ngless Documentation

Release 1.4.0

NGLess Authors

Jul 20, 2022

Contents

NGLess is a domain-specific language for NGS (next-generation sequencing data) processing.

For questions, you can also use the ngless mailing list.

Note: If you are using NGLess for generating results in a scientific publication, please cite

> *NG-meta-profiler: fast processing of metagenomes using NGLess, a domain-specific language* by Luis Pedro Coelho, Renato Alves, Paulo Monteiro, Jaime Huerta-Cepas, Ana Teresa Freitas, Peer Bork - Microbiome 2019 7:84; https://doi.org/10.1186/s40168-019-0684-8

CHAPTER 1

NG-meta-profiler

For metagenomics profiling, consider using ng-meta-profiler, which is a collection of predefined pipelines developed using NGLess.

CHAPTER 2

NGLess

NGLess is best illustrated by an example:

```
ngless "1.4"
input = paired('ctrl1.fq', 'ctrl2.fq', singles='ctrl-singles.fq')
input = preprocess(input) using |read|:
    read = read[5:]
    read = substrim(read, min_quality=26)
    if len(read) < 31:
        discard

mapped = map(input, reference='hg19')

write(count(mapped, features=['gene']),
        ofile='gene_counts.csv',
        format={csv})
```

CHAPTER 3

Building and installing

See the install page for more information.

CHAPTER 4

Basic functionality

- preprocessing and quality control of FastQ files
- mapping to a reference genome (implemented through bwa by default)
- assembly of contigs
- annotation and summarization of the alignments using reference gene annotations
- much more

Ngless has builtin support for model organisms:

1. Homo sapiens (hg19)
2. Mus Muscullus (mm10)
3. Rattus norvegicus (rn4)
4. Bos taurus (bosTau4)
5. Canis familiaris (canFam2)
6. Drosophila melanogaster (dm3)
7. Caenorhabditis elegans (ce10)
8. Saccharomyces cerevisiae (sacCer3)

and the standard library includes support for mOTUs, metagenomics profiling of marine samples and human gut microbiome samples. We also have standard library modules for helping users upgrading from MOCAT or running many samples (we have used NGLess on projects with >10,000 samples).

NGLess puts a strong emphasis on reproducibility.

CHAPTER 5

Traditional Unix command line usage

`ngless` can be used as a traditional command line transformer utility, using the `-e` argument to pass an inline script on the command line.

The `-p` (or `--print-last`) argument tells ngless to output the value of the last expression to `stdout`.

5.1 Converting a SAM file to a FASTQ file

Extract file reads from a SAM (or BAM) file:

```
$ ngless -pe 'as_reads(samfile("file.sam"))' > file.fq
```

This is equivalent to the full script:

```
ngless "1.0" # <- version declaration, optional on the command line
samcontents = samfile("file.sam") # <- load a SAM/BAM file
reads = as_reads(samcontents) # <- just get the reads (w quality scores)
write(reads, ofname=STDOUT) # <- write them to STDOUT (default format: FASTQ)
```

This only works if the data in the samfile is single ended as we pipe out a single FQ file. Otherwise, you can always do:

```
ngless "1.0"
write(as_read(samfile("file.sam")),
        ofile="output.fq")
```

which will write 3 files: `output.1.fq`, `output.2.fq`, and `output.singles.fq` (the first two for the paired-end reads and the last one for reads without a mate).

5.2 Getting aligned reads from a SAM file as FASTQ file

Building on the previous example. We can add a `select()` call to only output unmapped reads:

```
$ ngless -pe 'as_reads(select(samfile("file.sam"), keep_if=[{mapped}]))' > file.fq
```

This is equivalent to the full script:

```
ngless "1.0" # <- version declaration, optional on the command line
samcontents = samfile("file.sam") # <- load a SAM/BAM file
samcontents = select(samcontents, keep_if=[{mapped}]) # <- select only *mapped* reads
reads = as_reads(samcontents) # <- just get the reads (w quality scores)
write(reads, ofname=STDOUT) # <- write them to STDOUT (default format: FASTQ)
```

5.3 Reading from STDIN

For a true Unix-like utility, the input should be read from standard input. This can be achieved with the special file `STDIN`. So the previous example now reads

```
$ cat file.sam | ngless -pe 'as_reads(select(samfile(STDIN), keep_if=[{mapped}]))' >␣
↪file.fq
```

Obviously, this example would more interesting if the input were to come from another programme (not just `cat`).

Full documentation

Frequently Asked Questions (FAQ)

CHAPTER 6

Authors

- Luis Pedro Coelho (email: coelho@embl.de) (on twitter: @luispedrocoelho)
- Paulo Monteiro
- Renato Alves
- Ana Teresa Freitas
- Peer Bork

6.1 Introduction

6.1.1 Motivation

Nearly all next generation sequence (NGS) applications rely on sequence alignment as the first analysis step. Before the alignment they require some kind of pre-processing of data, that is always dependent on the researcher interest. Our objective is to allow the creation of a pipeline of work for all the first phase of NGS analysis until the point (inclusive) of annotation. We want to do this while achieving the following goals:

- Ease the development of NGS Tools;
- Enable an easy identification of errors;
- Easily reproduce an experiment;
- Easy configuration and execution of workflows;
- Exploit available computational resources.

6.1.2 Target Users

Bioinformaticians working in a wetlab setting. Every serious biological lab in the world now needs to hire at least one. They know programming (at least basic programming), but are not method developers.

The tool can still be useful for more advanced users.

6.1.3 Basic Properties

- The syntax is a pythonesque syntax with Ruby-like blocks.
- The types are statically and strictly.
- Types are implicit, but limited language allows for type inference and checking.
- Quality control is implicit and mandatory (you get it for free)
- Types are domain types (biological).

6.2 Installation

6.2.1 Bioconda (binary)

The recommended way to install NGLess is through bioconda:

```
conda install -c bioconda ngless
```

This will install the most recent released version

Docker

Alternatively, a docker container with NGLess is available at docker hub:

```
docker run -v $PWD:/workdir -w /workdir -it nglesstoolkit/ngless:1.4.0 ngless --
↪version
```

Adapt the mount flags (`-v`) as needed. You can use the `latest` tag to get a more up to date version as well.

6.2.2 Linux (binary)

You can download a statically linked version of NGless 1.4.0.

This should work across a wide range of Linux versions (please report any issues you encounter):

```
curl -L -O https://github.com/ngless-toolkit/ngless/releases/download/v1.4.0/NGLess-
↪v1.4.0-Linux-static-full
chmod +x NGLess-v1.4.0-Linux-static-full
./NGLess-v1.4.0-Linux-static-full
```

This downloaded file bundles bwa, samtools and megahit (also statically linked).

6.2.3 From source

Stack is the simplest way to install the necessary requirements.

The following sequence of commands should download and build the software

```
git clone https://github.com/ngless-toolkit/ngless
cd ngless
make
```

The first time you run this, it will take a while as it will download all dependencies. After this ngless is ready to use and subsequent builds will be much faster.

6.2.4 Make targets

The following are targets in the Makefile.

- `make`: compiles NGLess and haskell dependencies
- `clean`: remove local generated files by compilation
- `check`: run tests
- `bench`: run benchmarks

6.3 NG-meta-profiler

NG-meta-profiler is a collection of predefined pipelines for processing shotgun metagenomes.

1. `human-gut.ngl` for human gut samples
2. `marine.ngl` for marine samples
3. `mouse-gut.ngl` for mouse gut samples
4. `dog-gut.ngl` for dog gut samples
5. `pig-gut.ngl` for pig gut samples

These are predefined, but users are encouraged to adapt them to their specific needs.

6.3.1 INSTALL

1. install ngless
2. install ng-meta-profiler by downloading the appropriate pipeline from github: https://github.com/ngless-toolkit/ng-meta-profiler

6.3.2 USAGE

To use the profiler, select the appropriate script (e.g., `human-gut.ngl` for human gut samples), put all the FastQ files from the same sample in the same directory (`INPUT-DIRECTORY`) with the extension `.fq.gz` or `fastq.gz` and run:

```
ngless human-gut.ngl INPUT-DIRECTORY OUTPUT-DIRECTORY
```

6.4 What's New (History)

6.4.1 Version 1.4.2

Released *21 July 2022*

Bugfixes

- Fix bug with parsing GFF files (it was assumed that _scores_ were always positive)

6.4.2 Version 1.4.1

Released *3 June 2022*

Bugfixes

- Fix bug with *low memory mode*

6.4.3 Version 1.4.0

Released *30 May 2022*

User-visible Improvements

- `write()` now returns the filename used
- `write()` can use multiple threads
- Better error messages in multiple situations
- Add a module for GMGC — Global Microbial Gene Catalogue
- Old `motus` (version 1) module deprecated

Bugfixes

- Update *–install-reference-data* mode to newer URLs, see #107
- Update *–create-reference-pack* mode to newer format (where indices are versioned), see #108
- Do not fail when merging empty files (#113)

Internal improvements

- Better building infrastructure
- Switched to the tasty testing framework
- `assemble()` is now using a more up to date version of megahit, which means that the older versions cannot be run.

6.4.4 Version 1.3.0

Released *28 January 2021*

User-visible improvements

- Adds conversion from string to numbers (int or double) and back
- Better error message if the user attempts to use the non-existent <\> operator (suggest </>)
- Validate `count()` headers on `--validate-only`

Internal improvements

- Switched internal interval structure to interval-int. For users using GFF-style annotation in `count()`, this should result in a significant improvement (less memory, faster performance)
- Use zstd compression for more temporary files

Bugfixes

- Fix cases where sample names contain `/` and `collect()` (issue 141)

6.4.5 Version 1.2.0

Released *12 July 2020*.

User-visible improvements

- Added function load_fastq_directory to the builtin namespace. This was previously available under the `mocat` module, but it had become much more flexible than the original MOCAT version, so it was no longer a descriptive name.
- Better messages in parallel module when there are no free locks.

