved afterwards.

The `/usr/local` directory is protected from accidental or malicious changes by the operating system so you need to make some extra effort to get access to it and to add things to it.

In Terminal, from within the package that you downloaded, enter:

```
sudo cp -R kSNP4.1pkg /usr/local.
```

You will be asked for a password. Enter your administrator's password. If you don't know the password ask the System Administrator for help. To check that the package was actually moved to /usr/local enter `cd`

`/usr/local`, then enter `ls` to list the contents of /usr/local. You should see kSNP4.1pkg listed. If you do not, try again or get help from your computer's Administrator.

## Set the PATH variable

For you to be able to use kSNP4.1, the operating system needs to be able to find all of the programs in kSNP4.1pkg when you type `kSNP4` on the command line. To do that you must set the PATH environmental variable to include the path of the kSNP4.1pkg installation (`/usr/local/kSNP4.1pkg`). A path is just a description of the nesting of directories starting with the root directory (referenced as just /), with the directory names separated by /. The path `/usr/local` means the `local` directory that is in the `usr` directory that is at the root. The PATH variable usually includes paths to several default directories where unix programs may be located, and you can add paths to other directories where you have installed other unix programs. To see where the operating system currently looks for programs you try to run, enter `echo $PATH.`

If the path variable has never been modified you will see something like this (there may be other paths as well):
`/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin`
The different paths are separated by colons (:).

To add paths you must modify a file that sets environmental variables. The file is found in your home directory Under the bash shell, depending on your OS and its age, that file is called .bash_profile or .bashrc and it is found in your home directory. Under the zsh shell (the default shell in some recent versions of both Mac OS and Linux operating systems) the file that holds environmental variables is called .zprofile, also found in your home directory. *In both cases the file is invisible because its name begins with a dot.*

In your home directory in **Linux** type control-H, or under the **View** menu choose **Show hidden files**. In **Mac OS X** type Command-shift-period. You should now see the hidden files.

The instructions below assume that you installed kSNP4.1pkg into `/usr/local`. If you installed kSNP4.1pkg elsewhere add the path to the installed location instead.

Enter the following line *exactly as given here; i.e. no spaces around the = sign next to the text PATH.*
`export PATH="/usr/local/kSNP4.1pkg:$PATH"`

Save the file in your home directory, then close the Terminal window. A Terminal window only becomes aware of the available paths when it is first opened.
If you already had kSNP3s or kSNP4 installed you will see the export PATH line containing kSNP3 or kSNP4. Just change it to kSNP4.1 and you are done.

## What computer should you use?

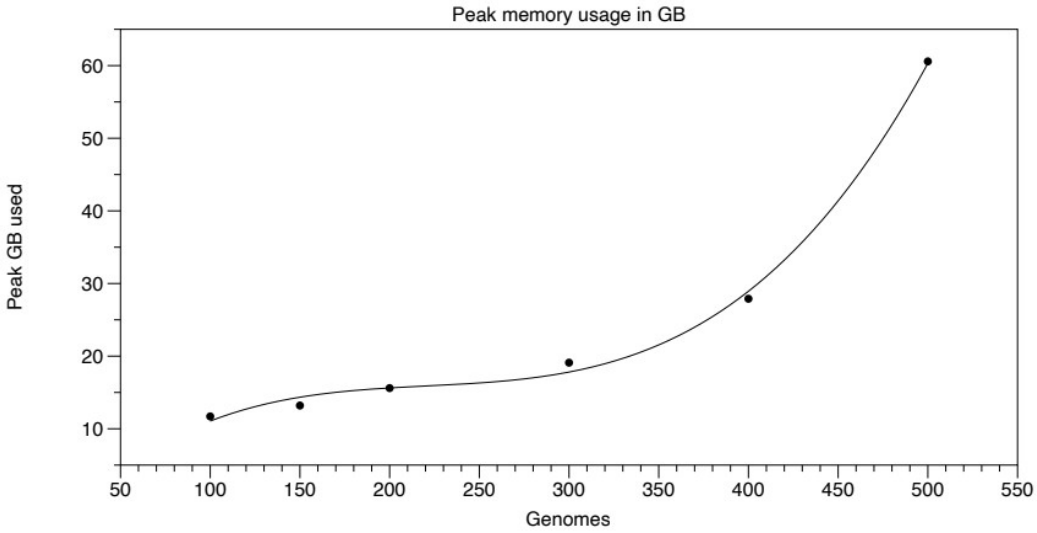
Either a Linux or a Macintosh computer will work, but it is important to use a computer that is sufficiently fast, that has sufficient RAM (memory), and that has sufficient disk space. You should use the most powerful computer that you can obtain. We recommend at least 32 GB of RAM, and a CPU with at least 4 cores if you expect to analyze hundreds of bacterial genomes. Fewer genomes or viral genomes will require less RAM. During execution kSNP4.1 writes thousands of temporary files, so you should have at least 100 GB of free disk

space to classify significant numbers of genomes. For a large data set of 400 *E. coli* genomes the temporary files amounted to 807 GB. Those temporary files are deleted automatically at the end of the run, but if you are using very large data sets be sure that you have sufficient free space to accommodate the files that are written during the run. For genomes of roughly 5 MB a very rough guide is that you need to have about 2 GB free space per genome.

BGH uses a dedicated laptop running Ubuntu 22.04 Linux with a 16 core CPU, 1 TB of disk storage and 32 GB of RAM. He regrets being cheap and not getting 64 GB of RAM

### Memory Management

When using kSNP4.1 to compare large numbers of genomes, say over 150 bacterial genomes, kSNP4.1 can use up 15 GB of RAM resulting in premature termination of kSNP4.1 if all you have is 16 GB.
The graph below shows peak memory usage by kSNP4.1 vs number of *E. coi* genomes, which average about 5.2 MB per genome.




Memory usage is a function of the cube of number of genomes. Where X is number of genomes peak memory usage in GB = $1.885 * 10^{-6} * X^3 - 0.001.5018 * X^2 + 0.28863 * X - 7.18221$, with $R^2 = 0.9974$. You can use that equation to estimate whether you have sufficient RAM for your data set.

#### *Mac OS*

Memory management is not a problem on Mac because it automatically manages memory

#### *Linux operating systems*

For Linux we suggest setting up a swapfile equal to twice the installed RAM. SWAP is a partition of disk space that can be used a virtual memory when real memory starts to run out. This site https://www.ubuntupit.com/how-to-add-and-configure-swap-space-on-ubuntu-linux/ shows you how to increase that space.

## III. Obtaining the genome sequence files

kSNP4.1 works on genome sequence files in the fasta format. Those files must be stored on your hard drive in directories. It is important to put the files into appropriate directories and not to move them after making the input file for kSNP4.1 (more about that in the next section). Barry Hall (BGH) organizes his genome sequence files in a directory named Genomes, within which is a directory named Bacteria and another directory named

Viruses. Within each of those directories he has a directory for each species; e.g. E. coli, Salmonella, etc. You can use any organization that you wish, but it is a good idea to be consistent.

### Organizing your genome files

We suggest collecting all fasta files for a species into a single directory that is named for the species and *putting that directory somewhere from which it will never be moved*. The best strategy might be to have a directory named, for instance, "Genomes" Within that you might have a directory named "E_coli", another named "S_aureus", etc. If you wanted to keep annotated genomes separate from unannotated genomes, within each species directory you could have directories "Annotated" and "Unannotated".

If you move a genome file or an enclosing directory then the paths in any existing kSNP4.1 input file will be wrong and kSNP4.1 won't be able to find the genome files if you try to run kSNP4.1 on that dataset again - nothing will work. You would have to reconstruct every affected input file. Not only must you never move the individual directories or the "Genomes" directory *you must never move any of the directories that enclose those directories!!*

The solution is to put the Genomes directory into your home directory and to leave it alone. Do ***not*** put your Genomes directory onto another Volume, such as an external hard drive. kSNP4.1 won't be able to find the files if the external hard drive is removed or renamed or if the Volume is inaccessible for any reason.

A common application of kSNP4.1 is to compare a set of genome sequences that one has made from a collection of isolates of interest with known genome sequences that are in NCBI (National Center for Biotechnology Information) databases. Often that comparison involves making a phylogeny that includes both NCBI's genomes and the lab genomes.

It can be very tedious to collect all the genome sequence files from NCBI, so I have created three programs to assist with finding and downloading genome sequence files: ***parseNCBIcsv, parseViralNCBIcsv*** and ***getFastaGenomes***. Their use is detailed in Section XII "Downloading genome sequences from NCBI ". All three programs are in the kSNP4pkg folder that you installed.

## IV. Input files

All of the programs in the kSNP4.1 package require input files. Those input files are simple text files. Word processor files written by Microsoft Word, WordPerfect, etc. will not create compatible files. Input files should be opened in a text editor. See Appendix I for suggested text editors.

### File names

It is important that the input files are correctly named. Depending on your operating system, file names that violate the rules below may cause horrible results - including greatly inflating the number of SNPs, or failing to annotate the SNPs correctly. kSNP4.1 may appear to have run properly and unless you have had considerable experience with similar data sets you may not recognize that errors have occurred. To avoid that situation kSNP4.1 and later versions check the input file for file names that violate the rules and abort if any files are incorrectly named.

A file name consists of two parts: a file ID and an extension. The extension is separated from the file ID by a dot. For the file named **Escherichia_coli_042.fasta**, the file ID is Escherichia_coli_042 and the extension is fasta.

The rules for naming files are:

1 The file name cannot include more than one dot (.); i.e. only the dot separating the file ID from the extension is allowed. Escherichia_coli_042.fasta is legal; Escherichia_coli_0.42.fasta is not. (It is probably legal for your operating system, but it is not legal for kSNP4.1).
2 The file name may not contain any spaces. Escherichia_coli_042.fasta is legal; Escherichia coli 042.fasta is not.
3 The file name can only contain the characters A-Z a-z 0-9 . - (the dash or minus character) and _ (the underscore character). Characters such as : *() > are illegal. When necessary for readability replace spaces, colons etc. with the underscore character or use internal capitalization. For instance Ecoli_O157H7 instead of Ecoli O157:H7.

If kSNP4.1 finds a file with an illegal character it terminates the run and writes a file named NameErrors.txt. NameErrors.txt reminds you of the naming rules and writes a list of the files that contain naming errors. Part of that file might look like this:

```
Line 2: CFSA 003.fasta
Line 3: CFSA:004.fasta
```

Each error includes the line in the input file where the error occurred and the illegal file name. On Line 2 the file name is illegal because name includes a space. On Line 3 the file name is illegal because it includes a colon.

Correct the illegal file names on the files themselves and in the input file!!! Once that is done delete the NameErrors.txt file. That is essential because one way that kSNP4.1 knows to terminate the run is by looking for a file named NameErrors.txt. If it finds that file, even if it is a left-over file, kSNP4.1 will terminate the run.

### Directory (folder) names

Be sure that there are no spaces in the names of any of the directories in the path from kSNP4.1pkg to the directory where the input file is located. Spaces in path names will result in an error `[48632] Failed to execute script 'Kchooser4' due to unhandled exception!` when running Kchooser or kSNP4.

## V. Making the input file for kSNP4.1

A data set consists of a set of a set of genome sequences in fasta format, one file per genome. **Note! Raw-read files in the fastq format will not work. See the FAQ about Fastq in the FAQ section)** The genome can be a finished (closed) sequence, multiple chromosomes and plasmids, an assembly of multiple contigs, or raw, unassembled reads.

The input file is a list that gives the *path* to each sequence file containing a genome and a name for that genome, with each genome on a new line. The input list tells kSNP4.1 where to find each genome sequence file.

