ngskit4b NGS K-mer Informed Toolkit for Bioinformatics

Release 0.4.6

CAUTION – this document has not been updated to include new functionality added since original cloning of the BioKanga 4.2.0 documentation

Overview

ngskit4b is an integrated single process image integrating a number of subprocesses, each providing bioinformatics task functionality. Having to a single process image instead of multiple independent process images provides assured integration and release compatibility between the various subprocesses; if 'ngskit4b' is loaded then all dependent subprocesses are guaranteed available and these subprocesses will all be at the same release level as they are part of the same process binary image.

ngskit4b can be natively compiled for execution on either Windows (Visual Studio 2017 or 2019) or Linux (GNU tool chain) x64 hosting platforms – build instructions are provided in the source release INSTALL text document.

Subprocesses within 'ngskit4b' are invoked by specifying the subprocess name on the command line immediately following the 'ngskit4b' startup request . Entering 'ngskit4b' with no subprocess specified will return a list of available subprocesses. You can get help on any subprocess by following the subprocess name with the '-h' command parameter –e.g "ngskit4b kalign –h".

Example 'ngskit4b' command invocation

```
>ngskit4b
 Please specify one of the following subprocesses:
 simreads     Generate simulated NGS readsets
 ngsqc        Process NGS reads and report quality scores with compositional distributions
 fasta2nxx    Generate N10..N90 over Fasta sequences
 filter       Filter NGS reads for sequencer errors and/or exact duplicates
 assemb       de Novo assemble filtered reads into contigs
 scaffold     Scaffold de Novo assembled contigs
 index        Generate index over genome assembly or sequences
 kmarkers     NGS reads alignment-less K-mer derived marker sequences generation
 prekmarkers  NGS reads alignment-less prefix K-mer derived marker sequences generation
```

pseudogenome	Concatenate sequences to create pseudo-genome assembly
kalign	Align NGS reads to indexed genome assembly or sequences
pescaffold	Scaffold assembly contigs using PE read alignments
ssr	Identify SSRs in multifasta sequences
maploci	Map aligned reads loci to known features
rnade	RNA-seq differential expression analyser with optional Pearsons generation
gendeseq	Generate tab delimited counts file for input to DESeq or EdgeR
xfasta	Extract fasta sequences from multifasta file
mergeoverlaps	Merge PE short insert overlap reads
snpmarkers	SNP alignment derived marker sequences identification
markerseqs	Generate marker sequences from SNP loci
blitz	Blat like local align genomic sequences
remaploci	Remap alignment loci
filtchrom	Filter SAM/BAM alignments by chromosome
locateroi	Locate and report regions of interest
alignsbs	Alignments bootstrapper
ultras	Identify Utra/Hyper conserved elements
psl2sqlite	Generate SQLite Blat alignment Database from Blat generated PSL alignments
snpm2sqlite	Generate SQLite Marker Database from SNP markers
snps2sqlite	Generate SQLite SNP Database from aligner identified SNPs
de2sqlite	Generate SQLite DE Database from RNA-seq DE
genbioseq	Generate bioseq pre-parsed sequence file
goassoc	GO association inferencing
gengoassoc	Generate biogoassoc pre-indexed GO associations
gengoterms	Generate biogoterms pre-indexed GO terms
hammings	Generate hamming distances for K-mer over sequences
fasta2bed	Generate BED file from fasta containing sequence names and lengths

To obtain parameter help on any subprocess then enter that subprocess name e.g:

>ngskit4b filter -h

Subprocess Overview

Use the 'ngskit4b filter' subprocess request to invoke filtering of raw reads for removal of reads likely to contain sequencing errors.

The 'ngskit4b kmarker' subprocess request will efficiently process indexed NGS reads from multiple cultivars or species directly, without requiring a reference assembly or transcriptome to align against, and identify K-mers of specified length which are unique to a cultivar or species.

To create a pseudo-genome containing many individual short sequences concatenated together which can then be efficiently indexed for post-processing then use the 'ngskit4b pseudogenome' request.

Creation of an index over a targeted genome assembly, transcriptome , or any set of sequences, is requested using the 'ngskit4b index' subprocess request.

Alignments are performed with the 'ngskit4b kalign' subprocess request.

When aligning RNA-seq to genomic DNA then use 'ngskit4b maploci' to map the DNA alignment loci onto known annotated transcribed region loci.

RNA-seq differential expression processing is with the 'ngskit4b rnade' subprocess request.

Marker generation is enabled with the 'ngskit4b snpmarker' subprocess request

Local alignments of long sequences (could be transcripts etc.) to an indexed target aka Blat/Blast can be efficiently achieved with the 'ngskit4b blitz' functionality.