Internal improvements

- Modules can now specify their annotation as a URL that NGLess downloads on a "as needed" basis: in version 1.1, only FASTA files were supported.
- Memory consumption of count() function has been improved when using GFF files (*ca.* less memory used).
- This one is *hopefully* ***not** user-visible*: Previously, NGLess would ship the Javascript libraries it uses for the HTML viewer and copy them into all its outputs. Starting in v1.2.0, the HTML viewer links to the live versions online.

6.4.6 Version 1.1.1

This is a bugfix release and results should not change. In particular, a sequence reinjection bug was fixed.

6.4.7 Version 1.1.0

User-visible improvements

- Added discard_singles() function.
- Added `include_fragments` option to orf_find().
- The countfile now reorders its input if it is not ordered. This is necessary for correct usage.
- More flexible loading of `functional_map` arguments in count to accept multiple comment lines at the top of the file as produced by eggnog-mapper.
- Added `sense` argument to the count function, generalizing the previous `strand` argument (which is deprecated). Whereas before it was only possible to consider features either to be present on both strands or only on the strand to which they are annotated, now it is also possible to consider them present only on the opposite strand (which is necessary for some strand-specific protocols as they produce the opposite strand).
- Added `interleaved` argument to fastq
- `load_mocat_sample` now checks for mismatched paired samples (#120) - Better messages when collect call could not finish (following discussion on the mailing list)
- Modules can now specify their resources as a URL that NGLess downloads on a "as needed" basis.
- len now works on lists

Internal improvements

- ZSTD compression is available for output and intermediate files use it for reduced temporary space usage (and possibly faster processing).
- Faster check for column headers in `functional_map` argument to count() function: now it is performed *as soon as possible* (including at the top of the script if the arguments are literal strings), thus NGLess can fail faster.
- ZSTD compression is available for output and intermediate files use it for reduced temporary space usage (and possibly faster processing).
- Faster check for column headers in `functional_map` argument to count() function: now it is performed *as soon as possible* (including at the top of the script if the arguments are literal strings), thus NGLess can fail faster.

6.4.8 Version 1.0.1

This is a bugfix release and results should not change.

Bugfixes

- Fix bug with external modules and multiple fastQ inputs.
- Fix bug with resaving input files where the original file was sometimes moved (thus removing it).
- When `bwa` or `samtools` calls fail, show the user the stdout/stderr from these processes (see #121).

6.4.9 Version 1.0

User-visible improvements

- The handling of multiple annotations in count (i.e., when the user requests multiple `features` and/or `subfeatures`) has changed. The previous model caused a few issues (#63, but also mixing with collect(). Unfortunately, this means that scripts asking for the old behaviour in their version declaration are no longer supported if they use multiple features.

6.4.10 Version 0.11

Released March 15 2019 (**0.11.0**) and March 21 2019 (**0.11.1**).

Version 0.11.0 used ZStdandard compression, which was not reliable (the official haskell zstd wrapper has issues). Thus, it was removed in v0.11.1. Using v0.11.0 is **not recommended**.

User-visible improvements

- Module samtools (version 0.1) now includes *samtools_view*
- Add *–verbose* flag to check-install mode (*ngless –check-install –verbose*)
- Add early checks for input files in more situations (#33)
- Support compression in *collect()* output (#42)
- Add smoothtrim() function

Bugfixes

- Fix bug with *orf_find* & *prots_out* argument
- Fix bug in garbage collection where intermediate files were often left on disk for far longer than necessary.
- Fix CIGAR (#92) for select() blocks

Internal improvements

- Switched to diagrams package for plotting. This should make building easier as cairo was often a complicated dependency.
- Update to LTS-13 (GHC 8.6)
- Update minimap2 version to 2.14
- Call bwa/minimap2 with interleaved fastq files. This avoids calling it twice (which would mean that the indices were read twice).
- Avoid leaving open file descriptors after FastQ encoding detection
- Tar extraction uses much less memory now (#77)

6.4.11 Version 0.10.0

Released Nov 12 2018

Bugfixes

- Fixed bug where header was printed even when STDOUT was used
- Fix to lock1's return value when used with paths (#68 - reopen)
- Fixed bug where writing interleaved FastQ to STDOUT did not work as expected
- Fix saving fastq sets with –subsample (issue #85)
- Fix (hypothetical) case where the two mate files have different FastQ encodings

User-visible improvements

- samtools_sort() now accepts by={name} to sort by read name
- Add __extra_megahit_args to assemble() (issue #86)
- arg1 in external modules is no longer always treated as a path
- Added expand_searchdir to external modules API (issue #56)
- Support _F/_R suffixes for forward/reverse in load_mocat_sample
- Better error messages when version is mis-specified
- Support NO_COLOR standard: when `NO_COLOR` is present in the environment, print no colours.
- Always check output file writability (issue #91)
- `paired()` now accepts `encoding` argument (it was documented to, but mis-implemented)

Internal improvements

- NGLess now pre-emptively garbage collects files when they are no longer needed (issue #79)

6.4.12 Version 0.9.1

Released July 17th 2018

- Added NGLess preprint citation

6.4.13 Version 0.9

Released July 12th 2018

User-visible improvements

- Added `allbest()` method to MappedRead.
- NGLess will issue a warning before overwriting an existing file.
- Output directory contains PNG files with basic QC stats
- Added modules for gut gene catalogs of mouse, pig, and dog
- Updated the integrated gene catalog

Internal improvements

- All lock files now are continuously "touched" (i.e., their modification time is updated every 10 minutes). This makes it easier to discover stale lock files.
- The automated downloading of builtin references now uses versioned URLs, so that, in the future, we can change them without breaking backwards compatibility.

6.4.14 Version 0.8.1

Released June 5th 2018

This is a minor release and upgrading is recommended.

Bugfixes

- Fix for systems with non-working locale installations
- Much faster collect calls
- Fixed lock1 when used with full paths (see issue #68)
- Fix expansion of searchpath with external modules (see issue #56)

6.4.15 Version 0.8

Released May 6th 2018

Incompatible changes

- Added an extra field to the FastQ statistics, with the fraction of basepairs that are not ATCG. This means that uses of qcstats must use an up-to-date version declaration.
- In certain cases (see below), the output of count when using a GFF will change.

User-visible improvements

- Better handling of multiple features in a GFF. For example, using a GFF containing "gene_name=nameA,nameB" would result in:

```
  nameA,nameB    1

Now the same results in::

  nameA          1
  nameB          1
```

  This follows after https://git.io/vpagq and the case of *Parent=AF2312,AB2812,abc-3*

- Support for minimap2 as alternative mapper. Import the `minimap2` module and specify the `mapper` when calling map. For example:

```
ngless '0.8'
import "minimap2" version "1.0"

input = paired('sample.1.fq', 'sample.2.fq', singles='sample.singles.fq')
mapped = map(input, fafile='ref.fna', mapper='minimap2')
write(mapped, ofile='output.sam')
```

- Added the </> operator. This can be used to concatenate filepaths. `p0 </> p1` is short for `p0 + "/" + p1` (except that it avoids double forward slashes).
- Fixed a bug in select where in some edge cases, the sequence would be incorrectly omitted from the result. Given that this is a rare case, if a version prior to 0.8 is specified in the version header, the old behaviour is emulated.
- Added bzip2 support to write.
- Added reference argument to count.

Bug fixes

- Fix writing multiple compressed Fastq outputs.
- Fix corner case in select. Previously, it was possible that some sequences were wrongly removed from the output.

Internal improvements

- Faster collect()
- Faster FastQ processing
- Updated to bwa 0.7.17
- External modules now call their init functions with a lock
- Updated library collection to LTS-11.7

6.4.16 Version 0.7.1

Released Mar 17 2018

Improves memory usage in `count()` and the use the `when-true` flag in external modules.

6.4.17 Version 0.7

Released Mar 7 2018

New functionality in NGLess language

- Added max_trim argument to `filter` method of `MappedReadSet`.
- Support saving compressed SAM files
- Support for saving interleaved FastQ files
- Compute number Basepairs in FastQ stats

- Add `headers` argument to samfile function

Bug fixes

- Fix `count`'s mode `{intersection_strict}` to no longer behave as `{union}`
- Fix `as_reads()` for single-end reads
- Fix `select()` corner case

In addition, this release also improves both speed and memory usage.

6.4.18 Version 0.6

Released Nov 29 2017

Behavioural changes

- Changed `include_m1` default in count() function to True

New functionality in NGLess language

- Added orf_find function (implemented through Prodigal) for open reading frame (ORF) predition
- Add qcstats() function to retrieve the computed QC stats.
- Added reference alias for a more human readable name
- Updated builtin referenced to include latest releases of assemblies

New functionality in NGLess tools

- Add –index-path functionality to define where to write indices.
- Allow *citations* as key in external modules (generally better citations information)
- Use multiple threads in SAM->BAM conversion
- Better error checking/script validation

Bug fixes

- Output preprocessed FQ statistics (had been erroneously removed)
- Fix –strict-threads command-line option spelling
- Version embedded megahit binary
- Fixed inconsistency between reference identifiers and underlying files

6.4.19 Version 0.5.1

Released Nov 2 2017

Fixed some build issues

6.4.20 Version 0.5

Released Nov 1 2017

First release supporting all basic functionality.

6.5 List of backwards compatibility fixes

As NGLess uses a version declaration string at the top of script means that NGLess can change its behaviour depending on the version used in the script.

This page documents the fixes that are currently implemented.

6.5.1 NGLess 1.4

- The old `motus` module (which supports only motus version 1, which is a reference from 2013) is not supported any more. Upgrade to the new external motus module if possible.

6.5.2 NGLess 1.1

- The way that CIGAR sequence lengths are computed has changed to match samtools. This implies that the computation of `min_match_size` and `min_identity_pc` have slightly changed.
- Starting in NGLess 1.1, `countfile` reorders its input if necessary.
- The `count` function now accepts multiple lines of comments at the top of its `functional_map` arguments

6.5.3 NGLess 0.8

- The `select` handles a strange corner case differently (it was arguably wrong before, but affects very few reads).

6.5.4 NGLess 0.6

- The `count` function now defaults to `include_minus1` being true.