The input file might look like this:

File Path : ~/GenomeSequences/Eco_Shig/Eco_Shig_Finished/EcoShig_inlist

EcoShig_inlist

```
1  /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli_042.fasta    Escherichia_coli_042
2  /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli_536.fasta    Escherichia_coli_536
3  /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli_53638.fasta Escherichia_coli_53638
4  /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli_55989.fasta Escherichia_coli_55989
5  /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli__BL21_Gold.fasta    Escherichia_coli__BL21_Gold
6  /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli__clone_D_i14_.fasta Escherichia_coli__clone_D_i14_
7  /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli__clone_D_i2_.fasta  Escherichia_coli__clone_D_i2_
8  /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli_ABU_83972.fasta Escherichia_coli_ABU_83972
9  /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli_APEC_O1.fasta   Escherichia_coli_APEC_O1
10 /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli_APEC_O78.fasta  Escherichia_coli_APEC_O78
11 /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli_ATCC_8739.fasta Escherichia_coli_ATCC_8739
12 /Users/barry/GenomeSequences/Eco_Shig/Eco_Shig_Finished/FAstaFiles/Escherichia_coli_B_REL606.fasta  Escherichia_coli_B_REL606
```

Each line consists of a path separated by a tab from the genome ID. The genome ID will be used in all of the trees and other output files. The genome ID on the first line is Escherichia_coli_042, while the *file name* is Escherichia_coli_042.fasta.

The easiest way to create the input file is to use the program ***MakeKSNP4infile*** in either the fully automatic mode (default) or the semi-automatic mode.

Fully automatic method

Navigate to the directory that *encloses* the directory containing the fasta genome files and enter `MakeKSNP4infile -indir myDir -outfile myInfile`. myDir is the name of the directory that contains the fasta files. myInfile is the name for the kSNP4.1 input file that will be written. The genome ID for each file is generated from the file name (any extensions, such as .fasta, are dropped).
Note: The genome ID is the taxon label that will appear on trees written by ***kSNP4.1***.

Semi-automatic method

Navigate to the directory that *encloses* the directory containing the fasta files and enter `MakeKSNP4infile -indir myDir -outfile myInfile S`. myDir is the name of the directory that contains the fasta files. myInfile is the name for the kSNP4.1 input file that will be written, and S indicates the semi-automated process in which the paths to each of the files are written followed by a tab, but you must manually enter the genome ID. This method is useful when the names of the fasta genomes are too long to fit easily onto a drawing a tree.

Regardless of the method used to generate the file you can subsequently edit the genome ID in the input file at any time.

Manually

In a text file enter the path to the genome file, press Tab, then type the name for the genome. When there are many genomes this can be a tedious process.

You certainly don't want to have to figure out and type the entire path (`/Users/barry/Desktop/Test_kSNP3/Genomes/Escherichia_coli_042.fasta`), but there is an easier way to enter the path depending upon your operating system:

In Mac OS X hold down the Command key and drag the file into the text document. The path will a appear.

In Ubuntu Linux (and several other Linux versions, we understand) select the file, right-click and in the resulting menu choose Copy. Paste into the text document and the path will automatically appear.

For other versions of Linux you can enter `cd` , then drag the file into the terminal window. Don't forget the space after `cd`.

## VI Using Kchooser4 to determine optimal value of k, the kmer size

kSNP4.1 requires that you enter the kmer size on the command line using the -k option. Kmer size, *which must be an odd number*, defines the length of the oligonucleotides (kmers) that kSNP4.1 identifies in all of the sequences. Oligos that are identical between different genomes at all but the central base (odd-length kmers are used so that there is a central base) are taken as being homologous, i.e. a SNP locus. If the kmer size is set too low, say to a value of 5 bp, then there will be many kmers with a central base difference that are identical by chance alone within a genome and between genomes, rather than being identical by descent from a common ancestor; i.e. homologous. If the target genomes are short, then the chance of spuriously identical kmers is reduced; whereas long genomes increase the chance of spurious matches. That consideration would seem to favor choosing large values for kmer size. However, if kmer size is set too high, say 51 bp, then many SNPs will be missed because of sequence variation at multiple sites within the kmer. That is because a SNP locus is defined by the *conserved* sequence surrounding the central base of the kmer. When there is little base-pair variation large values decrease the chance of spurious matches, but when there is much variation large values decrease the sensitivity of SNP detection. The program ***Kchooser4*** will do an initial review of your particular data set to identify the optimum kmer size to use.

***Kchooser4*** first identifies an optimum value of k for your specific data set, then it evaluates the extent of sequence variation to provide insight into the efficiency with which kSNP4.1 is likely to identify SNPs from your data set and the accuracy of parsimony trees at that optimum k.

To estimate the optimum k, ***Kchooser4*** begins with a k value that is a function of the median genome size. It determines FUK, the fraction of unique kmers, i.e. those that occur only once in the median-length genome at that value of k. Some kmers will occur more than once because of genuine duplicated regions, so ***Kchooser4*** continues seeking a value of k such that at least 0.99 of kmers are unique. If FUK is < 0.99 it increments k by 2 and tries again. If the fraction is ≥ 0.99 it reports that as the optimum value of k. The expectation is that less than 1% of a real genome will be duplicated.

In some cases more than 1% of the genome is duplicated, so ***Kchooser4*** would continue increasing k forever. However, if more than 1% of the genome is duplicated, FUK approaches the fraction of the genome that is actually unique. When FUK increases by <0.001 between successive trials, ***Kchooser4*** reports the current value of k as the optimum value.

***Kchooser4*** then determines what fraction of the kmers from the shortest sequence are present in all of the genomes; i.e. the fraction of core kmers (FCK). As sequence variation increases FCK decreases and the efficiency with which kSNP4.1 detects SNPs decreases. Simulation studies (Gardner & Hall 2013 PLoS One **8**(12): e81760. doi:10.1371/journal.pone.0081760, Hall 2015 Cladistics **32:** 90-99) have shown that SNP detection efficiency is >97%, and the accuracy of parsimony trees is > 0.97 when the fraction of core kmers reported by ***Kchooser4*** is ≥ 0.1.

When ***Kchooser4*** is complete it writes a report file whose name is generated from the name of the input file by prepending “Kchooser4_” and replacing the “.in” extension with “.report”.
The input to ***Kchooser4*** is the input file for kSNP4.1 that you created in Section IV using the program ***MakeKSNP4infile***.

Usage: `Kchooser4 -in infileName`

Example: `Kchooser4 -in Eco50.in.`

Output file: Kchooser4_Eco50.report.

Example output file contents:

```
Initial value of K is 13
When k is 13 0.8360598262862955 of the kmers from the median length sequence are unique.
When k is 15 0.9669400995593544 of the kmers from the median length sequence are unique.
When k is 17 0.9817466241297378 of the kmers from the median length sequence are unique.
When k is 19 0.9835874469180003 of the kmers from the median length sequence are unique.
When k is 21 0.9841848784478533 of the kmers from the median length sequence are unique.
The optimum value of k is 21

There were 50 genomes.
The median length genome was Eco_D8
Its length is 5130646
The time used to determine the optimum k was 38.58 seconds.
The shortest genomes is Eco_CAR its length is 4544646

From a sample of 1000 unique kmers 463 are core kMers.
FCK = 0.463.
The total time used was 58 seconds
```

In this example, the fraction of unique kmers reached a plateau at a bit over 0.98, suggesting that a little more than 1% of that particular genome is duplicated. Increasing k from 19 to 21 increased FUK by < 0.001, so ***Kchooser4*** reports an optimum k of 21.

***Kchooser4*** runs fairly quickly. For a data set consisting of 400 *E. coli* genomes ***Kchooser4*** required only 62 seconds on my system. The inclusion of genomes as raw reads can dramatically increase the time required by ***Kchooser4***. To speed up the process raw read genomes should be removed from the input file before running ***Kchooser4***.

## VII. Running kSNP4.1

kSNP4.1 is run from the command line within the Terminal program.

kSNP4.1 should be run from within the directory that contains the input file. That can be any directory that is convenient for you but it should ***not*** be the kSNP4.1 package directory that you downloaded, nor should it be located within /usr/local.

Enter the program name `kSNP4` followed by a series of *arguments* separated by spaces. Arguments are instructions that tell ***kSNP4*** exactly what to do. An argument consists of a flag and a value, for instance the argument `-in in_list` says that the input file is named in_list. There must always be a space between the flag `-in` and the value in_list, and there must always be a space between an argument and the next argument. The table below lists the arguments and what they mean. Arguments may be entered in any order.

Table 1 Command line arguments*

| Argument | Flag | Value | Example | Comment |
|---|---|---|---|---|
| Input file listing paths | -in | file name | `-in infileName` | Required. |

| | | | | |
|---|---|---|---|---|
| and genome names of genome fasta files for analysis | | | | |
| Directory for output files | -outdir | name of directory | `-outdir Run1` | Required. |
| Kmer size | -k | odd integer greater than 7 | `-k 13` | Required. Length of kmer containing the SNP. |
| File of genome names to use as references for gene annotation of SNPs | -annotate | file name | `-annotate annotatedGenomes` | Optional. List of names of genomes to use for annotation (see next section). Default is no annotation. |
| Calculate core SNPs and core parsimony tree | -core | no value | `-core` | Optional. Calculate a set of loci and a tree based on only the SNPs found in all genomes. |
| Minimum fraction of genomes with locus | -min_frac | decimal fraction between 0 and 1 | `-min_frac 0.5` | Optional. Calculate a tree based on only SNP loci occurring in at least this fraction of genomes. |
| Number of CPUs to use | -CPU | integer between 1 and the number of CPU's available | `-CPU 12` | Optional. Defaults to the number of processors available. |
| Estimate Neighbor Joining tree | -NJ | no value | `-NJ` | Optional. Calculates an NJ tree. Default is to not calculate an NJ tree. |
| Estimate | -ML | no value | `-ML` | Optional. |

| | | | | |
|---|---|---|---|---|
| maximum likelihood tree | | | | calculates an ML tree. Default is to not calculate an ML tree. |
| Generate VCF (Variant Call Format) files | -vcf | no value | `-vcf` | Optional. Generates VCF files. Default is to not generate VCF files. |
| Set the debug state | -debug | no value | `-debug` | Optional. |

* file and directory names must not contain any spaces and should avoid any special symbols such as ()*& %#@{}[]. Case matters. -nj is not the same as -NJ! See Section IV for the rules about naming files.

**Example:** `kSNP4 -in Eco100.in -k 21 -outdir Run1 -annotate annotatedGenomes`

The command line in the example above says that ***kSNP4.1*** is to use `Eco100.in` as the input file, the kmer size is set to 21, and the output files will all be written in a directory named `Run1` to be created in the current directory. In addition to those required arguments ***kSNP4.1*** is to annotate the genomes using a list of genome names in the file `annotatedGenomes` that match genome names in `Eco100.in`. The files listed in annotatedGenomes **must** themselves be annotated and must have an accession number.

## Annotating genomes

It can be very useful to know something about where each SNP is found in a genome, what gene product (if any) the SNP lies within, whether the SNP results in a synonymous or non-synonymous changes, etc. ***kSNP4.1*** retrieves that information from NCBI by downloading the fully annotated version of a genome file and using the information in those annotations to annotate each SNP. The command to annotate (-annotate) must be followed by the name of a file that lists the genomes to use for annotation, `annotatedGenomes` in the above example but you can name the file anything you wish.

To easily make a list of file names run the program ***genomeNames4***. The command is:
`genomeNames4 infileName outfileName`, where `infileName` is the name of the input file you used for ***kSNP4.1*** and `outfileName` is the name of the file listing the genomes to use for annotation.
Example: `genomeNames4 Eco100.in annotatedGenomes`.

It is important to edit the resulting file to list only genomes that you know are annotated in NCBI to avoid wasting NCBI resources by requesting / attempting to download annotation data for un-annotated genomes. It is also important to put the most reliably annotated genomes at the top of the list; i.e. finished genomes before annotated genome assemblies because SNPs are annotated with reference to the first genome in the list in which the SNP is present. Do not include genomes available only as raw reads in the annotatedGenomes list! Only include those genomes for which you want to know the position of the SNP. Otherwise, kSNP4.1 will waste time finding and reporting positions for every read that covers the SNP, which can result in a massive SNPs_all file.

### Choosing the genomes to use for annotation

Every genome that is in the annotatedGenomes list must be a genome for which the annotations are available at NCBI in a .gbk file. You can assume that all complete genomes are full annotated. Some that are not complete, but instead are available as scaffolds will be annotated, while those that are only available as contigs may or may not be annotated. The best choices are closed, complete genomes.

## Capture a Logfile

By default ***kSNP4.1*** generates screen output as it runs, including the time used in the run. It can be useful to have a record of that output, for instance to compare run times for different data sets. To capture the screen output as a log file use "tee" to send the output to both the screen and a file. In the bash shell the command is implemented by adding `tee Logfile.txt` after the usual kSNP4 commands, for example:

```
kSNP4 -in Eco100.in -k 21 -outdir Run1 -annotate annotateGenomes | tee
Run1Log.txt
```

If you encounter unexpected output from kSNP4.1 you can use the kSNP_verbose.log file that is automatically written for later review.