SQLite3 database generation for SNPs, Markers and Differential Expression is enabled with the 'ngskit4b snps2sqlite', 'ngskit4b markers2sqlite' and 'ngskit4b de2sqlite' subprocess requests.

Installation

To install, simply copy the compiled ngskit4b binary image into a directory which is on your executable path. There are a number of user specified parameters specific to each subprocess of the kit4b toolkit. Most of these are optional, and the optional parameters would generally only be required to be specified by the user when targeting some specific analytics issue. In general, the only mandatory parameters are those specifying input datasets and where to write output result sets.

Resource Requirements

The ngskit4b architecture is optimised for execution on modestly resourced hardware. It is multithreaded so as to take advantage of modern multicore CPUs, and processes memory resident datasets which are loaded from disk at process initiation.

To avoid memory paging with consequent disk threshing it is recommended that there should be at least the following amount of physical memory installed: 5x the targeted genome assembly size for genome assemblies (maximum 100Gb), plus 50 bytes per read times the number of reads times the mean read length. Thus assuming that the targeted genome is human, and there are 100 million 100bp reads to align, then the physical memory required to avoid memory paging should be at least 32GB. If the targeted genome was Arabidopsis, with the same number of reads, then the minimum physical memory required is reduced down to approximately 16GB. When aligning to genomes larger than 4Gbp (aka Wheat) then 6x the targeted genome assembly size plus reads allowance will be required.

Disk requirements will depend on a number of factors. The 'ngskit4b index' generated targeted genome assembly suffix array will require disk space for approximately 5x (6x for genomes more than 4Gbp) the genome assembly size, the raw NG short read datasets need to be on an accessible path, and the result set size requirements will be dependent on the output format selected as well as the number of reads to be aligned and accepted alignment rate. BED or CSV result file formats are generally much more compact than the default SAM output format. Note though that if SAM output format is selected, and the file extension has been specified as '.bam', then BAM (compressed with bgzf) and an associated BAI, or CSI if any target sequence more than 512Mbp, index file will be generated. BAM output files are typically less than 20% of the size of the equivalent SAM file containing the same number of alignments and many downstream processing applications will only accept BAM.

Usage Scenario

All ngskit4b toolset components can write progress details to an optional log file which is recommended to always be enabled. The log filename is specified with the '-F' option to each of the toolset components. This log will also be written to the users console allowing the user to observe processing progress.

In the following simple usage scenario, it is assumed that the ngskit4b toolset has been copied into a directory which is on the users path and thus directly executable without any path prefixing. It is further assumed that there exists a directory named 'experiment' into which alignment results are to be generated, this directory contains four subdirectories:

a) 'Logs' into which log files are to be generated

b) 'KR' used by the BioKanga toolset to hold generated resources

c) 'GA' containing the targeted genome assembly multifasta files

d) 'RDS' containing the raw NG reads datasets to be aligned

Thus in a Linux hosting environment the forgoing directory pathing structure would be -

/home/mylogin/experiment # results to this directory

/home/mylogin/experiment/Logs # contains log files

/home/mylogin/experiment/KR # aligner resources

/home/mylogin/experiment/GA # genome assembly

/home/mylogin/experiment/RDS # NG reads dataset

In a Windows hosting environment the equivalent directory structure could be:

c:\experiment # results to this directory

c:\experiment\Logs # contains log files

c:\experiment\KR # aligner resources

c:\experiment\GA # genome assembly

c:\experiment\RDS # NG reads dataset

The following workflow assumes that the user's current directory from which the steps are executed is the 'experiment' directory.

Step 1: Create the suffix array lookup database for the reference genome of interest, which in this scenario will be Arabidopsis with the chromosome fasta files downloaded from TAIR into the 'GA' directory. The generated suffix array database is to be called 'araTha10.sfx', output into the 'KR' subdirectory, and the assembly reference species name is to be saved as being 'araTha10'. Generate the suffix index array with the following command -

```
ngskit4b index -F./Logs/log.txt -i./GA/*.fasta -o./KR/araTha10.sfx -raraTha10
```

Step 2A: Using default parameters now align NG short reads against the targeted reference genome. You will have copied an Illumina short reads dataset into the 'RDS' directory, and these all are in the 'fastq' format named with '.fastq' extension. The generated alignments, by default, will be in sorted SAM format. Align with the following command -

```
ngskit4b align -F./Logs/log.txt -i./RDS/*.fastq -I./KR/araTha10.sfx -oalgn.sam
```

Experimenter now wants to see what the impact would be if a more sensitive alignment was used and because of the large SAM formatted file generated decides to directly generate BAM format alignments -

```
ngskit4b align -F./Logs/log.txt -m1 -i./RDS/*.fastq -I./KR/araTha10.sfx -oalgn_m1.bam
```

ngskit4b Toolkit Options

All ngskit4b sub-processes can be invoked with the '-h' option, e.g. 'ngskit4b kalign -h', to display the list of options accepted by that specific sub-process.