6.5.5 NGLess 0.5

- The `preprocess` function now modifies its argument. Older code using

```
preprocess(input) using |r|:
    ...
```

is automatically treated as:

```
input = preprocess(input) using |r|:
    ...
```

6.6 Human Gut Metagenomics Functional & Taxonomic Profiling

Note: If you are starting out with NGLess for metagenomics profiling, consider using the predefined pipeline collection, NG-meta-profiler. This tutorial is based on deconstructing a pipeline very similar to those.

In this tutorial, we will analyse a small dataset of human gut microbial metagenomes.

Note: This tutorial is also available as a slide presentation

1. Download the toy dataset

First download all the tutorial data:

```
ngless --download-demo gut-short
```

This will download and expand the data to a directory called `gut-short`.

This is a toy dataset. It is based on real data, but the samples were trimmed so that they contains only 250k paired-end reads.

The dataset is organized in classical MOCAT style, with one sample per directory. NGLess does not require this structure, but this tutorial also demonstrates how to upgrade from your existing MOCAT-based projects.:

```
$ find
./igc.demo.short
./SAMN05615097.short
./SAMN05615097.short/SRR4052022.single.fq.gz
./SAMN05615097.short/SRR4052022.pair.2.fq.gz
./SAMN05615097.short/SRR4052022.pair.1.fq.gz
./SAMN05615096.short
./SAMN05615096.short/SRR4052021.pair.1.fq.gz
./SAMN05615096.short/SRR4052021.single.fq.gz
./SAMN05615096.short/SRR4052021.pair.2.fq.gz
./SAMN05615098.short
./SAMN05615098.short/SRR4052033.pair.2.fq.gz
./SAMN05615098.short/SRR4052033.pair.1.fq.gz
./SAMN05615098.short/SRR4052033.single.fq.gz
./process.ngl
```

The whole script we will be using is there as well (`process.ngl`), so you can immediately run it with:

```
ngless process.ngl
```

The rest of this tutorial is an explanation of the steps in this script.

2. Preliminary imports

To run ngless, we need write a script. We start with a few imports:

```
ngless "1.0"
import "parallel" version "0.6"
import "mocat" version "0.0"
import "motus" version "0.1"
import "igc" version "0.0"
```

These will all be used in the tutorial.

3. Parallelization

We are going to process each sample separately. For this, we use the `lock1` function from the parallel module (which we imported before):

```
samples = readlines('igc.demo.short')
sample = lock1(samples)
```

The `readlines` function reads a file and returns all lines. In this case, we are reading the `tara.demo.short` file, which contains the three samples (`SAMEA2621229.sampled`, `SAMEA2621155.sampled`, and `SAMEA2621033.sampled`).

`lock1()` is a slightly more complex function. It takes a list and *locks one of the elements* and returns it. It always chooses an element which has not been locked before, so you each time you run `NGLess`, you will get a different sample.

Note: When you are using `lock1()` you will need to run `NGLess` multiple times. But you can run multiple instances in parallel.

---

3. Preprocessing

First, we load the data (the FastQ files):

```
input = load_mocat_sample(sample)
```

And, now, we preprocess the data:

```
input = preprocess(input, keep_singles=False) using |read|:
    read = substrim(read, min_quality=25)
    if len(read) < 45:
        discard
```

4. Filter against the human genome

We want to remove reads which map to the human genome, so we first map the reads to the human genome:

```
mapped = map(input, reference='hg19')
```

`hg19` is a built-in reference and the genome will be automatically download it the first time you use it. Now, we discard the matched reads:

```
mapped = select(mapped) using |mr|:
    mr = mr.filter(min_match_size=45, min_identity_pc=90, action={unmatch})
    if mr.flag({mapped}):
        discard
```

The `mapped` object is a set of `mappedreads` (i.e., the same information that is saved in a SAM/BAM file). we use the `as_reads` function to get back to reads:

```
input = as_reads(mapped)
```

Now, we will use the `input` object which has been filtered of human reads.

5. Profiling using the IGC

Note: This section of the tutorial uses the Integrated Gene Catalogue and requires ca. **15GiB** of RAM. Skip to step 9 if your machine does not have this much memory.

After preprocessing, we map the reads to the integrated gene catalog:

```
mapped = map(input, reference='igc', mode_all=True)
```

The line above is the reason we needed to import the `igc` module: it made the `igc` reference available.

Now, we need to `count` the results. This function takes the result of the above and aggregates it different ways. In this case, we want to aggregate by KEGG KOs, and eggNOG OGs:

```
counts = count(mapped,
            features=['KEGG_ko', 'eggNOG_OG'],
            normalization={scaled})
```

7. Aggregate the results

We have done all this computation, now we need to save it somewhere. We will use the `collect()` function to aggregate across all the samples processed:

```
collect(counts,
        current=sample,
        allneeded=samples,
        ofile='igc.profiles.txt')
```

9. Taxonomic profling using mOTUS

Map the samples against the `motus` reference (this reference comes with the motus module we imported earlier):

```
mapped = map(input, reference='motus', mode_all=True)
```

Now call the built-in `count` function to summarize your reads at gene level:

```
counted = count(mapped, features=['gene'], multiple={dist1})
```

To get the final taconomic profile, we call the `motus` function, which takes the gene count table and performs the motus quantification. The result of this call is another table, which we can concatenate with `collect()`:

```
motus_table = motus(counted)
collect(motus_table,
        current=sample,
        allneeded=samples,
        ofile='motus-counts.txt')
```

10. Run it!

This is our script. We save it to a file (`process.ngl` in this example) and run it from the command line:

```
$ ngless process.ngl
```

Note: **You need to run this script once for each sample**. However, this can be done in parallel, taking advantage of high performance computing clusters.

6.6.1 Full script

Here is the full script:

```
ngless "1.0"
import "parallel" version "0.6"
import "mocat" version "0.0"
import "motus" version "0.1"
import "igc" version "0.0"

samples = readlines('igc.demo.short')
sample = lock1(samples)

input = load_mocat_sample(sample)

input = preprocess(input, keep_singles=False) using |read|:
    read = substrim(read, min_quality=25)
    if len(read) < 45:
        discard

mapped = map(input, reference='hg19')

mapped = select(mapped) using |mr|:
    mr = mr.filter(min_match_size=45, min_identity_pc=90, action={unmatch})
    if mr.flag({mapped}):
        discard

input = as_reads(mapped)


mapped = map(input, reference='igc', mode_all=True)

counts = count(mapped,
            features=['KEGG_ko', 'eggNOG_OG'],
            normalization={scaled})

collect(counts,
        current=sample,
        allneeded=samples,
        ofile='igc.profiles.txt')

mapped = map(input, reference='motus', mode_all=True)

counted = count(mapped, features=['gene'], multiple={dist1})

motus_table = motus(counted)
collect(motus_table,
        current=sample,
        allneeded=samples,
        ofile='motus-counts.txt')
```

6.7 Ocean Metagenomics Functional Profiling

Note: If you are starting out with NGLess for metagenomics profiling, consider using the predefined pipeline collection, NG-meta-profiler. This tutorial is based on deconstructing a pipeline very similar to those.

In this tutorial, we will analyse a small dataset of oceanic microbial metagenomes.

Note: This tutorial uses the full Ocean Microbial Reference Gene Catalog presented in Structure and function of the global ocean microbiome Sunagawa, Coelho, Chaffron, et al., Science, 2015

This catalog contains ca. 40 million genes and requires **56GiB** of RAM

1. Download the toy dataset

First download all the tutorial data:

```
ngless --download-demo ocean-short
```

This will download and expand the data to a directory called `ocean-short`.

This is a toy dataset. It is based on real data, but the samples were trimmed so that they contains only 250k paired-end reads.

The dataset is organized in so that each directory contains a sample with multiple fastq files). NGLess does not require this structure, but it is convenient:

```
$ find
./SAMEA2621229.sampled
./SAMEA2621229.sampled/ERR594355_2.short.fq.gz
./SAMEA2621229.sampled/ERR594355_1.short.fq.gz
./SAMEA2621155.sampled
./SAMEA2621155.sampled/ERR599133_1.short.fq.gz
./SAMEA2621155.sampled/ERR599133_2.short.fq.gz
./SAMEA2621033.sampled
./SAMEA2621033.sampled/ERR594391_2.short.fq.gz
./SAMEA2621033.sampled/ERR594391_1.short.fq.gz
./tara.demo.short
./process.ngl
```

The whole script we will be using is there as well (`process.ngl`), so you can immediately run it with:

```
ngless process.ngl
```

The rest of this tutorial is an explanation of the steps in this script.

2. Preliminary imports

To run ngless, we need write a script. We start with a few imports:

```
ngless "1.4"
import "parallel" version "1.0"
import "omrgc" version "1.0"
```

These will all be used in the tutorial.

3. Parallelization

We are going to process each sample separately. For this, we use the `lock1` function from the parallel module (which we imported before):

```
samples = readlines('tara.demo.short')
sample = lock1(samples)
```

The `readlines` function reads a file and returns all lines. In this case, we are reading the `tara.demo.short` file, which contains the three samples (`SAMEA2621229.sampled`, `SAMEA2621155.sampled`, and `SAMEA2621033.sampled`).

`lock1()` is a slightly more complex function. It takes a list and *locks one of the elements* and returns it. It always chooses an element which has not been locked before, so you each time you run _ngless_, you will get a different sample.