## Other arguments

**minimum fraction with locus:** Because of indels or sequence variations within the kmer around a SNP, many SNPs will not be present in all of the genomes, and some may be present in only two genomes. In addition to analyses based on all of the SNPs and analyses based only on those that are present in all genomes (core SNPs), it is sometimes useful to also base a SNP analysis only on the SNPs that are present in some minimum fraction of the genomes . This can be set to any decimal fraction between 0 and 1.0. Note that any value under 2/Number of genomes (e.g. 0) is the same as all SNPs, and any value greater than 1 – 1/Number of genomes (e.g. 1) is the same as core SNPs.

## kSNP4.1 tree accuracy

Simulation studies (Hall, 2015 Cladistics **32:** 90-99) have shown that kSNP parsimony trees are the most accurate, followed by ML trees, with NJ trees being the least accurate. For that reason the default is to only estimate parsimony trees. The parsimony tree that is estimated is a consensus of up to 100 equally parsimonious trees. If other tree methods such as NJ or ML are specified those trees are writing *in addition to* the parsimony trees.

## Labeling tree nodes with imperfect but significant SNPs associated with the node

The tree_AlleleCounts.X.tre output files are phylogenetic tree files in which the nodes are labeled with the number of SNPs that are present in all of the descendants of that node and nowhere else (see Table 4, below). Think of those SNPs as being perfectly associated with that node. In many cases there are nodes that have no perfectly associated SNPs, but nevertheless do have SNPs that are imperfect but significantly associated with that node. The utility ***NodeChiSquare2tree4*** identifies those SNPs and writes a tree file in which the nodes are

labeled with the number of those significantly associated SNPs. (See section X, The kSNP4.1 Utilities, under Utilities to use after running kSNP4).

***NodeChiSquare2tree4*** is run from within the kSNP4.1 output file directory after kSNP4.1 has run. Decide if you want to run ***NodeChiSquare2tree4*** after examining the tree_AlleleCounts.X.tre file of interest.

# VIII. Tutorials with Examples

Because ***kSNP4.1*** was designed to maximize flexibility, and in anticipation of future uses of comparative SNP information, ***kSNP4.1*** writes a plethora of output files. Indeed, analysis of just 47 bacterial genomes produces up to 106 output files, depending on the options that were chosen on the command line.

Note! You should download the Example.zip sets of input files that are discussed below. Going through the examples will not only familiarize you with ***kSNP4.1***, it will allow you to determine whether you have installed ***kSNP4.1*** correctly.

Note! Please do not copy the command line entries from this document. The dashes that precede the flags may be converted to em-dashes in this document, in which case nothing will work. Instead, enter the commands manually or copy them from the CommandLines.txt file in each example.

## VIIIA Tutorial: Example 1: Only finished genomes

The first example is one in which the data set consists of finished genome sequences. The Example1 directory contains three items: (1) the data set within a directory named Genomes. That data set consists of finished sequences of 10 encephalitis virus genomes. (2) a file named CommandLines.txt and (3) a directory named "Results" The purpose of the Results directory is to allow you to compare your results from this tutorial with our results. Within that directory there are the input files Example1.in and annotatedGenomes for comparison. ***You cannot use the file Example1.txt for input to your kSNP4.1 program!!!*** The paths in that file are not correct for you because the Genomes directory is somewhere on your hard drive and the paths point to the directory on Barry Hall's machine. The results of Runs 1, 2 & 3 are also included. All of that information is for you to compare with the input and output files that you generate during this tutorial.

**Step 1: Generate the input file and run Kchooser4**

1. Generate the input file list by running ***MakeKSNP4infile***. Navigate to the Example1 directory and enter `MakeKSNP4infile -indir Genomes -outfile Example1.in.` The result will be an input list file, Example1.in.
2. Run ***Kchooser4*** to determine the optimum kmer length and to estimate fraction of the kmers found that are core kmers (FCK). Enter `Kchooser4 -in Example1.in`. The output file will be named Kchooser4_Example1.report. It tells us that the optimum k is 11, and FCK = 0.05. Since FCK is < 0.1 the accuracy of the tree is probably not very high and we should be cautious about its interpretation.

3. Generate a list of annotated genomes. Since all of the genomes are annotated you can extract the list from Example1.in. For the sake of this tutorial call the output file annotatedGenomes, but you could call it anything you like. Enter `genomeNames4 Example1.in annotatedGenomes`

**Step 2: Run kSNP4.1**

2A **without annotations**

The simplest way to run kSNP4.1 is without doing any SNP annotation. We will set the kmer size to 11 as suggested by ***Kchooser4***. We will call this run Run1. All the options except -in, -outdir, and -k take their default values.

Enter `kSNP4 -in Example1.in -outdir Run1 -k 11 | tee Run1Log`

The 20 output files will be found in the Run1 directory. At the bottom of the Run1Log file you will see the time that the run required. Note that in column 4 of the SNPs_all file, the position information is all x (unknown) because that information is not calculated unless annotations will be performed.

2B **with annotations**

To annotate the SNPs we simply add the -annotate option, specifying the annotatedGenomes file we generated in step 1(3) that lists the annotated genomes.

In this case all 10 of the genomes are annotated so the annotatedGenomes file lists all of the genomes. More typically a data set would include some finished genomes, some assembled genomes that are annotated, some assembled genomes that are not annotated, and perhaps some raw read files. In a case like that you would delete any unannotated and raw read genomes from the annotatedGenomes list file before using it with kSNP4.1.

Enter: `kSNP4 -in Example1.in -outdir Run2 -k 11 -annotate annotatedGenomes | tee Run2Log`

Because of annotation you will find 24 files in the Run2 directory. From the log file you will see that annotation increased the run time.

The annotations output files you want to look at are "SNPs_all", "Annotation_summary", and "SNPs_all_annotated". See Table 3 for the contents of those files

2C **with all options used**

Finally, we can add the options to estimate ML and NJ trees, to generate VCF files, to calculate core SNPs to estimate a core parsimony tree, and to set the minimum faction with locus to 0.75.

Enter: `kSNP4 -in Example1.in -outdir Run3 -k 11 -annotate annotatedGenomes -ML -NJ -vcf -core -min_frac 0.75 | tee Run3Log`

The Run3 directory will now include 107 files, and the run further increased.

The purpose of these multiple runs was to illustrate that the more you ask kSNP4.1 to do, the longer it takes. It is also the case that the more genomes you include in the data sets, the larger those genomes are and the more genomes you choose to annotate, the longer it takes.

## VIIIB Tutorial Example2: Finished and unfinished genomes

Before discussing this example, a quick word about unassembled genomes. Before running ***kSNP4.1***, we advise a first step of filtering/trimming low quality bases or reads from the fastq using something like fastq_quality_trimmer or seqtk before creating the fasta file of reads. This will help avoid considering sequencing errors as SNPs.

This *Vibrio cholera* data set includes three finished genomes (VcMS6.fa, VcLMA3984-4.fa and VcO1-ElTorN16961.fa, each of which consists of two chromosomes), two assembled genome that consists of a set of contigs (Vc523-80.fa and Vc63-93_MO45.fasta), and an unassembled genome that consists of **raw reads** that has been filtered and trimmed (ERR579925.fasta). You should notice that the raw reads file is *huge*, 617 megabytes, compared with the other files that are about 4 MB.

The purpose of this example is four-fold: (1) to illustrate the use of a mixture of genome types, (2) to point out how ***kSNP4.1*** handles and annotates multi-chromosome genomes, (3) to illustrate the use of ***Kchooser4*** to determine the optimum kmer size with bacterial genomes.

The Example2 directory contains the same items as does the Example1 directory

Begin by navigating to the Example2 directory and making a ***kSNP4.1 / Kchooser4*** input file just as you did for Example1. The command line is `MakeKSNP4infile Genomes Example2.in`.

Next run ***Kchooser4*** using the command line `Kchooser4 –in Example2.in`. The output file will be named Kchooser4_Example2.report. The optimum value of k is 21, and FCK = 0.727. FCK is well above 0.1, so the trees estimated by ***kSNP4.1*** should be quite accurate and > 97% of the SNPs should be found.

Next, make a list of the genomes to use for annotation. Enter `genomeNames4 Example2.in annotatedGenomes`. The annotatedGenomes file will list all six genomes, but we know that only VcMS6.fa, VcLMA3984-4.fa and VcO1-ElTorN16961.fa are complete and should be used for annotation. Open annotatedGenomes in a text editor and delete the lines that are not complete genomes. Be sure to delete the entire line in each case.

Finally, after all that, run ***kSNP4.1*** by entering:

```
kSNP4 –in Example2.in –outdir Run1 –k 21 –annotate annotatedGenomes | tee Run1Log
```

## IX The output files

Most of the output of ***kSNP4.1*** is presented separately for three groups of SNPS:

(a) ***all*** SNPs
(b) the ***core*** SNPS, which are the SNPs that are present in all of the genomes
(c) the ***majority*** SNPs, which are SNPs that are present in the user-defined minimal fraction of genomes (see option –min_frac in Table 1). The cutoff for inclusion in the majority is incorporated into the name; i.e. majority0.5 means that only SNPs in at least 0.5 of genomes are included. ***Core*** and ***majority*** SNPs are subsets of ***all*** SNPs.

The output files can be grouped into several categories:

(1) alignment, or *matrix*, files that contain alignments of various groups of SNPs
(2) files that give information about the SNPs, including position of the SNPs in the genomes and annotation information
(3) files that give phylogenetic trees based on all, core, or majority SNPs created using several methods
(4) files that give counts of SNP alleles shared by various subsets of genomes
(5) files that give the number of SNPs that correspond to nodes on the trees

(6) files that give information about homoplasy groups
(7) files that give information for each locus about the node or homoplasy group to which it belongs
(8) other files

Table 2. The alignment files

| File name | Explanation |
|---|---|
| SNPs_all_matrix<br>core_SNPs_matrix<br>SNPs_in_majority0.5_matrix | Relaxed PHYLIP format of the SNP alleles. Genome name - SNP allele string<br>The SNP alleles for each genome are concatenated into a string whose length is the number of SNPs. N indicates that the SNP is absent in that strain. Loci are concatenated in the same order as listed in the SNPs_all file. |
| SNPs_all_matrix.fasta<br>core_SNPs_matrix.fasta<br>SNPs_in_majority0.5_matrix.fasta | >Genome name<br>SNPs string<br>The same information as above, except in fasta format with missing SNP:s indicated by a dash rather than an N. |

The fasta alignment files can be thought of as the "master" files of SNPs. Fasta is the most common input format for phylogenetics software, and can be used to estimate either phylogenetic or minimum spanning trees using your favorite software. Likewise, the fasta files can be used for other purposes such as phenotype association studies.

Table 3. Files that provide information about the SNPs

| File name | Explanation |
|---|---|
| SNPs_all_annotated | Provides the most complete information about each SNP. Each SNP is given a locus number, and a particular SNP is listed once for each genome in which it occurs. Information followed by an asterix is provided only for annotated portions of annotated genomes. The information includes the SNP base in that genome, the genome name, the accession number*, the position of the SNP in the genome*, the codon in which it occurs, the amino acid encoded (in protein coding regions), the residue peptide context*, and the gene product*. |
| Annotation_summary | For each SNP shows the alternative SNPs, e.g. G_A, the alternative codons, e.g. ACA_GCA; the alternative amino acids, e.g. T_A, whether the SNP alleles are Synonymous or Nonsynonymous, and the gene product. If the SNP site is not present in an annotated genome that SNP is shown as **Not in annotated genome**. If the SNP is in an annotated genome but not in an annotated region it is shown as **Not in annotated region**. |
| SNPs_all<br>SNPs_in_majority0.5<br>core_SNPs<br>nonCore_SNPs | Similar to above, but provides less information: SNP number, context (the SNP allele with the central base indicated by a dot), SNP base, *position in genome, strand,* and genome name. **Italicized information is provided only for genomes listed in the annotated_list file (argument -annotate)!** |

These files in Table 3 provide a wealth of information about each SNP, some of which is essential if you are exploring the roles of SNPs. For instance, if you have identified certain SNPs as being associated with a particular phenotype, then these files tell you not only where those SNPs are, but what proteins they are in etc.