Note: Options and associated parameters can be entered into an option parameter file, one option and its associated parameter per line. To specify usage of this option parameter file to the ngskit4b toolkit sub-process then precede the parameter file name with '@', e.g.

>ngskit4b kalign @myparams.txt

The lists of subprocess options are specified in accompanying text files with the naming convention –

ngskit4b.<subprocess>.txt

e.g. the sub-process alignment options are in the accompanying file 'biokanga.kalign.txt'

SQLite Result Summaries

Most 'ngskit4b' subprocesses allow the user to specify an output SQLite3 database file to which both subprocess invoking parameters and summary results will be written. These summary results are essentially a copy of the summaries generated in the human readable log files but allow for ease of SQL querying and do not require log file parsing. The schema employed is documented in a an accompanying file 'ngskit4b.sql.schema.txt'..

SQLite result parameters:

-q, --sumrslts=<file>	Output results summary to this SQLite3 database file
-w, --experimentname=<str>	Experiment name SQLite3 database file
-W, --experimentdescr=<str>	Experiment description SQLite3 database file

Using SQLite for ad hoc queries

Download and install SQLite3 command shell from http://www.sqlite.org/download.html (look for the 'sqlite-shell-xxxxx' downloads as appropriate for your hosting environment. There are general usage instructions at http://www.sqlite.org/sqlite.html .

Assuming you have down loaded and installed the SQLite3 command shell then you can directly query the SNP/Marker/DE SQLite databases. Following examples are against a SNP database but can, with appropriate table and field name substitutions, also be used to query the Marker and DE databases. In the following examples the SNP database is named 'snps.db' and note that the SQL query statements are terminated with semicolon ';'.

a) To start querying the SNPs database first it must be loaded so at your login shell command prompt enter the following (do not enter the single quote marks and terminate with 'enter' key) :
'sqlite3 snps.db<enter>'

b) View list of SQLite commands, note that commands are prefixed with full stop '.':
'.help<enter'

c) Turn on headers for subsequent query result sets and also show results as CSV
'.headers on<enter>'
'.mode csv<enter>'

d) A simple SQL query to return the total number of called SNPs:
'*SELECT COUNT(*) FROM TblSNPs<enter>'*

e) A complex query where we would like to find minor allelic SNP loci in which the proportion of mismatches to reference at the allelic loci was between 0.4 and 0.6 inclusive:

'SELECT SeqName,Offset,TotCovCnt,TotMMCnt, (CAST(TotMMCnt AS REAL)/TotCovCnt) AS PropAllelic FROM TblSeqs JOIN TblLoci USING (SeqID) JOIN TblSNPs USING (LociID) WHERE PropAllelic BETWEEN 0.40 AND 0.6 ORDER BY PropAllelic DESC;<enter.'

f) To write results to file called 'mycsvfile' then simply precede the SQL query with:
'.output mycsvfile<enter>'

g) To exit from the SQLite shell back to your loin shell:
'.exit<enter>'

Automated SQLite Queries

To automate the SQLite queries there are many scripting language bindings available. An example binding is the native bindings for recent releases of python, with a good introductory tutorial at http://zetcode.com/db/sqlitepythontutorial/ . The following python script demonstrates automating the complex query used in the SQLite shell and writing the results to file:

```
#!/usr/bin/python

import sqlite3 as lite

import csv

import sys

con = lite.connect('testsnps.db')

with open("testout.csv","wb") as csvfile:

   snpwriter = csv.writer(csvfile,delimiter=',',quotechar='"',quoting=csv.QUOTE_MINIMAL)

   with con:

      cur = con.cursor()

      cur.execute("SELECT SeqName,Offset,TotCovCnt,TotMMCnt, (CAST(TotMMCnt AS
REAL)/TotCovCnt) AS PropAllelic FROM TblSeqs JOIN TblLoci USING (SeqID) JOIN TblSNPs USING
(LociID) WHERE PropAllelic BETWEEN 0.40 AND 0.6 ORDER BY PropAllelic DESC")

      col_names = [cn[0] for cn in cur.description]

      rows = cur.fetchall()

      snpwriter.writerow(col_names)

      for row in rows:

         snpwriter.writerow(row)
```