4. Preprocessing

First, we load the data (the FastQ files):

```
input = load_fastq_directory(sample)
```

And, now, we preprocess the data:

```
input = preprocess(input, keep_singles=False) using |read|:
    read = substrim(read, min_quality=25)
    if len(read) < 45:
        discard
```

5. Profiling using the OM-RGC

After preprocessing, we map the reads to the ocean microbial reference gene catalog:

```
mapped = map(input, reference='omrgc', mode_all=True)
```

The line above is the reason we needed to import the `omrgc` module: it made the `omrgc` reference available.

```
mapped = select(mapped, keep_if=[{mapped}, {unique}])
```

Now, we need to `count` the results. This function takes the result of the above and aggregates it different ways. In this case, we want to aggregate by KEGG KOs, and eggNOG OGs:

```
counts = count(mapped,
            features=['KEGG_ko', 'eggNOG_OG'],
            normalization={scaled})
```

7. Aggregate the results

We have done all this computation, now we need to save it somewhere. We will use the `collect()` function to aggregate across all the samples processed:

```
collect(counts
        current=sample,
        allneeded=samples,
        ofile='omgc.profiles.txt')
```

8. Run it!

This is our script. We save it to a file (`process.ngl` in this example) and run it from the command line:

```
$ ngless process.ngl
```

Note that we need a large amount (ca. 64GB) of RAM memory to be able to use the OM-RGC. **You also need to run it once for each sample.** However, this can be done in parallel, taking advantage of high performance computing clusters.

6.7.1 Full script

Here is the full script:

```
ngless "1.4"
import "parallel" version "0.0"
import "omrgc" version "1.0"


samples = readlines('tara.demo.short')
sample = lock1(samples)
input = load_fastq_directory(sample)

input = preprocess(input, keep_singles=False) using |read|:
    read = substrim(read, min_quality=25)
    if len(read) < 45:
        discard

mapped = map(input, reference='omrgc', mode_all=True)
mapped = select(mapped, keep_if=[{mapped}, {unique}])
collect(
        count(mapped,
                features=['KEGG_ko', 'eggNOG_OG'],
                normalization={scaled}),
        current=sample,
        allneeded=samples,
        ofile='omgc.profile.txt')
```

6.8 Ocean Metagenomics Assembly and Gene Prediction

In this tutorial, we will analyse a small dataset of oceanic microbial metagenomes.

Note: This tutorial uses the full Ocean Microbial Reference Gene Catalog presented in Structure and function of the global ocean microbiome Sunagawa, Coelho, Chaffron, et al., Science, 2015

---

1. Download the toy dataset

First download all the tutorial data:

```
ngless --download-demo ocean-short
```

We are reusing the same dataset as in the Ocean profiling tutorial. It may be a good idea to read steps 1-4 of that tutorial before starting this one.

2. Preliminary imports

To run ngless, we need write a script. We start with a few imports:

```
ngless "1.4"
```

3. Preprocessing

First, we want to trim the reads based on quality:

```
sample = 'SAMEA2621155.sampled'
input = load_mocat_sample(sample)

input = preprocess(input, keep_singles=False) using |read|:
```

(continues on next page)

(continued from previous page)

```
    read = substrim(read, min_quality=25)
    if len(read) < 45:
        discard
```

4. Assembly and gene prediction

This is now very simply two calls to the function assemble and orf_find:

```
contigs = assemble(input)
write(contigs, ofile='contigs.fna')

orfs = orf_find(contigs)
write(contigs, ofile='orfs.fna')
```

6.8.1 Full script

```
ngless "1.4"

sample = 'SAMEA2621155.sampled'
input = load_mocat_sample(sample)

input = preprocess(input, keep_singles=False) using |read|:
    read = substrim(read, min_quality=25)
    if len(read) < 45:
        discard

contigs = assemble(input)
write(contigs, ofile='contigs.fna')

orfs = orf_find(contigs)
write(contigs, ofile='orfs.fna')
```

6.9 Command line options

Running `ngless --help` will show you all the command line options. Here we describe the most important ones.

Most of the command line options can be set in a configuration file, which defaults to `~/.config/ngless.conf`, but you can set this explicitly:

–config-file ARG Configuration files to parse

The configuraton section of the manual has more information on which options can be set in the configuration file. Whenever an option is set both in the config file and on the command line, then the command line will take priority.

Note that, like configuration files, *command line options* ***do not change the results***. Any change in the results results from changing the NGLess script. Command line options change *how* the results are computed, not what they should be.

6.9.1 Using multiple threads

The main option is called `-j` and sets the number of threads.

-j,–jobs,–threads ARG Nr of threads to use

Using `--strict-threads/--no-strict-threads` controls whether this is a strict or soft upper limit.

–strict-threads strictly respect the –threads option (by default, NGLess will, occasionally, use more threads than specified) –no-strict-threads opposite of –strict-threads

6.9.2 Paths

NGLess can generate large temporary files. By default it uses the system's temporary directory, but it is often a good idea to set it to a path with a lot of free disk space:

```
-t,--temporary-directory ARG
                         Directory where to store temporary files


--search-path ARG        Reference search directories (replace <references> in
                         script)
--index-path ARG         Index path (directory where indices are stored)
```

6.9.3 Debugging

A few options are useful for debugging:

```
-n,--validate-only       Only validate input, do not run script
--subsample              Subsample mode: quickly test a pipeline by discarding
                         99% of the input
--trace                  Set highest verbosity mode
--no-trace               opposite of --trace
--keep-temporary-files   Whether to keep temporary files (default is delete
                         them)
--no-keep-temporary-files
                         opposite of --keep-temporary-files
```

6.9.4 QC Reporting

–create-report create the report directory –no-create-report opposite of –create-report -o,–html-report-directory ARG name of output directory

6.10 Command Line Wrappers

Some of the functionality of NGLess can also be accessed using traditional command-line scripts. These are written in Python and can be installed using Python package management tools:

```
pip install NGLessPy
```

All of the wrappers can install NGLess if passed the `--auto-install` flag.

All of these wrappers also have Common Workflow Language so that they can be used in larger pipelines.

6.10.1 ngless-install.py

This is only supported on Linux

Installs NGLess either for a single user (`$HOME/.local/bin/ngless`) or globally (`/usr/local/bin'`). All the other tools in this package can also install NGLess automatically.

```
usage: ngless-install.py [-h] [-f] [-t TARGET] [-m {user,global}] [--verbose]

optional arguments:
  -h, --help            show this help message and exit
  -f, --force           Install NGLess even if it is already found
  -t TARGET, --target TARGET
                        Output file/path for results
  -m {user,global}, --mode {user,global}
                        Global or user install
  --verbose             Verbose mode
```

6.10.2 ngless-count.py

This is the equivalent of calling the count function from within NGLess:

```
usage: ngless-count.py [-h] -i INPUT -o OUTPUT [-f FEATURES]
                       [-m {dist1,all1,1overN,unique_only}] [--auto-install]
                       [--debug]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        SAM/BAM/CRAM file to count reads on
  -o OUTPUT, --output OUTPUT
                        Output file/path for results
  -f FEATURES, --features FEATURES
                        Feature to count
  -m {dist1,all1,1overN,unique_only}, --multiple {dist1,all1,1overN,unique_only}
                        How to handle multiple mappers
  --auto-install        Install NGLess if not found in PATH
  --debug               Prints the payload before submitting to ngless
```

6.10.3 ngless-map.py

This is the equivalent of calling the map function from within NGLess.

```
usage: ngless-map.py [-h] -i INPUT [-i2 INPUT_REVERSE] [-s INPUT_SINGLES] -o
                     OUTPUT [--auto-install] [--debug]
                     (-r {sacCer3,susScr11,ce10,dm3,gg4,canFam2,rn4,bosTau4,mm10,hg19}
↪ | -f FASTA)

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        FastQ file with reads to map (forward)
  -i2 INPUT_REVERSE, --input-reverse INPUT_REVERSE
                        FastQ file with reads to map (reverse) - if paired end
  -s INPUT_SINGLES, --input-singles INPUT_SINGLES
```

(continues on next page)

(continued from previous page)

```
                        FastQ file with reads to map (singles) - if paired end
                        and unpaired reads exist
  -o OUTPUT, --output OUTPUT
                        Output file/path for results
  --auto-install        Install NGLess if not found in PATH
  --debug               Prints the payload before submitting to ngless
  -r {sacCer3,susScr11,ce10,dm3,gg4,canFam2,rn4,bosTau4,mm10,hg19}, --reference
↪{sacCer3,susScr11,ce10,dm3,gg4,canFam2,rn4,bosTau4,mm10,hg19}
                        Map against a builtin reference
  -f FASTA, --fasta FASTA
                        Map against a given fasta file (will be indexed if
                        index is not available)
```

6.10.4 ngless-mapstats.py

This is the equivalent of calling the mapstats function from within NGLess. This will take a SAM/BAM file as input and produce some simple statistics.

```
usage: ngless-mapstats.py [-h] -i INPUT -o OUTPUT [--auto-install] [--debug]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        SAM/BAM/CRAM file filter
  -o OUTPUT, --output OUTPUT
                        Output file/path for results
  --auto-install        Install NGLess if not found in PATH
  --debug               Prints the payload before submitting to ngless
```

6.10.5 ngless-select.py

This is the equivalent of calling the select function from within NGLess:

```
usage: ngless-select.py [-h] -i INPUT -o OUTPUT -a {keep_if,drop_if} -c
                        {mapped,unmapped,unique}
                        [{mapped,unmapped,unique} ...] [--auto-install]
                        [--debug]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        SAM/BAM/CRAM file filter
  -o OUTPUT, --output OUTPUT
                        Output file/path for results
  -a {keep_if,drop_if}, --action {keep_if,drop_if}
                        Whether to keep or drop when condition are met
  -c {mapped,unmapped,unique} [{mapped,unmapped,unique} ...], --conditions {mapped,
↪unmapped,unique} [{mapped,unmapped,unique} ...]
                        One or more conditions to filter on
  --auto-install        Install NGLess if not found in PATH
  --debug               Prints the payload before submitting to ngless
```

6.10.6 ngless-trim.py

This is equivalent of calling the preprocess function trimming the reads (with either substrim or endstrim depending on the arguments passed. Finally, any (trimmed) reads which are not of a minimum length are discard.

```
usage: ngless-trim.py [-h] -i INPUT -o OUTPUT -m {substrim,endstrim} -q
                      MIN_QUALITY [-d DISCARD] [--auto-install] [--debug]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        FastQ file with reads to trim
  -o OUTPUT, --output OUTPUT
                        Output file/path for results
  -m {substrim,endstrim}, --method {substrim,endstrim}
                        Which trimming method to use
  -q MIN_QUALITY, --min-quality MIN_QUALITY
                        Minimum quality value
  -d DISCARD, --discard DISCARD
                        Discard if shorted than
  --auto-install        Install NGLess if not found in PATH
  --debug               Prints the payload before submitting to ngless
```

6.10.7 ngless-unique.py

This is the equivalent of calling the count function from within NGLess:

```
usage: ngless-unique.py [-h] -i INPUT -o OUTPUT [-c MAX_COPIES]
                        [--auto-install] [--debug]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        FastQ file to filter
  -o OUTPUT, --output OUTPUT
                        Output file/path for results
  -c MAX_COPIES, --max-copies MAX_COPIES
                        Max number of duplicate copies to keep
  --auto-install        Install NGLess if not found in PATH
  --debug               Prints the payload before submitting to ngless
```

6.11 NGLess one liners

`ngless` can be used as a traditional command line transformer utility, using the `-e` argument to pass an inline script on the command line.