Table 4. Tree files

| Tree files are written for each phylogenetic method that is used. Below each file is described with the word 'method' substituted for the particular phylogenetic method. In each case an example tree using the parsimony method is provided. Methods are parsimony, ML, NJ and majority#.# where #.# can range from 0.0 to 1.0. | |
|---|---|
| Parsimony trees are consensus trees based on an Extended Majority Rule consensus of the equally most parsimonious trees from a sample of 100 trees. | |
| **Example File** | **Contents** |
| | |
| tree.parsimony.tre | The tree.method.tre trees have internal node labels that show the support for that node as calculated by FastTreeMP. |
| | |
| tree_AlleleCounts.parsimony.tre | The tree_AlleleCounts.method.tre trees have internal node labels that show the number of SNP alleles that are present in all descendants of that node and nowhere else. Allele counts for branch tips are not shown. |
| | |
| tree_AlleleCounts.parsimony.NodeLabel.tre | The tree_AlleleCounts.method.NodeLabel.tre trees all have internal node labels that show the node numbers corresponding to node numbers in ClusterInfo files and the number of SNP alleles that are present in all descendants of that node and nowhere else. |
| | |
| tree_tipAlleleCounts.parsimony.tre | The tree_tipAlleleCounts.method.tre files are similar to tre_AlleleCounts.x.tre trees except that strain names have been modified to show the strain specific allele counts with an "_" after the strain name. |

The Neighbor Joining (NJ) trees are based on the number of SNP allele differences between sequences. The distances used for those differences are 2 for locus presence/absence (present in one sequence and absent in the other), 1 for allele differences (both sequences contain the locus but have a different allele), and 0 if the locus is absent in both sequences or both share the same allele.

The 20 files that have the extension .tre are all phylogenetic tree description files in the Newick format. Those files are understood by most tree drawing programs, but we suggest MEGA https://www.megasoftware.net/ which is available for Mac OS X and Linux platforms, to visualize the trees. MEGA is discussed in Hall, B. G. 2013 Building Phylogenetic Trees from Molecular Data with MEGA. Mol. Biol. Evol. 30: 1229-1235, doi: 10.1093/molbev/mst012 https://academic.oup.com/mbe/article/30/5/1229/992850?login=false

It is important to bear in mind that all of the trees are ***unrooted*** trees. The tree drawing programs will, by default, draw them in the rectangular phylogram or cladogram format so that they appear to be rooted. That appearance is for convenience only. There is no evolutionary direction in these trees and you should not infer such a direction. If you have knowledge that a cluster is a true outgroup with respect to the rest of the sequences then you can root the tree on that outgroup in the tree drawing programs, after which you can legitimately infer evolutionary direction.

It is also important to bear in mind that branch lengths are expressed in terms of changes per number of SNPs, not changes per site. When trees are based on comparisons of aligned sequences, branch lengths are expressed in terms of changes per site, which means changes divided by the length of the alignment; many of the sites may be invariant. When trees are based on SNPs there are no invariant sites, so the number of sites is likely to be much smaller. The result is that the absolute value of branch lengths are likely to be much higher than one would see on an alignment-based tree. The relative branch lengths, however, should be very similar. SNP-based branch lengths should not be used for estimating divergence times, but for other purposes should present no problems.

Table 5. Other Output Files

| **Various counts of the number of SNPs** | |
|---|---|
| **File name** | **Explanation** |
| COUNT_SNPs | Shows the total number of SNPs |
| COUNT_coreSNPs | Shows the number of core SNPs, the number of non-core SNPs, and the number of SNPs in at least "minimum fraction with locus" sequences |
| | |
| **As in Table 4 many kinds of files are written for each phylogenetic method and/or for subsets of SNPs. In each case the word 'method' is substituted for the method or SNP subset. Methods are parsimony, ML, NJ, core, and majority#.#. An example using the parsimony method is provided** | |
| | |
| COUNT_Homoplastic_SNPs.method (e.g. COUNT_Homoplastic_SNPs.parsimony) | The COUNT_Homoplastic_SNPs files all show the number of homoplastic SNPs (that do not correspond to a node) on the indicated tree. |
| | |
| | |
| tip_SNP_counts.method (e.g. tip_SNP_counts.parsimony) | The tip_SNP_counts.method files give the number of genome-specific alleles in each genome. Useful to show the strain specific allele counts with FigTree, for the tree_AlleleCounts trees. |
| | |
| **Node_SNP_counts** | |
| | |
| Node_SNP_counts.method (e.g. Node_SNP_counts.parsimony) | The Node_SNP_counts files give the number of SNPs that correspond to nodes on the indicated tree, and the genomes under that node. |
| | |
| **Homoplasy groups** | |
| | |

| | |
|---|---|
| Homoplasy_groups.method<br>(e.g. Homoplasy_groups.parsimony) | The Homoplasy files give the number of SNPs in each homoplastic group of genomes. These files are similar to the Node_SNP_counts files, except for the groups of genomes that share SNPs but that do not correspond nodes of the tree. They give a group identifier (e.g. "Group.25" which corresponds to the info reported in the ClusterInfo files), and the number of target sequences that make up this group and the number of SNP alleles that are shared by this group of genomes, followed by the genome identities that make up the group. |
| | |
| **ClusterInfo files** | |
| | |
| ClusterInfo.method<br>(e.g. ClusterInfo.parsimony) | The ClusterInfo files list for each locus which node or homoplastic group of sequences the locus is present in. Group numbers correspond to the groups listed in the Homoplasy_groups files. |
| | |
| **Still more files** | |
| **File name** | **Explanation** |
| kSNP_verbose.log | A verbose log of all of the output of constituent programs when the -debug option is not invoked. If a problem arises this can be helpful in diagnosing the source of the problem. |
| annotate_list | List of annotated genomes. Empty if no list was input |
| | |
| fasta_list | List of genome fasta files. Identical to the input list file |
| genbank_from_NCBI.gbk | The concatenated set of GenBank files that were downloaded based on the accession numbers of the finished genomes |
| headers.annotate_list | list of the fasta header lines for the annotated genomes (needed for annotation) |
| NJ.dist.matrix | A pairwise distance matrix showing the distances between the genomes on the NJ tree |
| unresolved_clusters | Lists the genomes that fall into unresolved clusters and those that are uniquely resolved. Sequences with the same cluster number in the first column are not resolvable, but those with different cluster numbers are uniquely resolved by SNPs. |
| VCF.reference_genome.vcf | variant call format file ( http://en.wikipedia.org/wiki/Variant_Call_Format ) for compatibility with other tools, using the reference genome indicated in the file name. Since some SNPs may not be present in this reference genome, these are listed in the file VCF.SNPsNotInRef.reference_genome. The reference genome is automatically selected to be the first genome in the finished_genomes file (-annotate option), or if that is empty, the first sequence in the input list (-in option).<br>Note! These files are generated only if the –vcf option is invoked on the command line.<br>Be aware that generating these files can use enormous amounts of |

| | |
|---|---|
| | memory. In a data set consisting of over 200 bacterial genomes and containing over 2,000,000 SNPs generating these files used over 37 GB of RAM. |

# X. The kSNP4.1 Utilities

In addition to the executables (programs) that make up ***kSNP4.1*** itself there are several "utility" programs that are designed to make your life easier. You have already encountered three utility programs: ***MakeKSNP4infile, genomeNames4,*** and ***Kchooser4***. This section discusses several other utility programs in detail. You won't always need the utility programs, they are just there to make life easier for you when you do need them.

## Utilities in alphabetical order

### checkGenbankFromNCBI

During the annotation process ***kSNP4.1*** downloads and concatenates all of the Genbank files of the genomes that are specified for annotations. Those Genbank file are concatenated to create the file genbank_from_NCBI.gbk. Occasionally one or more files will fail to download and ***kSNP4.1*** will crash during the annotation process with an error message that includes "ParAnn returned -1".

The utility checkGenbankFromNCBI checks the genbank_from_NCBI.gbk file by comparing it with the headers.annotate_list file that lists the GenBank files that *should* have been downloaded. Navigate to the directory that contains the ***kSNP4.1*** output files. Enter `checkGenbankFromNCBI`. The accession numbers of the missing GenBank files will be written to a file named missing_accession_numbers.txt.

Using those accession numbers and your favorite browser retrieve each of those files from NCBI (https://www.ncbi.nlm.nih.gov/nucleotide) and copy the contents of that file starting with the word "Locus" down through the terminator "//" and paste it at the end of the genbank_from_NCBI file and save the file.

Finally, navigate to the directory that contains the ***kSNP4.1*** output files, including the modified genbank_from_NCBI file. Enter `ParAnn` to restart the annotation process and complete the ***kSNP4.1*** analysis.

### getFastaGenomes

Batch downloads a list of genomes from NCBI. See Section XII.

### extractNthLocus4

This utility extracts locus number n from a SNPs file. In the outfile it lists, for each genome in which the SNP is present: the SNP number, the sequence surrounding the central base, the position of the SNP in that genome, the amino acid, the genome name, and the genome accesion number.

Example for SNP 9

```
9    AAACAA.AGGAGC  A    1842 F    TTV_T3PB  AF247138.1
9    AAACAA.AGGAGC  C    1865 F    TTV_BDH1  AF116842.1
9    AAACAA.AGGAGC  C    1865 F    TTV_1a    AB017610.1
```

The command is:
`extractNthLocus4 n (SNPs_all` or `core_SNPs` or `SNPs_in_majority0.5) > outfileName`

Examples:

```
extractNthLocus4 5 core_SNPs > locus_5_in_core_SNPs
extractNthLocus4 155 SNPs_all > locus_155_in_SNPs_all
extractNthLocus4 30 SNPs_in_majority0.5  > locus_30_in_SNPs_in_majority
```

### fixOldFastaHeaders

Problem: one or more of the genome files includes an old-style fasta header that starts with gi|.

The presence of even one such header may cause ***kSNP4.1*** to crash. It is easy to miss seeing an old-style header for a plasmid or other replicon that is not the chromosome.

Navigate to the directory that *encloses* the directory that contains the fasta genome files and enter:
`fixOldFastaHeaders targetFolderName`

All old-style headers starting with `gi|` will be corrected. New-style headers will be unchanged.

### genomeNames4

Extracts the genome names from a ***kSNP4.1*** input file such as Example1.in and writes a file that lists the genome names.
Usage: `genomeNames4 inFile.in annotatedGenomes`

### Kchooser4

***Kchooser4*** is discussed extensively in Section VI. The command line is
`Kchooser4 -in infileName.in` where infileName.in is the input file that you plan to use for ***kSNP4.1***.

### MakeKSNP4infile

***MakeKSNP4infile*** is discussed extensively in Section V.

### NodeChiSquare2tree4

***NodeChiSquare2tree4*** identifies for each node the number of SNPs for which the $\chi^2$ probability of the alleles being randomly distributed with respect to that node are ≤ some threshold *p* value, then writes a tree file with the nodes labeled with these counts; e.g. tree_ChiSqAlleleCounts.X.tre where X is the tree type (ML, parsimony, core or majority0.5)

***NodeChiSquare2tree4*** is run from within the ***kSNP4.1*** output file directory after ***kSNP4.1*** has run.

Usage: `NodeChiSquare2tree4 -p maximumChiSqProbability -f tree_AlleleCounts file`

Both arguments are required. If -h is used, or if an error is detected, a help / usage screen may be displayed.

`-p` maximum Chi Sq probability for assigning a SNP to a node, e.g. 0.05, 0.01

`-f` tree_AllelesCounts is one of these files (not all will be present depending on the kSNP4.1 command line):

tree_AlleleCounts.core_SNPs.ML.tre
tree_AlleleCounts.core_SNPs.NJ.tre
tree_AlleleCounts.core_SNPs.parsimony.tre
tree_AlleleCounts.SNPs_all.ML.tre
tree_AlleleCounts.SNPs_all.NJ.tre
tree_AlleleCounts.SNPs_all.parsimony.tre
tree_AlleleCounts.parsimony.tre
tree_AlleleCounts.SNPs_in_majority0.75.ML.tre
tree_AlleleCounts.SNPs_in_majority0.75.NJ.tre
tree_AlleleCounts.SNPs_in_majority0.75.parsimony.tre

***NodeChiSquare2tree4*** produces two files, NodeChiSquares.txt listing the number of alleles significantly associated with each node, and tree_ChiSqAlleleCounts.X with a Newick tree showing the number of significant alleles at the nodes.

### parseNCBIcsv

Parses a file of bacterial genome sequences downloaded from NCBI. See Section XII

### parseViralNCBIcsv

Parses a file of viral genome sequences downloaded from NCBI. See Section XII

### rmNodeNamesFromTree4

***rmNodeNamesFromTree4*** removes the labels from internal nodes.

kSNP4.1 labels the internal nodes with a name. Those labels may interfere with other labels that the drawing program writes. rmNodeNamesFromTree4 simply removes the ***kSNP4.1*** node labels

Usage:

```
rmNodeNamesFromTree tree.withNodeLabel.tre tree.nodeLabelsRemoved.tre
```

The first argument is the name of the tree. The second argument is whatever you want to call that tree with the internal node labels removed

## XI How kSNP4.1 works

There are three fundamental functions of kSNP4.1:
- Finding Single-Nucleotide Polymorphisms (SNPs) (steps 1-3),
- Annotating found SNPs using Genbank annotation data (steps 4, 6 and 7),

and
- Generating phylogenetic trees based on those SNPs (steps 5a - 5c).

kSNP4.1 works by 1) processing the input genome data into a per-genome kmer list with low-frequency kmers filtered out, 2) splitting kmers into partitions to speed processing (permits parallelization), 3) identifying SNPs within a given partition by processing SNP candidates to: 3a) Remove conflicting genomes (i.e. kmers with allelic variation within a genome), and then 3b) Identify any remaining allelic variations between genomes as SNPs for those genomes, 4) Associating SNPs with locations within genomes using mummer (only for those genomes to be annotated), 5) building and labeling trees by: 5a) Creating SNP matrixes (including core and minimum fraction) and then 5b) Building trees (parsimony, ML, NJ) followed by 5c) Clustering SNPs by genome groups and label tree nodes with allele counts, and then finishing up by 6) collecting annotation data from Genbank / NCBI, and finally 7) associating that annotation data with SNPs.

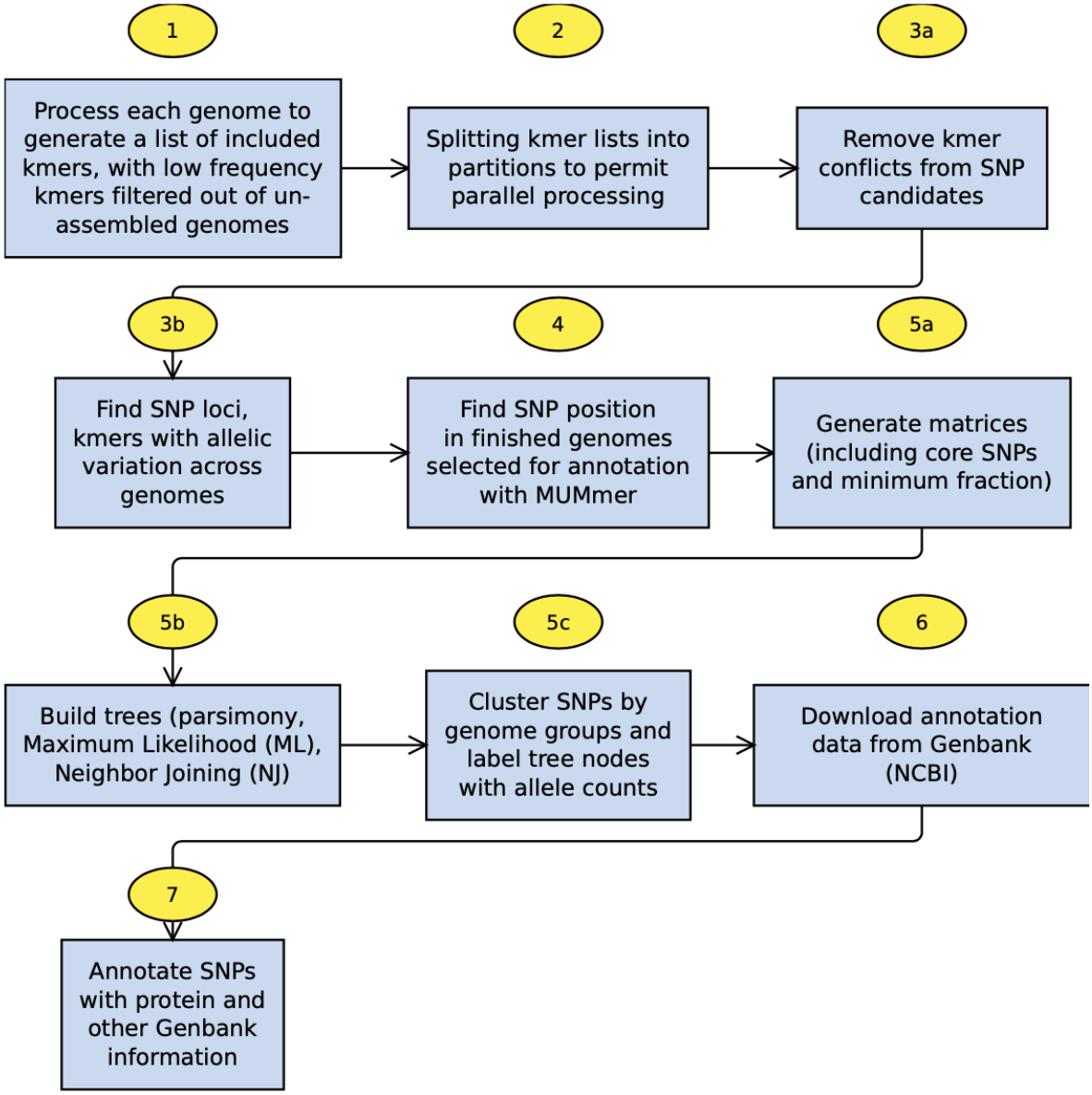



Figure 1: diagram outlining kSNP4.1 process

## XII Downloading genome sequences from NCBI

While not actually part of the kSNP4.1 process, acquiring genome sequences is the essential first step in genome analysis.

### Downloading Bacterial Genome sequences from NCBI

Strictly speaking, the matter of downloading bacterial genomes is not part of kSNP4.1, but you can't do much with kSNP4.1 without genome sequences. Accordingly, three programs to assist in obtaining bacterial genome

sequence are included with the kSNP4pkg. Those programs are **FilterCSV, parseNCBIcsv**, and **getFastaGenomes**. This document explains how to use those programs
With over 24,800 eukaryotic genomes, 435,000 prokaryotic genomes, 50,500 viral genomes, 41,000 plasmid, and 25,000 organelle sequences available at NCBI as of August 2022 identifying and downloading sequences of interest can be challenging, to say the least (https://www.ncbi.nlm.nih.gov/genome/browse/#!/overview/).

## Downloading a list of genomes of a species of interest

In your favorite browser go to https://www.ncbi.nlm.nih.gov/genome/browse/#!/overview/. It will look like Figure 2 below

Figure 2

On that page, there is a list of organisms, 50 per page. At the present time there are 1,428 pages of organisms. You could spend a lot of time trying to find the page that has the organism of interest to you. Fortunately you can quickly go to the one that is of interest to you.
Notice the list of alternative biological kingdoms (red arrow) that begins with **Overview** in boldface (Figure 2). That tells you that you are currently in the **Overview** view.

## Prokaryotic Genomes

For this example I am interested in *E. coli*, which is in the Prokaryotes kingdom, so I will click Prokaryotes to bring up the view in Figure 3. There are 436,914 prokaryotic genomes listed.

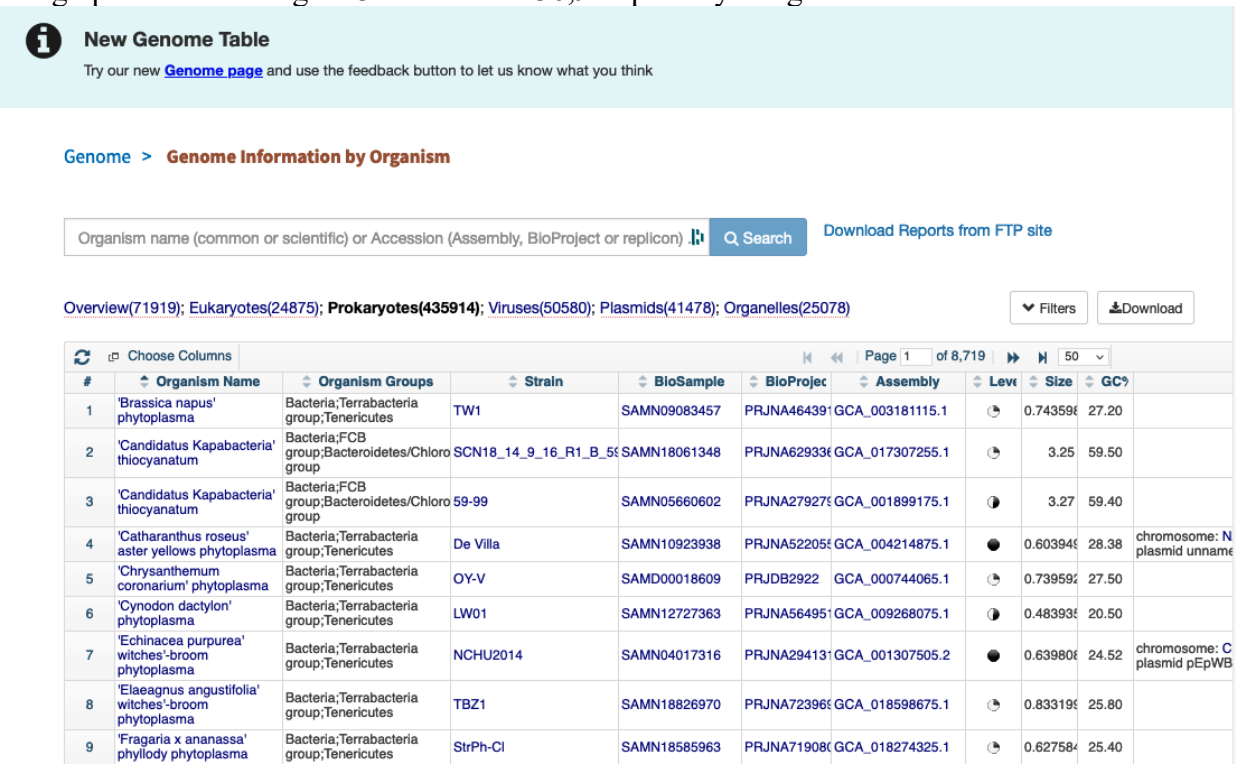


Figure 3

## E. coli

To quickly get to E. coli genomes enter **Escherichia coli** in the search box (Figure 4). Click the Search button next to the search box (blue arrow in Figure 4).

Figure 4

Notice that figures 1 and 2 show 435,914 Prokaryotes, but figure 3 shows only 32,687 Prokaryotes. That is because that number now refers only to the number of *E. coli* genomes.

You can reduce the number of genomes by including serotype information in the search box, e.g. O157:H7, the very pathogenic serotype original identified as the Jack in the Box strain (Figure 4)


Figure 5

Notice that the number of Prokaryotes is now 320. However the Search does not work as you might expect. If you enter just O157 (Figure 6)


Figure 6

One might expect that O157 would include all the O157:H7 isolates, but that is not how the NCBI search works.

Table 1 shows the number of prokaryotic genomes with different Search Entries. The search box is great for searching with terms that NCBI uses, e.g. serotypes, but that is all that it can do to limit the number of genomes it finds.

**Table 1**

| Search Entry | Prokaryotes found |
|---|---|
| Escherichia coli | 32687 |

| | |
|---|---|
| Escherichia coli K-12 | 92 |
| Escherichia coli O157 | 23 |
| Escherichia coli O157:H7 | 320 |
| Escherichia coli O145 | 12 |
| Escherichia coli O145:H28 | 105 |
| Escherichia coli O127 | 0 |
| Escherichia coli O127:H6 | 7 |

Each line in Figure 4 is the genome of a different *E. coli* isolate. There are 32,687 *E. coli* genomes shown, one per line. Each line has several fields; at the top of the list the name of each field is shown. The figure is not wide enough to see all of the fields.