The `-p` (or `--print-last`) argument tells ngless to output the value of the last expression to *stdout*.

6.11.1 Converting a SAM file to a FASTQ file

```
$ ngless -pe 'as_reads(samfile("file.sam"))' > file.fq
```

This is equivalent to the full script:

```
ngless "0.8" # <- version declaration, optional on the command line
samcontents = samfile("file.sam") # <- load a SAM/BAM file
reads = as_reads(samcontents) # <- just get the reads (w quality scores)
write(reads, ofname=STDOUT) # <- write them to STDOUT (default format: FASTQ)
```

This only works if the data in the samfile is single ended as we pipe out a single FQ file. Otherwise, you can always do:

```
ngless "0.8"
write(as_read(samfile("file.sam")),
        ofile="output.fq")
```

which will write 3 files: `output.1.fq`, `output.2.fq`, and `output.singles.fq` (the first two for the paired-end reads and the last one for reads without a mate).

6.11.2 Getting aligned reads from a SAM file as FASTQ file

Building on the previous example. We can add a `select()` call to only output unmapped reads:

```
$ ngless -pe 'as_reads(select(samfile("file.sam"), keep_if=[{mapped}]))' > file.fq
```

This is equivalent to the full script:

```
ngless "0.8" # <- version declaration, optional on the command line
samcontents = samfile("file.sam") # <- load a SAM/BAM file
samcontents = select(samcontents, keep_if=[{mapped}]) # <- select only *mapped* reads
reads = as_reads(samcontents) # <- just get the reads (w quality scores)
write(reads, ofname=STDOUT) # <- write them to STDOUT (default format: FASTQ)
```

6.11.3 Reading from STDIN

For a true Unix-like utility, the input should be read from standard input. This can be achieved with the special file `STDIN`. So the previous example now reads:

```
$ cat file.sam | ngless -pe 'as_reads(select(samfile(STDIN), keep_if=[{mapped}]))' >␣
↪file.fq
```

Obviously, this example would more interesting if the input were to come from another programme (not just `cat`).

6.12 Preprocessing FastQ Data

Preprocessing FastQ files consists of quality trimming and filtering of reads as well as (possible) elimination of reads which match some reference which is not of interest.

6.12.1 Quality-based filtering

Filtering reads based on quality is performed with the `preprocess` function, which takes a block of code. This block of code will be executed for each read. For example:

```
ngless "0.8"

input = fastq('input.fq.gz')

input = preprocess(input) using |r|:
    r = substrim(r, min_quality=20)
    if len(r) < 45:
        discard
```

If it helps you, you can think of the `preprocess` block as a `foreach` loop, with the special keyword `discard` that removes the read from the collection. Note that the name `r` is just a variable name, which you choose using the `|r|` syntax.

Within the preprocess block, you can modify the read in several ways:

- you can trim it with the indexing operator: `r[trim5:]` or `r[:-trim3]`
- you can call `substrim`, `endstrim` or `smoothtrim` to trim the read based on quality scores. `substrim` finds the longest substring such that all bases are above a minimum quality (hence the name, which phonetically combines substring and trim). `endstrim` chops bases off the ends and `smoothtrim` averages quality scores using a sliding window before applying `substrim`.
- you can test for the length of the sequence (before or after trimming). For this, you use the `len` function (see example above).
- you can test for the average quality score (using the `avg_quality()` method).

You can combine these in different ways. For example, the behaviour of the fastx quality trimmer can be recreated as:

```
preprocess(input) using |r|:
    r = endstrim(r, min_quality=20)
    if r.fraction_at_least(20) < 0.5:
        discard
    if len(r) < 45:
        discard
```

Handling paired end reads

When your input is paired-end, the preprocess call above will handle each mate independently. Three things can happen:

1. both mates are discarded,
2. both mates are kept (i.e., not discarded),
3. one mate is kept, the other discarded.

The only question is what to do in the third case. By default, the `preprocess` call keep the mate turning the read into an unpaired read (a single), but you can change that behaviour by setting the `keep_singles` argument to `False`:

```
preprocess(input, keep_singles=False) using |r|:
    r = substrim(r, min_quality=20)
    if len(r) < 45:
        discard
```

Now, the output will consist of only paired-end reads.

6.12.2 Filtering reads matching a reference

It is often also a good idea to match reads against some possible contaminant database. For example, when studying the host associated microbiome, you will often want to remove reads matching the host. It is almost always a good to at least check for human contamination (during lab handling).

For this, you map the reads against the human genome:

```
mapped_hg19 = map(input, reference='hg19')
```

Now, `mapped_hg19` is a set of mapped reads. Mapped reads are reads, their qualities, plus additional information of how they matched. Mapped read sets are the internal ngless representation of SAM files.

To filter the set, we will `select`. Like `preprocess`, `select` also uses a block for the user to specify the logic:

```
mapped_hg19 = select(mapped_hg19) using |mr|:
    mr = mr.filter(min_match_size=45, min_identity_pc=90, action={unmatch})
    if mr.flag({mapped}):
        discard
```

We first set a minimum match size and identity percentage to avoid spurious hits. **We keep the reads** but **unmatch** them (i.e., we clear any information related to a match). Then, we discard any reads that match by checking the flag `{mapped}`.

Finally, we convert the mapped reads back to simple reads using the `as_reads` function (this discards the matching information):

```
input = as_reads(mapped_hg19)
```

Now, `input` can be passed to the next step in the pipeline.

6.13 NGLess Builtin Functions

These are the built-in NGLess functions. Make sure to check the standard library as well.

6.13.1 fastq

Function to load a FastQ file:

```
in = fastq('input.fq')
```

Argument:

String

Return:

ReadSet

Arguments by value:

Name	Type	Required	Default Value
encoding	Symbol ({auto}, {33}, {64}, {sanger}, {solexa})	no	{auto}
interleaved	Bool	no	False

Possible values for `encoding` are:

- `{sanger}` or `{33}` assumes that the file is encoded using sanger format. This is appropriate for newer Illumina outputs.
- `{solexa}` or `{64}` assumes that the file is encoded with a 64 offset. This is used for older Illumina/Solexa machines.
- `{auto}`: use auto detection. This is the default.

If `interleaved` is `True`, then the input is assumed to be interleaved. This means that paired-end reads are represented by each mate being adjacent in the file with the same identifier (if the identifiers end with `/1` and `/2`, but are otherwise identical, this is still considered a match). Thus, an interleaved file can contain both paired-end and single-end reads.

When loading a data set, quality control is carried out and statistics can be visualised in a graphical user interface (GUI). Statistics calculated are:

- percentage of guanine and cytosine (%GC)
- number of sequences
- minimum/maximum sequence length
- mean, median, lower quartile and upper quality quartile for each sequence position

If not specified, the encoding is guessed from the file.

Gzip and bzip2 compressed files are transparently supported (determined by file extension, `.gz` and `.bz2` for gzip and bzip2 respectively).

6.13.2 paired

Function to load a paired-end sample, from two FastQ files:

```
in = paired('input.1.fq', 'input.2.fq', singles='input.3.fq')
```

`paired()` is an exceptional function which takes **two** unnamed arguments, specifying the two read files (first mate and second mate) and an optional `singles` file (which contains unpaired reads).

Argument:

String, String

Return:

ReadSet

Arguments by value:

Name	Type	Required	Default Value
encoding	Symbol ({auto}, {33}, {64}, {sanger}, {solexa})	no	{auto}
singles	String	no	•

The `encoding` argument has the same meaning as for the `fastq()` function:

- `{sanger}` or `{33}` assumes that the file is encoded using sanger format. This is appropriate for newer Illumina outputs.
- `{solexa}` or `{64}` assumes that the file is encoded with a 64 offset. This is used for older Illumina/Solexa machines.
- `{auto}`: use auto detection. This is the default.

6.13.3 load_fastq_directory

New in version NGLess: 1.2 Previously, this function was available in the `mocat` module as `load_mocat_sample`. Now, it is a builtin function. Even though the concept originated with MOCAT, this function is now more flexible than the original MOCAT implementation.

This function takes a directory name and returns a set of reads by scanning the directory for (compressed) FastQ files. This is slightly more flexible than MOCAT2 in terms of the patterns in matches. In particular, the following extensions are accepted:

- `fq`
- `fq.gz`
- `fq.bz2`
- `fastq`
- `fastq.gz`
- `fastq.bz2`

Paired-end reads are assumed to be split into two files, with matching names with `.1`/`.2` appended. `_1`/`_2` as is used by the European Nucleotide Archive (ENA) is also accepted.

If paired-end reads have been pre-filtered, an unpaired/single file is often available. `load_fastq_directory` recognizes the suffix `single`. In the following example, all three files are read as one group:

```
sample
├── sample.pair.1.fq.gz
├── sample.pair.2.fq.gz
└── sample.single.fq.gz
```

Arguments by value:

Name	Type	Required	Default Value
name	String	no	“”

Argument

String (directory path)

Returns

ReadSet

6.13.4 group

Groups a list of ReadSet objects into a single ReadSet:

```
rs1 = paired('data0.1.fq.gz', 'data0.2.fq.gz')
rs2 = paired('data1.1.fq.gz', 'data1.2.fq.gz')
rs = group([rs1, rs2], name='input')
```

Arguments by value:

Name	Type	Required	Default Value
name	String	no	“”

Argument

List of ReadSet

Returns

ReadSet

6.13.5 samfile

Loads a SAM file:

```
s = samfile('input.sam')
```

This function takes no keyword arguments. BAM files are also supported (determined by the filename), as are `sam.gz` files.