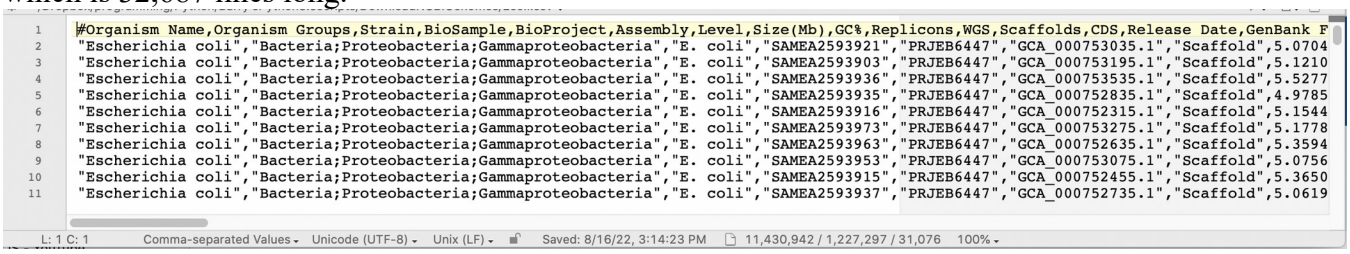
The fields we are interested in are: Strain, Level, and Replicons. "Strain" is obvious. "Level" refers to completeness of the genome sequence: Complete, Chromosome, Scaffold, or Contigs. "Replicons" lists the GenBank accession numbers for each of the replicons.

The first isolate listed in Figure 4, E. coli K-12 substr. MG1655, shows two accession numbers in the Replicons field. The first starts with NC and is the accession number for the Reference Genome, the second accession number, separated from the first accession number by a '/', is the accession number of the Genbank sequence. They are the same, but are stored in different NCBI databases.

The second isolate listed is Escherichia coli O157:H7 str. Sakai. The **Replicons** field shows that there are 3 replicons present, the chromosome and two plasmids. There are two accession numbers for each replicon.

Now that we understand what the fields are we could download the entire list of all 32,687 *E. coli* genomes, not just the 50 genomes visible on page 1 in Figure 3, by clicking the Download button (red arrow) at the upper right of Figure 3. **Whatever you entered in the search box, the downloaded file is always named prokaryotes.csv.** I suggest renaming that file to Ecoli.csv once the download completes. Move the Ecoli.csv file to a convenient folder for the remaining steps.

The upper portion of the file Ecoli.csv looks like Figure 7, but fortunately you don't ever have to read that file, which is 32,687 lines long.



```
#Organism Name,Organism Groups,Strain,BioSample,BioProject,Assembly,Level,Size(Mb),GC%,Replicons,WGS,Scaffolds,CDS,Release Date,GenBank F
"Escherichia coli","Bacteria;Proteobacteria;Gammaproteobacteria","E. coli","SAMEA2593921","PRJEB6447","GCA_000753035.1","Scaffold",5.0704
"Escherichia coli","Bacteria;Proteobacteria;Gammaproteobacteria","E. coli","SAMEA2593903","PRJEB6447","GCA_000753195.1","Scaffold",5.1210
"Escherichia coli","Bacteria;Proteobacteria;Gammaproteobacteria","E. coli","SAMEA2593936","PRJEB6447","GCA_000753535.1","Scaffold",5.5277
"Escherichia coli","Bacteria;Proteobacteria;Gammaproteobacteria","E. coli","SAMEA2593935","PRJEB6447","GCA_000752835.1","Scaffold",4.9785
"Escherichia coli","Bacteria;Proteobacteria;Gammaproteobacteria","E. coli","SAMEA2593916","PRJEB6447","GCA_000752315.1","Scaffold",5.1544
"Escherichia coli","Bacteria;Proteobacteria;Gammaproteobacteria","E. coli","SAMEA2593973","PRJEB6447","GCA_000753275.1","Scaffold",5.1778
"Escherichia coli","Bacteria;Proteobacteria;Gammaproteobacteria","E. coli","SAMEA2593963","PRJEB6447","GCA_000752635.1","Scaffold",5.3594
"Escherichia coli","Bacteria;Proteobacteria;Gammaproteobacteria","E. coli","SAMEA2593953","PRJEB6447","GCA_000753075.1","Scaffold",5.0756
"Escherichia coli","Bacteria;Proteobacteria;Gammaproteobacteria","E. coli","SAMEA2593915","PRJEB6447","GCA_000752455.1","Scaffold",5.3650
"Escherichia coli","Bacteria;Proteobacteria;Gammaproteobacteria","E. coli","SAMEA2593937","PRJEB6447","GCA_000752735.1","Scaffold",5.0619
```



Figure 7

## Filter the .csv file

You certainly don't want to do anything with all 32,687 E. coli genomes so you need to choose some way to include only those of interest. Entering the serotype in the search box shortens that list (Table 1) and thus shortens the prokaryotes.csv file accordingly. But what if you want to constrain the .csv file by something other than serotype?

There is a set of *E. coli* strains from a 2014 study all of whose names begin with '2014C'. You can filter that Ecoli.csv file so that it includes only strains in which '2014C' occurs. '2014C' is the *search string*.

In Terminal run ***FilterCSV*** by entering (for this example:
`FilterCSV -s '2014C' -i Ecoli.csv -o 2014C.csv.`
That command line says to search the file Ecoli.csv and print every line containing the search string '2014C' to a new file named 2014C.csv'. Notice the single quotes around the search string (-s argument). The search string **must** be surrounded by single quotes.

The resulting file, 2014C.csv, lists 24 genomes, a relatively short file to deal with.

## Parsing the 2014C.csv file

***parseNCBIcsv*** is a program that takes a .csv file as its input and write an output text file that will be used as the input file to the program ***getFastaGenomes*** to actually download the genome sequences.
A .csv file lists both complete (Complete or Chromosome) and incomplete (Contig or Scaffold) genomes. The problem is that accession numbers in the Replicons field are **only** provided for complete genomes. That means that ***getFastaGenomes*** cannot download incomplete genomes because no accession numbers are available. As a result ***parseNCBIcsv*** only parses complete genomes.

In Terminal navigate to the folder containing 2014C.csv and run ***parseNCBIcsv***. The command line requires at least two arguments: `-in` infileName and `-out` outfileName , and there are two optional arguments: (1) `-p` a prefix for file names, (2) `-n`, an integer giving the number of files to list.

Issue the command:
`parseNCBIcsv -in 2014C.csv -out 2014CList.txt -n 20.`
These arguments tell ***parseNCBIcsv*** to read the 2014C.csv file and to write the genomes to the parsed file as 2014CList.txt. The 2014CList.txt file looks like Figure 8.

```
Strain  AccNums
2014C-4705      CP027640.1 CP027641.1
2014C-3050      NZ_CP027472.1 NZ_CP027473.1
2014C-3057      NZ_CP027387.1 NZ_CP027386.1
2014C-3051      NZ_CP027338.1 NZ_CP027339.1
2014C-3655      NZ_CP027351.1 NZ_CP027350.1
2014C-3599      NZ_CP027435.1 NZ_CP027436.1
2014C-4639      NZ_CP027361.1 NZ_CP027359.1
2014C-3061      NZ_CP027548.1 NZ_CP027549.1
2014C-3075      NZ_CP027447.1 NZ_CP027448.1
2014C-4587      NZ_CP027342.1 NZ_CP027343.1
2014C-4135      NZ_CP027310.1 NZ_CP027311.1
2014C-3338      NZ_CP027452.1 NZ_CP027453.1
2014C-3946      CP027344.1 CP027345.1 CP027346.1
2014C-3003      CP027672.1 CP027673.1 CP027674.1
2014C-4423      NZ_CP027454.1 NZ_CP027455.1 NZ_CP027456.1
2014C-3011      NZ_CP027591.1 NZ_CP027589.1 NZ_CP027590.1 NZ_CP027592.1
2014C-3097      NZ_CP027449.1 NZ_CP027450.1 NZ_CP027451.1
2014C-3307      NZ_CP027368.1 NZ_CP027369.1 NZ_CP027370.1
2014C-3084      NZ_CP027319.1 NZ_CP027320.1 NZ_CP027321.1
2014C-3716      NZ_CP027335.1 NZ_CP027336.1 NZ_CP027337.1
```

Figure 8

Of course, exactly what appears in the output file depends on the command line arguments.

Table 2. Command line arguments

| Argument | Flag | Value | Example | Comment |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Input .csv file | -in | input file name | `-in Canada.csv` | Required |
| Output file | -out | output file name | `-out CanadaList.txt` | Required |
| Max number of genomes to include | -n | max genomes | `-n 20` | |
| Number of random genomes | -ran | integer | `-ran 500` | |
| Prefix to prepend to each file name | -p | prefix | -p Eco | prepends Eco_ to each file name |

If the -n option is not used a message like Figure 9 will be displayed:

```
The file 2014C.csv contains 21 complete genomes.
If there are more than 100 complete genomes you will need
to use a text editor to manually break up the output
list into chunks of no more than 100 genomes to comply with NCBI
rules against dowwnloading more than 100 genomes at a time

Do you want to limit the number of genomes? (y or n)
```

Figure 9

In case your .csv file might be over 100 genomes ***parseNCBIcsv*** checks the number of genomes in the input file whenever you do not specify an number for the `-n` option. If you want all the genomes to be included just enter `n,` if you do want to limit the number enter `y` to see Figure 10

```
Enter the limit as an integer
```

Figure 10

then enter the limit you want. The result will be exactly the same as if you had entered the limit on the command line with the `-n` option.

### *A random set of genomes*

When the -n option is used to limit the number of genomes ***parseNCBIcsv*** returns the first n genomes in the .csv file. The basis of ordering those genome in the .csv file is not clear, but the order certainly is not random, so n does not represent a random sample of the complete genomes. The -ran option; e.g. -ran 500, returns a list of n genomes randomly chosen from all of the complete genomes. When using the -ran option the -n option probably should not be used because then the random genomes are chosen from the first n genomes. If you use both the -ran and -n options you will get a warning asking if that is really what you want to do. If it is what you want to do just enter 'y'.

## Download the 2014CList.txt genomes with *getFastaGenomes*

Begin by creating a new folder *within* the folder where you put the input file (2014CList.txt) to receive the downloaded genome files. We will create the empty folder 2014C_Genomes.

The last step is to run the program ***getFastaGenomes***. ***getFastaGenomes*** downloads all of the genomes in the input file. In Terminal navigate to the folder where you put the input file then enter

```
getFastaGenomes -i 2014CList.txt -o 2014C_Genomes -e
yourname@somewhere.com.
```

You *must* enter your email address for the -e option. That allows NCBI to contact you if you violate their excessive requests limit. That limit is 100 genomes at a time, but also an unspecified number of series of requests. If you actually need to download more than 100 genomes you will need to do separate runs of ***getFastaGenomes*** using different input files. You should do so on weekends or outside USA peak hours, separating the runs by an hour or so.

In a short time the folder 2014C_Genomes will contain the genomes from the 2014CList.txt.

Sometimes the .csv file will include multiple genomes with exactly the same strain name. Were you to use a list.txt file with multiple genomes having the same name ***getFastaGenomes*** would download each one, overwriting the previous genome of the same name. In the end you would have only one genome file of that name and you would have wasted time and NCBI resources downloading and overwriting those files. ***parseNCBIcsv*** avoids that problem by detecting duplicated names and appends a number to each name so that they are not overwritten. For example, 3 genomes, each named Ecoli_K-12, would automatically become Ecoli_K-12-0, Ecoli_K-12-1 and Ecoli_K-12-2 in the list file written by ***parseNCBIcsv***.

## Downloading Viral Genome sequences from NCBI

### Downloading a list of genomes of a virus of interest

In your favorite browser go to https://www.ncbi.SARSList.txtnlm.nih.gov/genome/browse/#!/overview/. It will look like Figure 11 below

Figure 11

On that page, there is a list of organisms, 50 per page. At the present time there are 1,439 pages of organisms. You could spend a lot of time trying to find the page that has the organism of interest to you. Fortunately you can quickly go to the one that is of interest to you.
Notice the list of alternative biological kingdoms (red arrow) that begins with **Overview** in boldface. That tells you that you are currently in the **Overview** view.