Returns

MappedReadSet

Arguments by value:

Name	Type	Required	Default Value
name	String	no	•
header	String	no	•

New in version 0.7: The `header` argument was added in version 0.7

- The `name` argument names the group (for `count()`, for example).
- The `headers` argument can be used if the SAM headers are kept in a separate file.

6.13.6 qcstats

New in version 0.6: This functionality was not available prior to 0.6

Returns the auto-computed statistics:

```
write(qcstats({fastq}), ofile='fqstats.txt')
```

Returns

CountsTable

Argument

Defines what type of statistics to return. Currently, two options are available

- `{fastq}`: FastQ statistics
- `{mapping}`: Mapping statistics

6.13.7 countfile

Loads a TSV file:

```
c = countfile('table.tsv')
```

This function takes no keyword arguments. If the filename ends with ".gz", it is assumed to be a gzipped file.

Returns

CountTable

6.13.8 as_reads

Converts from a `MappedReadSet` to a `ReadSet`:

```
reads = as_reads(samfile('input.sam'))
```

6.13.9 discard_singles

New in version NGLess: 1.1

Throws away unpaired reads from a `ReadSet`:

```
reads = discard_singles(reads)
```

Argument

ReadSet

Returns

ReadSet

6.13.10 unique

Function that given a set of reads, returns another which only retains a set number of copies of each read (if there are any duplicates). An example:

```
input = unique(input, max_copies=3)
```

Argument:

ReadSet

Return:

ReadSet

Arguments by value:

Name	Type	Required	Default Value
max_copies	Integer	no	2

The optional argument **max_copies** allows to define the number of tolerated copies (default: 2).

Two short reads with the same nucleotide sequence are considered copies, independently of quality and identifiers.

This function is currently limited to single-end samples.

6.13.11 preprocess

This function executes the given block for each read in the ReadSet. Unless the read is **discarded**, it is transferred (after transformations) to the output. For example:

```
inputs = preprocess(inputs) using |read|:
    read = read[3:]
```

Argument:

ReadSet

Return:

ReadSet

Arguments by value:

Name	Type	Required	Default Value
keep_singles	bool	no	true

When a paired-end input is being preprocessed in single-mode (i.e., each mate is preprocessed independently, it can happen that on eof the mates is discarded, while the other is kept). The default is to collect these into the singles pile. If `keep_singles` if false, however, they are discarded.

This function also performs quality control on its output.

6.13.12 map

The function map, maps a ReadSet to reference. For example:

```
mapped = map(input, reference='sacCer3')
mapped = map(input, fafile='ref.fa')
```

Argument:

ReadSet

Return:

MappedReadSet

Arguments by value:

Name	Type	Required	Default Value
reference	String	no	•
fafile	String	no	•
block_size_megabases	Integer	no	•
mode_all	Bool	no	•
__extra_args	[String]	no	[]

The user must provide either a path to a FASTA file in the `fafile` argument or the name of a builtin reference using the `reference` argument. The `fafile` argument supports search path expansion.

A list of datasets provided by NGLess can be found at *Available Reference Genomes*.

To use any of these, pass in the name as the reference value:

```
mapped_hg19 = map(input, reference='hg19')
```

NGLess does not ship with any of these datasets, but they are downloaded lazily: i.e., the first time you use them, NGLess will download and cache them. NGLess will also index any database used the first time it is used.

The option `block_size_megabases` turns on low memory mode (see the corresponding section in the mapping documentation)

The option `mode_all=True` can be passed to include all alignments of both single and paired-end reads in the output SAM/BAM.

Strings passed as `__extra_args` will be passed verbatim to the mapper.

6.13.13 mapstats

Computes some basic statistics from a set of mapped reads (number of reads, number mapped, number uniquely mapped).

Argument

MappedReadSet

Return

CountTable

6.13.14 select

select filters a MappedReadSet. For example:

```
mapped = select(mapped, keep_if=[{mapped}])
```

Argument:

MappedReadSet

Return:

MappedReadSet

Arguments by value:

Name	Type	Required	Default Value
keep_if	[Symbol]	no	•
drop_if	[Symbol]	no	•
paired	Bool	no	true

At least one of `keep_if` or `drop_if` should be passed, but not both. They accept the following symbols:

- `{mapped}`: the read mapped
- `{unmapped}`: the read did not map
- `{unique}`: the read mapped to a unique location

If `keep_if` is used, then reads are kept if they pass **all the conditions**. If `drop_if` they are discarded if they fail to **any condition**.

By default, `select` operates on a paired-end read as a whole. If `paired=False` is passed, however, then link between the two mates is not considered and each read is processed independently.

6.13.15 count

Given a file with aligned sequencing reads (MappedReadSet), `count()` will produce a counts table depending on the arguments passed. For example:

```
counts = count(mapped, min=2, mode={union}, multiple={dist1})
```

Argument:

MappedReadSet

Return:

CountTable

Arguments by value:

Name	Type	Required	Default value
gff_file	String	no*	•
functional_map	String	no*	•
features	[String]	no	'gene'
subfeatures	[String]	no	•
mode	Symbol	no	{union}
multiple	Symbol	no	{dist1}
sense	Symbol	no	{both}
normalization	Symbol	no	{raw}
include_minus1	Bool	no	true
min	Integer	no	0
discard_zeros	Bool	no	false
reference	String	no	""

If the features to count are `['seqname']`, then each read will be assigned to the name of reference it matched and only an input set of mapped reads is necessary. For other features, you will need extra information. This can be passed using the `gff_file` or `functional_map` arguments. If you had previously used a `reference` argument for the `map()` function, then you can also leave this argument empty and NGLess will use the corresponding annotation file.

The `gff_file` and `functional_map` arguments support search path expansion.

The `functional_map` should be a tab-separated file where the first column is the sequence name and the other columns are the annotations. This is often used for gene catalogues and can be produced by eggnog-mapper.

`features`: which features to count. If a GFF file is used, this refers to the "features" field.

`subfeatures`: this is useful in GFF-mode as the same feature can encode multiple attributes (or, in NGLess parlance, "subfeatures"). By default, NGLess will look for the `"ID"` or `"gene_id"` attributes.

`mode` indicates how to handle reads that (partially) overlap one or more features. Possible values for `mode` are `{union}`, `{intersection_non_empty}` and `{intersection_strict}` (default: `{union}`). For every position of a mapped read, collect all features into a set. These sets of features are then handled in different modes.

- `{union}` the union of all the sets. A read is counted for every feature it overlaps.
- `{intersection_non_empty}` the intersection of all non-empty sets. A read is only counted for features it exclusively overlaps, even if partially.
- `{intersection_strict}` the intersection of all the sets. A read is only counted if the entire read overlaps the same feature(s).

Consider the following illustration of the effect of different `mode` options:

```
Reference ************************
Feature A      =======
Feature B            ===========
Feature C                 ========
Read_1       -----
Read_2            -----
Read_3                    -----
Position     12345 12345  12345
```

(continues on next page)

(continued from previous page)

```
Read position          1    2    3    4    5
Read_1 feature sets    -    -    A    A    A
Read_2 feature sets    A    A  A,B    B    B
Read_3 feature sets  B,C  B,C  B,C  B,C  B,C

           union  intersection_non_empty  intersection_strict
Read_1         A                      A                    -
Read_2     A & B                      -                    -
Read_3     B & C                  B & C                B & C
```

How to handle multiple mappers (inserts which have more than one "hit" in the reference) is defined by the `multiple` argument:

- `{unique_only}`: only use uniquely mapped inserts
- `{all1}`: count all hits separately. An insert mapping to 4 locations adds 1 to each location
- `{1overN}`: fractionally distribute multiple mappers. An insert mapping to 4 locations adds 0.25 to each location
- `{dist1}`: distribute multiple reads based on uniquely mapped reads. An insert mapping to 4 locations adds to these in proportion to how uniquely mapped inserts are distributed among these 4 locations.

The argument `sense` should be used when the data are strand-specific and determines which strands should be considered:

- `{both}` (default): a read is considered overlapping with a feature independently of whether maps to the same or the opposite strand.
- `{sense}`: a read has to map to the same strand as the feature to be considered overlapping.
- `{antisense}`: a read has to map to the **opposite** strand to be considered overlapping.

If you have strand-specific data, then `{sense}` is probably appropriate, but with some protocols `{antisense}` is actually the correct version.

The following illustration exemplifies how counting would be performed.

Note: before version **1.1**, there was an argument `strand` which was either `True` or `False` mapping to `{sense}` and `{both}` respectively. `strand` is still supported, but deprecated.

`min` defines the minimum amount of overlaps a given feature must have, at least, to be kept (default: 0, i.e., keep all counts). If you just want to discard features that are exactly zero, you should set the `discard_zeros` argument to True.

`normalization` specifies if and how to normalize to take into account feature size:

- `{raw}` (default) is no normalization
- `{normed}` is the result of the `{raw}` mode divided by the size of the feature
- `{scaled}` is the result of the `{normed}` mode scaled up so that the total number of counts is identical to the `{raw}` (within rounding error)
- `{fpkm}` is *fragments per 1000 bp per million fragments*, so it is normalized by both the size of the feature and the number of fragments.

Unmapped inserts are included in the output if `{include_minus1}` is true (default: `False`).

New in version 0.6: Before version 0.6, the default was to **not** include the -1 fraction.

6.13.16 substrim

Given a read finds the longest substring, such that all bases are of at least the given quality. The name is a constraction of "substring trim". For example:

```
read = substrim(read, min_quality=25)
```

Argument:

ShortRead

Return:

ShortRead

Arguments

Name	Type	Required	Default Value
min_quality	Integer	yes	

`min_quality` parameter defines the minimum quality accepted.