For this example I am interested in *SARS*, which is in the viruses kingdom, so I will click viruses to bring up the view in Figure 12.

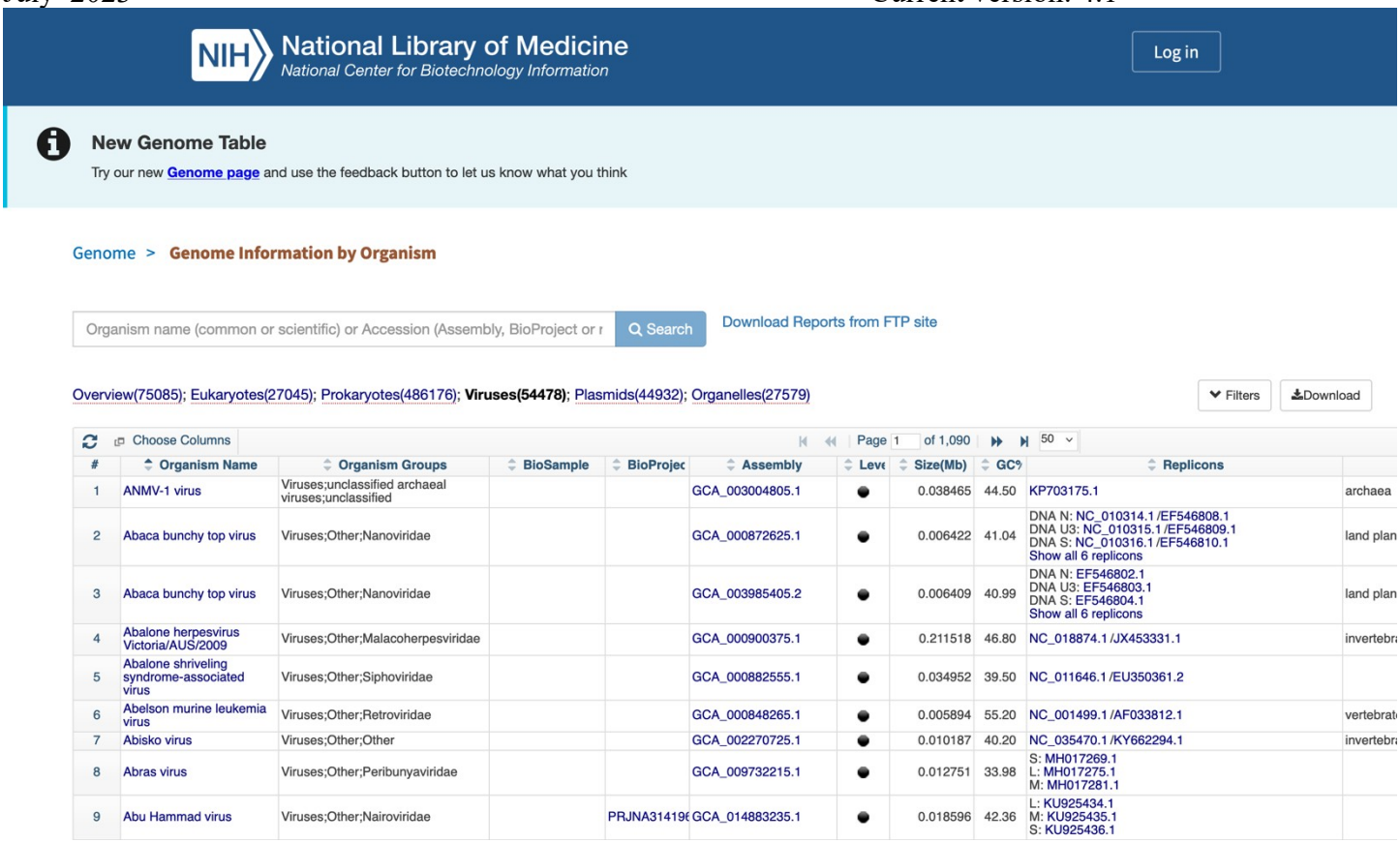



Figure 12

To reduce the number of virus genomes, enter a search term in the Search box. Using SARS as the search term enter SARS in the search box and click the Search button (Figure 13)

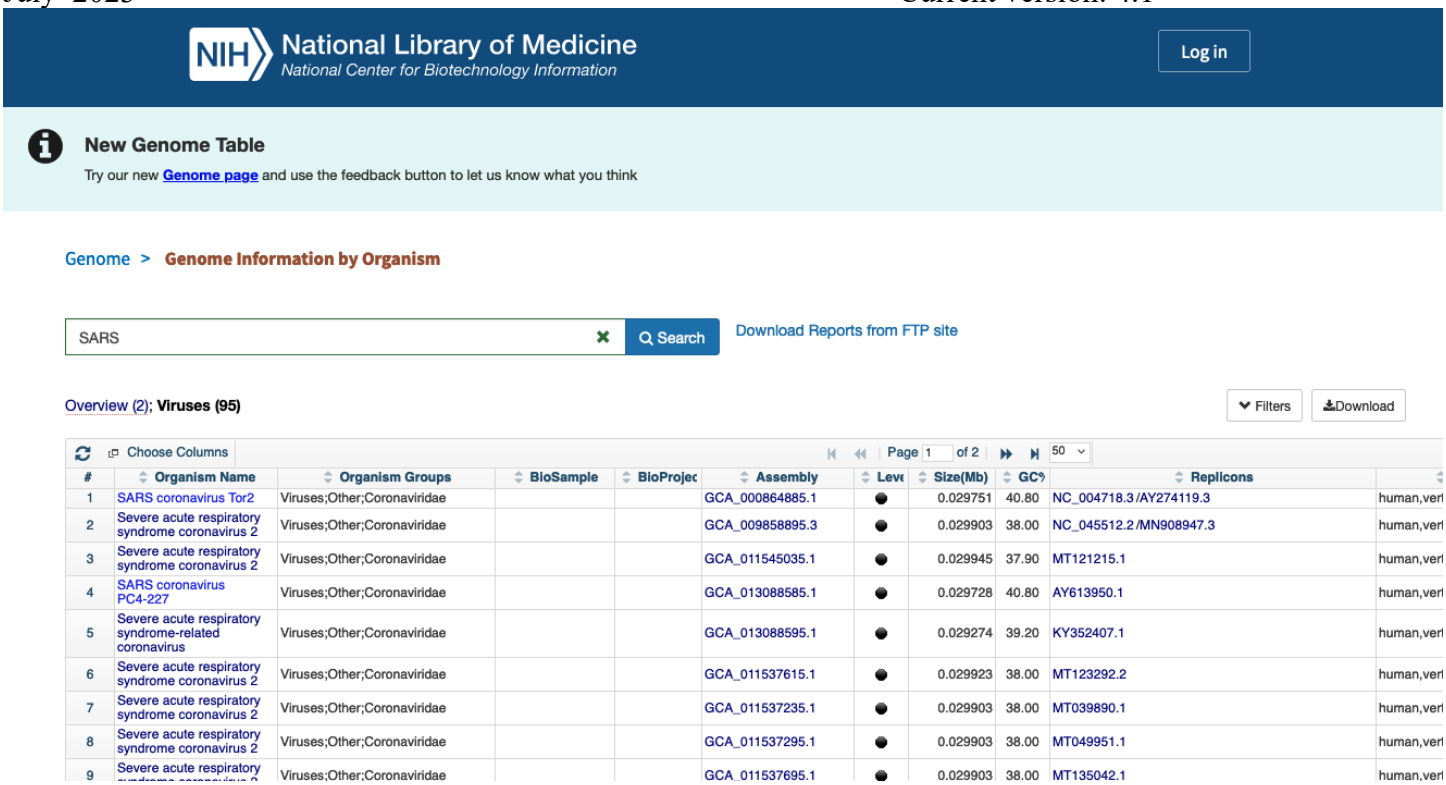



Figure 13

Notice that the number of viruses has changed to 95. That is because now only SARS viruses are counted.

Using the search box to search for specific viruses can be tricky. For instance, if you enter "Monkeypox" you see Figure 14.

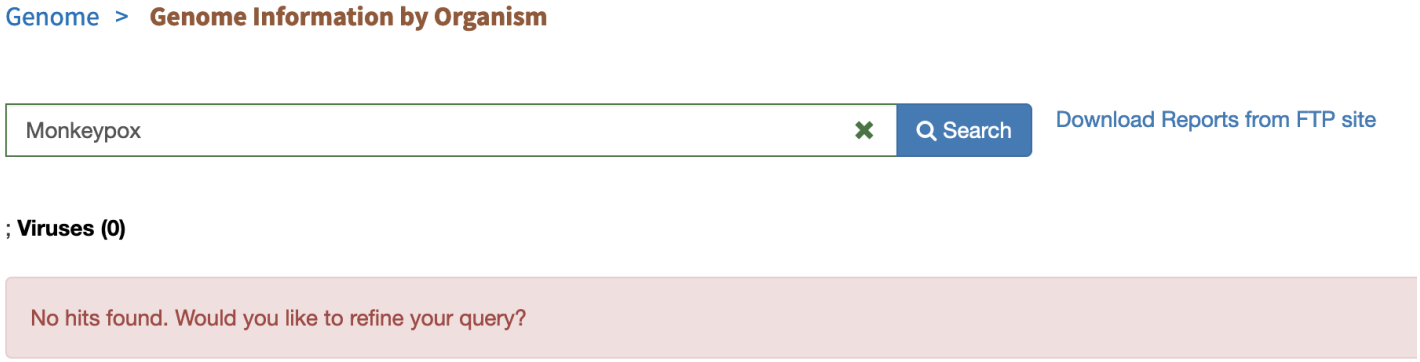



Figure 14

But if you enter "Monkeypox virus" you see Figure 15.


Figure 15

Table 3 shows some of the search terms I have tried and the number of viral genomes found for each

| Search term | Number of virus genomes returned |
|---|---|

| | |
|---|---|
| SARS | 95 |
| Monkeypox virus | 3204 |
| Enterovirus | 353 |
| Enterovirus C | 48 |
| Coronaviridae | 215 |
| Poxviridae | 3872 (includes Monkeypox virus) |
| Vaccinia virus | 128 |

In figure 13 notice the field named "Replicons". That 8th field contains the accession number for each virus.

Whatever search term you used, click the Download button at the right edge of Figure 3 to download a file named viruses.csv to your download folder.

Move that file to a convenient folder. I suggest renaming the file to something like SARS.csv.
SARS.csv, which you never really have to look at, lists 95 SARS viruses

### Parse the SARS.csv file

You need to use ***parseViralNCBIcsv*** to parse that file to generate a file useful for downloading the genomes of interest.

In Terminal run ***parseViralNCBIcsv*** . The command line requires two arguments: -in infileName, -out outfileName with an optional argument -n integer. For this example the command line would be:

```
parseViralNCBIcsv –in SARS.csv –out SARSList.txt.
```

The outfile name can be anything that makes sense to you.

That command line will generate a message similar to this:

```
The file SARS.csv contains 95 genomes.
If there are more than 100 genomes you will need
to use a text editor to manually break up the output
list into chunks of no more than 100 genomes to comply with NCBI
rules against dowwnloading more than 100 genomes at a time

Do you want to limit the number of genomes? (y or n)
```

If you enter 'n' it will produce a SARSList.txt file that lists all 95 genomes.
If you enter 'y' you will see this: `Enter the limit as an integer`
If you then enter '20' t will produce a SARSList.txt file that lists 20 genomes. This is exactly the same result as if you had entered the command line

```
parseViralNCBIcsv –in SARS.csv –out SARSList.txt –n 20
```

If you had used "Enterovirus" as the search term the EnterovirusList.txt file would list 353 genomes. NCBI does not permit downloading more than 100 genomes at a time, so such an EnterovirusList.txt file could not be used directly by ***getFastaGenomes*** to download the 353 genomes in a single batch. You have two options to generate a file that can be used as input to ***getFastaGenomes***.

Option 1 (the easy option): use the -n option on the ***parsseViralNCBIcsv*** command line to limit the number of genomes in the output file; e.g.

```
parseViralNCBIcsv –in Enterovirus.csv –out EnterovirusList.txt –n 100
```

so that the EnterovirusList.txt file includes only the first 100 genomes in Enterovirus.csv. Of course, if you only need to download fewer genomes you can set -n to any smaller integer. The important thing is to not exceed -n 100

Option 2: go ahead and get all 353 genomes in the EnterovirusList.txt file, in which case you will need to use a text editor to break up EnterovirusList.txt into several parts. The command line would be

```
parseViralNCBIcsv –in Enterovirus.csv –out EnterovirusList.txt
```

If you run ***parsseViralNCBIcsv*** without the -n option you will see a message similar to this:

```
The file Enterovirus.csv contains 353 genomes.
If there are more than 100 genomes you will need
to use a text editor to manually break up the output
list into chunks of no more than 100 genomes to comply with NCBI
rules against dowwnloading more than 100 genomes at a time

Do you want to limit the number of genomes? (y or n)
```

If you enter 'n' the resulting EnterovirusList.txt will include all 353 genomes.
You would then use a text editor to make four files: EnterovirusList-A.txt, EnterovirusList-B.txt, etc. by copying and pasting lines from the original EnterovirusList.txt. Each of those files **must** begin with the header line " Strain AccNums" in which the words are separated by a tab.

Returning to the SARSList.txt file, SARSList.txt has two columns: (1) Strain and (2) AccNums, the accession number for that strain.

SARSList.txt will be used as the input file for the program ***getFastaGenomes*** that actually downloads the genomes.

### Download the genomes with *getFastaGenomes*

Begin by creating a new folder *within* the folder where you put the input file (SARSList .txt) to receive the downloaded genome files. We will create the empty folder SARS_Genomes.

The last step is to run the program ***getFastaGenomes***. ***getFastaGenomes*** downloads all of the genomes in the input file. In Terminal navigate to the folder where you put the input file SARSList.txt then enter

```
getFastaGenomes –i SARSList.txt –o SARS_Genomes –e yourname@somewhere.com.
```

You *must* enter your email address for the -e option. That allows NCBI to contact you if you violate their excessive requests limit. That limit is 100 genomes at a time, but also an unspecified number of series of requests. If you actually need to download more than 100 genomes you will need to do separate runs of ***getFastaGenomes*** using different input files. You should do so on weekends or outside USA peak hours, separating the runs by an hour or so.

## XIII Citations

If you use ***kSNP4.1*** please cite:

**kSNP4:** B. G. Hall and J. Nisbet. 2023. Building Phylogenetic Trees from Genome Sequences with kSNP4. Mol. Biol. Evol. 40 https://doi.org/10.1093/molbev/msad235.

**kSNP3:** Gardner, S.N., T. Slezak, and B.G. Hall. 2015. kSNP3.0: SNP detection and phylogenetic analysis of genomes without genome alignment or reference genomes. Bioinformatics **31:** 2877-2878 doi: 10.1093/bioinformatics/btv271.

**kSNP v2:** Gardner, S. N. and B. G. Hall. 2013 When whole-genome alignments just won't work: kSNP v2 software for alignment-free SNP discovery and phylogenetics of hundreds of microbial genomes. PLoS One 8(12): e81760. doi:10.1371/journal.pone.0081760

**kSNP v1**:Gardner SN, Slezak T. Scalable SNP analyses of 100+ bacterial or viral genomes. 2010. J. Forensic Research, 1:107.

***kSNP4.1*** incorporates five third-party programs, which we gratefully acknowledge. These are:

**Jellyfish:** Guillaume Marcais and Carl Kingsford, A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics (2011) 27(6): 764-770 (first published online January 7, 2011) doi:10.1093/bioinformatics/btr011

**FastTree:** Price, M.N., Dehal, P.S., and Arkin, A.P. (2010) FastTree 2 -- Approximately Maximum-Likelihood Trees for Large Alignments. PLoS ONE, 5(3):e9490. doi:10.1371/journal.pone.0009490.

**Parsimonator:** A. Stamatakis distributed under GNU GPL via www.exelixis-.lab.org and https://github.com/stamatak.

**Mummer:** Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg (2004) Versatile and open software for comparing large genomes. Genome Biology **5**: R12

**Consense**: Consense is part of the Phylip suite of programs distributed via http://evolution.genetics.washington.edu/phylip.html Felsenstein, J. 2005. PHYLIP (Phylogeny Inference Package) version 3.6. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle.

## XIV Licenses

***KSNP4.1*** by Jeremiah Nisbet and Barry G. Hall is distributed under the terms of the BSD OPENSOURCE LICENSE.

***kSNP4.1*** is derived from and based upon ***kSNP3*** by Shea N. Gardner and Barry G. Hall

Jellyfish is distributed under the terms of the GNU GENERAL PUBLIC LICENSE. Source code for Jellyfish is freely obtainable at http://www.cbcb.umd.edu/software/jellyfish/.

FastTreeMP is distributed under the terms of the GNU GENERAL PUBLIC LICENSE. Source code for FastTreeMP is freely obtainable at http://www.microbesonline.org/fasttree/ - Install.

Parsimonator is distributed under the terms of the GNU GENERAL PUBLIC LICENSE. Source code for Parsimonator is freely obtainable at http://www.exelixis-lab.org/ by going to the Software tab and scrolling down to Parsimonator.

Mummer is distributed under the terms of the Open Source Initiative License. Source code for Mummer is freely obtainiable at http://mummer.sourceforge.net/

Consense, part of the Phylip suite, is distributed under an open-source license. See http://evolution.genetics.washington.edu/phylip/doc/main.html.

## XV FAQ (Frequently Asked Questions)

Who should I contact for support of kSNP4.1?

Unfortunately there is no "support team" for ***kSNP4.1*** and we can offer only the most limited support. Before asking for support please see the document "Troubleshooting kSNP4.1" and answer all the questions in that document. Put your answers to those question in your request for support. Requests without answers to those questions will be ignored.

If you are still unable to resolve your problem, or if you are confident that your problem comes from a bug, please direct enquiries to barryghall@gmail.com. Because of our many other commitments you should not expect a prompt response to such enquiries.

How can I easily download the genomes I want to analyze with ***kSNP4.1***?

Read section XII "Downloading genome sequences from NCBI.

Should I download the source code or one of the packages of executables?

We strongly encourage users to download one of the packages of executables and to install that package exactly as described in this user guide. Note that we offer no support whatsoever for kSNP4.1 installations of the source code.

Do I have to uninstall ***kSNP v3*** from my computer in order to run ***kSNP4.1***?

Yes. The earlier version can interfere with ***kSNP4.1***. For details see the next question.

How do I uninstall ***kSNP3*** or ***kSNP4pkg***?

To uninstall ***kSNP3*** enter `cd /usr/local` then enter `ls -l` to see list of the files and directories in `/usr/local`. You should see kSNP3 listed. Enter `sudo rm -r kSNP3`. You will be asked for a password. If you don't know the administrator password ask your computer's administrator.
Enter `ls -l` again. kSNP3 should no longer be listed.
Similarly to uninstall kSNP4pkg enter `sudo rm -r kSNP4pkg`

How do I find which SNP loci map to a particular node on a tree?

We have provided a little utility as a bash shell script for exactly this purpose. The utility is named **selectNodeAnnotations4**, and it is documented in section X (10) above.

How much time is required for a kSNP4.1 run?

That depends on the size of the data set, i.e. the number of genomes and the sizes of those genomes; on the number of finished genomes that will be used for annotations, and on the number of SNPs (which you cannot know in advance) and on your computer's CPU. Performance is shown in Figure 14 below. Shorter times are better

The time increases as the cube of the number of genomes. For the graph below the fit to a cubic equation for kSNP4.1 is excellent: $R^2 = 0.999237$

Annotation will increase the run time, the amount depends upon the number of genomes used for annotation; i.e. annotation takes much longer when using 100 genomes to annotate than using 10 genomes to annotate

Figure 14. kSNP4.1, kSNP4 and kSNP3 performance on a System76 computer running Ubuntu 22.04 Linux OS with 32 GB RAM and an Intel Core i7 -1260P processor with 12 cores.

How much RAM does ***kSNP4.1*** require?

That is a function of the data set. We recommend at least 32 GB of RAM, the more the better. 4GB of RAM is not sufficient for meaningful work.

What operating systems has kSNP4.1 been tested on?

Ubuntu22.04 Linux, and Mac OS 10.15.7 (Catalina). It runs about 2.5 times faster on the System76 machine running Ubuntu 22.04 than on a 2017 iMac Pro running OS 10.15.7.

How can I extract the nth locus from the core_SNPs file?

The bash script ***extractNthLocus4*** does exactly what you want. See Section X, the kSNP4.1 Utilities section.

All of the trees have some sort of label at the internal nodes. How can I remove those labels for the purpose of drawing the tree?

The utility ***rmNodeNamesFromTree4*** does exactly what you want. See Section X the kSNP4.1 Utilities.

Are line endings an issue for the genome fasta files?

Computer text files are cursed with at least three alternative characters that indicate the ends of lines: The Unix/Linux/Mac OSX platform recognizes the LF (line-feed) character; the old classic Mac platform uses the CR (carriage-return) character; and the Windows/DOS platform recognizes CRLF (CR followed by LF) as ending a line. kSNP4.1 expects all input files to have Unix line endings.

If you downloaded the genome files from NCBI they should have Unix line endings. If you were given genome files by a colleague some of the files may have Windows or (probably rarely) classic Mac line endings. Those files will not work with kSNP4.1 without modification. To avoid that problem kSNP4.1 checks each genome file for its line ending character. If the file has Windows line endings the program rewrites the files to have Unix line endings, a process that is transparent to the user. If kSNP4.1 finds classic Mac line endings it reports that fact on the screen and also warns the user to change the line endings manually. However, it then continues to process the rest of the data so in most cases the warning will not be noticed. kSNP4.1 winds up simply ignoring those genome sequences, so they are not included in any of the output including tree files. If you notice that some genomes are missing from the output look near the beginning of the log file to see if you overlooked that warning (you did remember to use | tee logfileName on the command line so that the logfile duplicates all screen output, didn't you???).

Can I edit the genome IDs in an input file written by ***MakeKSNP4.1infile*** in the automatic mode?

Absolutely. You can edit the genome IDs however you like, but if you have moved a file you **must** edit the path name accordingly, or kSNP4.1 won't be able to find the genome data.

If I didn't originally run ***kSNP4.1*** with the -annotate option, can I do annotation after the fact?

No. You need to repeat the run with the -annotate option.

Can I use raw-read files in the fastq format for ***kSNP4.1***?

The short answer is, no you cannot. All input files must be in fasta format.
You need to convert the fastq to fasta, using a tool that will convert low quality bases to N. It is really important to turn low-quality bases into N, otherwise the reliability of those raw-read sequences will be too low to trust at all.

A Google search reveals that there are numerous conversion programs available, but most of them require pretty sophisticated computer/programming skills to install and use.

As it turns out that conversion is not at all a trivial matter. Each sequence in a fastq file consists of a header line, a line consisting of the sequence itself, an essentially blank line, and a quality line. The quality line has, for each base on the sequence line, a corresponding character that indicates the quality of that base - essentially the probability that the base has been correctly called. If there were a single standard for what the characters on the quality line mean it would be easy to translate fastq to fasta, substituting N for low-quality bases. Sadly, there is no such single standard. Not only are there different ranges for the quality score itself, the characters corresponding to those scores differ. Solexa/Ilumina 1.0 uses one coding, Ilumina 1.3 uses a different coding, Ilumina 1.5 still a different coding. See https://en.wikipedia.org/wiki/FASTQ_format for a discussion of fastq and quality scores. In practice, that means that you need to know what manufacturer and what version of the software generated the fastq file in order to convert it to fasta while safely converting low-quality bases to N. You will need to consult with the proper manufacturer to solve that problem.

If you have other questions or find bugs, email Barry G. Hall: barryghall@gmail.com.

## Appendix I Suggested text editors

• For Macintosh we recommend the free TextWrangler from BareBones Software (http://www.barebones.com/) or the more sophisticated commercial BBEdit from the same source.

• For some Linux OS, those that use the Gnome desktop GUI, gedit is the default text editor (although it might just be called "Text Editor" in the sidebar). For other versions of Linux download Gedit from https://wiki.gnome.org/Apps/Gedit.

In both cases the Text Editor should be configured to save files with **Unix** line endings, and it is helpful to configure it to show line numbers and not to wrap text.