6.13.17 endstrim

Given a read, trim from both ends (5' and 3') all bases below a minimal quality. For example:

```
read = endstrim(read, min_quality=25)
```

Argument:

ShortRead

Return:

ShortRead

Arguments

Name	Type	Required	Default Value
min_quality	Integer	yes	

`min_quality` parameter defines the minimum quality value.

6.13.18 smoothtrim

This trims with the same algorithm as substrim but uses a sliding window to average base qualities. Quality scores are returned to their original value after trimming. For example:

```
read = smoothtrim(read, min_quality=15, window=3)
```

Quality values of bases at the edges of each read are repeated to allow averaging with quality centered on each base. For instance a read:

```
                                  left pad |--|        |--| right pad
Sequence   A  T  C  G    with a window     A  A  T  C  G  G
Quality   28 25 14 12  of size 3 becomes  28 28 25 14 12 12
```

and is smoothed:

```
Seq         A  A  T  C  G  G   smoothed quality   A  T  C  G
Qual       28 28 25 14 12 12         --->        27 22 17 13
Windows     |-----|            (28 + 28 + 25) / 3 = 27     ^
 ...           |-----|         (28 + 25 + 14) / 3 = 22     |
                  |-----|      (25 + 14 + 12) / 3 = 17     |
                     |-----|   (14 + 12 + 12) / 3 = 13 ----+
```

at which point `substrim` is applied for trimming.

If an even number is given as the window size (e.g. `window=4`), the left pad is 1 unit smaller than the right and scores are rounded to the nearest integer:

```
                                  left pad |--|        |-----| right pad
Sequence   A  T  C  G    with a window     A  A  T  C  G  G  G
Quality   28 25 14 12  of size 4 becomes  28 28 25 14 12 12 12
```

and is smoothed:

```
Seq         A  A  T  C  G  G  G   smoothed quality   A  T  C  G
Qual       28 28 25 14 12 12 12         --->        24 20 16 12
Windows     |--------|          (28 + 28 + 25 + 14) / 4 = 24  ^
 ...           |--------|       (28 + 25 + 14 + 12) / 4 = 20  |
                  |--------|    (25 + 14 + 12 + 12) / 4 = 16  |
                     |--------| (14 + 12 + 12 + 12) / 4 = 12 -+
```

Argument:

ShortRead

Return:

ShortRead

Arguments

Name	Type	Required	Default Value
min_quality	Integer	yes	
window	Integer	yes	

`min_quality` parameter defines the minimum quality accepted for the sub-sequence. `window` parameter defines the number of bases to average over.

6.13.19 write

Writes an object to disk.

Argument:

Any

Return:

New in version NGLess: 1.4 Prior to version 1.4, `write()` returned nothing

String: the file name used

Arguments by value:

Name	Type	Required	Default Value
ofile	String	yes	•
format	String	no	•
format_flags	[Symbol]	no	[]
comment	String	no	•
auto_comments	String	no	•

The argument `ofile` is where to write the content.

The output format is typically determined from the `ofile` extension, but the `format` argument overrides this. Supported formats:

- CountsTable: `{tsv}` (default) or `{csv}`: use TAB or COMMA as a delimiter
- MappedReadSet: `{sam}` (default) or `{bam}`
- ReadSet: FastQ format, optionally compressed (depending on the extension).

By default, ReadSets are written a set of one to three FastQ files (2 files for the paired-end reads, and one file for the single-end ones, with empty files omitted). `format\_flags` (since NGLess 0.7) currently supports only `{interleaved}` to output an interleaved FastQ file instead.

Compression is inferred from the `ofile` argument:

- `.gz`: gzip compression
- `.bz2`: bzip2 compression
- `.xz`: xz compression
- `.zstd`: ZStandard compression (since NGLess 1.1)

Comments can be added with the `comment` argument (a free form string), or a list of `auto\_comments`:

- `{date}`: date the script was run,
- `{script}`: script that generated the output,
- `{hash}`: machine readable hash of the computation leading to this output.

6.13.20 print

Print function allows to print a NGLessObject to IO.

Argument:

NGLessObject

Return:

Void

Arguments by value:

none

6.13.21 readlines

Reads a text file and returns a list with all the strings in the file

Argumment

string: the filename

Example

`readlines` is useful in combination with the parallel module, where you can then use the `lock1` function to process a large set of inputs:

```
sample = lock1(readlines('samplelist.txt'))
```

6.13.22 assemble

assemble

Implementation

assemble() uses the MEGAHIT assembler.

Arguments

ReadSet

Returns

string : generated file

Arguments by value:

Name	Type	Required	Default Value
__extra_megahit_arg	List of str	no	[]

`__extra_megahit_arg` is passed directly to megahit with no checking.

6.13.23 orf_find

orf_find finds open reading frames (ORFs) in a sequence set:

```
contigs = assemble(input)
orfs = select(contigs, is_metagenome=True)
```

Argument:

SequenceSet

Return:

SequenceSet

Arguments by value:

Name	Type	Required	Default Value
is_metagenome	Bool	yes	•
include_fragments	Bool	no	True
coords_out	FilePath	no	•
prots_out	FilePath	no	•

- `is_metagenome`: whether input should be treated as a metagenome
- `include_fragments`: whether to include partial genes in the output

Implementation

NGLess uses Prodigal as the underlying gene finder. `is_metagenome=True` maps to anonymous mode.

6.14 Methods

Methods are invoked using an object-oriented syntax. For example:

```
mapped = select(mapped) using |mr|:
    mr = mr.pe_filter()
```

They can also take arguments

```
mapped = select(mapped) using |mr|:
    mr = mr.filter(min_match_size=30)
```

6.14.1 Short reads

Short reads have the following methods:

- `avg_quality()`: the average quality (as a double)
- `fraction_at_least(q)`: the fraction of bases of quality greater or equal to `q`
- `n_to_zero_quality()`: transform the quality scores so that any `N` (or `n`) bases in the sequence get a quality of zero.

6.14.2 Mapped reads

Mapped reads contain several methods. *None of these methods changes its argument, they return new values.* The typical approach is to reassign the result to the same variable as before (see examples above).

- `pe_filter`: only matches where both mates match are kept.
- `flag`: Takes one of `{mapped}` or `{unmapped}` and returns true if the reads were mapped (in a paired-end setting, a read is considered mapped if at least one of the mates mapped).
- `some_match`: Takes a reference name and returns True if the read mapped to that reference name.
- `allbest`: eliminates matches that are not as good as the best. For NGLess, the number of errors (given by the `NM` field) divided by the length of the longest match is the fractional distance of a match. Thus, a match with 3 errors over 100 bp is considered better than a match with 0 errors over 90bps.

filter

`filter` takes a mapped read and returns a mapped read according to several criteria:

- `min_match_size`: minimum match size
- `min_identity_pc`: minimum percent identity (considered over the matching location, trimming on the left and right are excluded).
- `max_trim`: maximum number of bases trimmed off the ends. Use 0 to specify only global matches.

If more than one test is specified, then they are combined with the AND operation (i.e., all conditions have to be fulfilled for the test to be true).

The default is to discard mappings that do not pass the test, but it can be changed with the `action` argument, which must be one of `{drop}` (default: the read is excluded from the output), or `{unmatch}` (the read is changed so that it no longer reports matching).

You can pass the flag `reverse` (i.e., `reverse=True`) to reverse the sign of the test.

6.15 The `load_fastq_directory` function

The `load_fastq_directory` function is one of the main ways to get data *into* NGLess. It takes the name of a directory

```
$ find sample1
sample1/SRR8053346.pair.1.fq.gz
sample1/SRR8053346.pair.2.fq.gz
sample1/SRR8053346.single.fq.gz
sample1/SRR8053355.pair.1.fq.bz2
sample1/SRR8053355.pair.2.fq.bz2
```

This will return a sample that contains *both paired-end **and** single-end data*:

1. The paired-end dataset `sample1/SRR8053346.pair.1.fq.gz` - `sample1/SRR8053346.pair.2.fq.gz`
2. The paired-end dataset `sample1/SRR8053355.pair.1.fq.bz2` - `sample1/SRR8053355.pair.2.fq.bz2`
3. The single-end dataset `sample1/SRR8053346.single.fq.gz`

Currently (as of version `1.4`), `NGLess` supports the following

- Extensions `.gz` and `.bz2` are handled transparently
- The extension (prior to the compression extension) must be either `.fq` or `.fastq`
- Before the extension, one of `.1`/`.2` or `_1`/`_2` or `_F`/`_R` denotes the paired-end matching

If your data does not conform to these rules, we recommend that you use *symlinks* to build a directory that does conform to it.

6.16 Standard library

6.16.1 Parallel module

This module allows you to run several parallel computations.

Important: when you use this module, you will often need to *run the NGLess command multiple times* (one for each sample). These can be run in parallel (and even on different compute nodes on an HPC cluster).

The module provides two functions: `lock1` and `collect`.

`lock1 :: [string] -> string` takes a list of strings and returns a single element. It uses the filesystem to obtain a lock file so that if multiple processes are running at once, each one will return a different element. NGLess also marks results as *finished* once you have run a script to completion.

The intended usage is that you simply run as many processes as inputs that you have and ngless will figure everything out.

For example

```
ngless "1.0"
import "parallel" version "1.0"

samples = ['Sample1', 'Sample2', 'Sample3']
current = lock1(samples)
```

Now, when you run this script, `current` will be assigned to one of `'Sample1'`, `'Sample2'`, or `'Sample3'`. You can use this to find your input data:

```
input = paired("data/" + current + ".1.fq.gz", "data/" + current + ".2.fq.gz")
```

Often, it's a good idea to combine `lock1` with `readlines` (a function which returns the contents of all the non-empty lines in a file as a list of strings):

```
samples = readlines('samples.txt')
current = lock1(samples)
input = paired("data/" + current + ".1.fq.gz", "data/" + current + ".2.fq.gz")
```

You now use `input` as in any other ngless script:

```
mapped = map(input, reference='hg19')
write(input, ofile='outputs/'+current+ '.bam')
counts = count(mapped)
write(counts, ofile='outputs/'+current+ '.txt')
```

This will result in both BAM files and counts being written to the `outputs/` directory. The module also adds the `collect` function which can paste all the counts together into a single table, for convenience:

```
collect(
    counts,
    current=current,
    allneeded=samples,
    ofile='outputs/counts.txt.gz')
```

Now, only when all the samples in the `allneeded` argument have been processed, does ngless collect all the results into a single table.

Full "parallel" example

```
ngless "1.0"
import "parallel" version "1.0"

sample = lock1(readlines('input.txt'))
input = fastq(sample)
mapped = map(input, reference='hg19')
collect(count(mapped, features=['seqname']),
    current=sample,
    allneeded=readlines('input.txt'),
    ofile='output.tsv')
```

Now, you can run multiple `ngless` jobs in parallel and each will work on a different line of `input.txt`.

Parallel internals

Normally this should be invisible to you, but if you are curious or want to debug an issue, here are the gory details:

The function `lock1()` will create a lock file in a sub-directory of `ngless-locks`. This directory will be named by the hash value of the script. Thus, any change to the script will force all data to be recomputed. This can lead to over-computation but it ensures that you will always have the most up to date results (ngless' first priority is correctness, performance is important, but not at the risk of correctness). Similarly, `collect()` will use hashed values which encode both the script and the position within the script (so that if you have more than one `collect()` call, they will not clash).

Lock files have their modification times updated once every 10 minutes while NGLess is running. This allows the programme to easily identify stale files. The software is very conservative, but any lock file with a modification time older than one hour is considered stale and removed. Note that because NGLess will write always create its outputs atomically, the worse that can happen from mis-identifying a stale lock (for example, you had a compute node which lost network connectivity, but it comes back online after an hour and resumes processing) is that extra computation is wasted, **the processes will never interfere in a way that you get erroneous results**.

6.16.2 Samtools module

This module exposes two samtools functionalities: sorting (`samtools_sort`) and selecting reads in regions of interest (`samtools_view`).

```
ngless '1.0'
import "samtools" version "1.0"
input = samfile('input.bam')
sam_regions = samtools_view(input, bed_file="interesting_regions.bed")
write(sam_regions, ofile='interesting.sam')
```

`samtools_view :: mappedreadset -> mappedreadset` returns a subset of the mapped reads that overlap with the regions specified in the BED file.

```
ngless '1.0'
import "samtools" version "1.0"
to_sort = samfile('input.bam')
sorted = samtools_sort(to_sort)
name_sorted = samtools_sort(to_sort, by={name})
write(sorted, ofile='input.sorted.bam')
write(name_sorted, ofile='input.name_sorted.bam')
```

`samtools_sort :: mappedreadset -> mappedreadset` returns a sorted version of the dataset.

Internally, both function call ngless' version of samtools while respecting your settings for the use of threads and temporary disk space. When combined with other functionality, ngless can also often stream data into/from samtools instead of relying on intermediate files (these optimizations should not change the visible behaviour, only make the computation faster).

6.16.3 Mocat module

```
import "mocat" version "1.0"
```

This is a MOCAT compatibility layer to make it easier to adapt projects from MOCAT to ngless.

Functions

`load_mocat_sample :: string -> readset` this is now available as `load_fastq_directory`.

`coord_file_to_gtf :: string -> string` this function takes a MOCAT-style `.coord`, converts it internally to a GTF file and returns it.

Example usage:

```
ngless "1.1"
import "mocat" version "1.1"

sample = load_mocat_sample('Sample1')
mapped = map(sampled, fafile='data/catalog.padded.fna')
write(count(mapped, gff_file=coord_file_to_gtf('data/catalog.padded.coord')),
    ofile='counts.txt')
```

This module can be combined with the parallel module (see above) to obtain a very smooth upgrade from MOCAT to ngless.

6.17 Modules

To add a module to ngless there are two options: *external* or *internal* modules. External modules are the simplest option.

6.17.1 External modules

External modules can perform two tasks:

1. Add new references to ngless
2. Add functions to ngless

Adding references makes them available to the `map()` call using the `reference` argument and (optionally) allows for calls to `count()` without specifying any annotation file.

Like everything else in ngless, these are versioned for reproducibility so that the resulting script implicitly encodes the exact version of the databases used.

Functions in external modules map to command line calls to a script you provide.

6.17.2 How to define an external module

You can use the example module in the ngless source for inspiration. That is a complete, functional module.

A module is defined by an `YaML` file.

Every module has a name and a version:

```
name: 'module'
version: '0.0.0'
```

Everything else is optional.

References

References are added with a *references* section, which is a list of references. A reference contains a `fasta-file` and (optionally) a `gtf-file`. For example:

```
references:
    -
        name: 'ref'
        fasta-file: 'data/reference.fna'
        gtf-file: 'data/reference.gtf.gz'
```

Note that the paths are relative to the module directory. The GTF file may be gzipped.

Initialization

An `init` section defines an initialization command. This will be run **before** anything else in any script which imports this module. The intention is that the module can check for any dependencies and provide the user with an early error message instead of failing later after. For example:

```
init:
    init_cmd: './init.sh'
    init_args:
        - "Hello"
        - "World"
```

will cause ngless to run the command `./init.sh Hello World` whenever a user imports the module.

A note about paths: paths you define in the module.yaml file are *relative to the Yaml file itself*. Thus you put all the necessary scripts and data in the module directory. However, the scripts are run with the current working directory of wherever the user is running the ngless protocol (so that any relative paths that the user specifies work as expected). To find your data files inside your module, ngless sets the environmental variable `NGLESS_MODULE_DIR` as the path to the module directory.

Functions

To add new functions, use a `functions` section, which should contain a list of functions encoded in YaML format. Each function has a few required arguments:

`nglName` the name by which the function will be called **inside** of an ngless script.

`arg0` the script to call for this function. Note that the user will never see this.

For example:

```
functions:
    -
        nglName: "test"
        arg0: "./run-test.sh"
```

will enable the user to call a function `test()` which will translate into a call to the `run-test.sh` script (see the note above about paths).

You can also add arguments to your function, naturally. Remember that ngless functions can have only one unnamed argument and any number of named arguments. To specify the unnamed argument add a `arg1` section, with the key `atype` (argument type):

```
        arg1:
            atype: <one of 'readset'/'mappedreadset'/'counts'/'str'/'flag'/'int'/
↪'option'>
```

The arguments of type *readset*, *mappedreadset*, and *counts* are passed as paths to a file on disk. **Your command is assumed to not change these, but make a copy if necessary. Bad things will happen if you change the files.** You can specify more details on which kind of file you expect with the following optional arguments:

```
            filetype: <one of "tsv"/"fq1"/"fq2"/"fq3"/"sam"/"bam"/"sam_or_bam"/"tsv">
            can_gzip: true/false
            can_bzip2: true/false
            can_stream: true/false
```

The flags `can_gzip`/`can_bzip2` indicate whether your script can accept compressed files (default: *False*). `can_stream` indicates whether the input can be a pipe (default: *False*, which means that an intermediate file will always be used).

For example, if your tool wants a SAM file (and never a BAM file), you can write:

```
        arg1:
            atype: mappedreadset
            filetype: sam
```

`ngless` will ensure that your tool does receive a SAM file (including converting BAM to SAM if necessary).

Finally, additional argument are specified by a list called `additional`. Entries in this list have exactly the same format as the `arg1` entry, except that they have a few extra fields. The extra field `name` is mandatory, while everything else is optional:

```
        additional:
            -
                name: <name>
                atype: <as for arg1: 'readset'/'mappedreadset'/...>
                def: <default value>
                required: true/false
```

Arguments of type `flag` have an optional extra argument, `when-true` which is a list of strings which will be passed as extra arguments when the flag is true. You can also just specify a single string. If `when-true` is missing, ngless will pass an option of the form `--name` (i.e., a double-dash then the name used). For example:

```
        additional:
            -
                name: verbose
                atype: flag
                def: false
                when-true: "-v"
            -
                name: complete
                atype: flag
                def: false
                when-true:
                    - "--output=complete"
                    - "--no-filter"
```

All other argument types are passed to your script using the syntax `--name=value` if they are present or if a default has been provided.

Arguments of type `option` map to symbols in ngless and require you to add an additional field `allowed` specifying

the universe of allowed symbols. Ngless will check that the user specifies arguments from the allowable universe. For example:

```
        additional:
            -
                atype: 'option'
                name: 'verbosity'
                def: 'quiet'
                allowed:
                    - 'quiet'
                    - 'normal'
                    - 'loud'
```

If you do not have a fixed universe for your argument, then it should be a `str` argument.

The `required` flag determines whether the argument is required. Note that arguments with a default argument are automatically optional (ngless may trigger a warning if you mark an argument with a default as required).

To return a value, you must request that ngless generate a new temporary file for the script to generate output to. Therefore, you need to specify a `return` section, with three parameters: `rtype` (return type, see below), `name` the name of the argument to use, and `extension` the file extension of the output type.

```
    return:
        rtype: "counts"
        name: "ofile"
        extension: "sam"
```

`rtype` must be one of `"void"`, `"counts"` or `"mappedreadset"`. Returning `readset` isn't currently supported.

If you plan to make use of search path expansion, in order for NGLess to expand the argument prior to passing it to the external module you need to set `atype: "str"` and `expand_searchpath: true`.

```
    additional:
        -
            atype: 'str'
            name: 'reference'
            expand_searchpath: true
```

Citation

Finally, if you wish to, you can add one or more citations:

```
citation: "A paper which you want to be listed when users import your module"
```

This will be printed out whenever users use your module and thus will help you get exposure.

If you have more than one citation, you can use the `citations` key and provide a list:

```
citations:
    - "Paper 1"
    - "Paper 2"
```

6.17.3 Minimal NGLess Version

External modules can specify a minimal NGLess version that they need to run. This is optional, but if it is used, you need to additionally supply a reason for the requirement (using the aptly-named `reason` field):

```
min-ngless-version:
    min-version: "1.3"
    reason: "The min-ngless-version field is only supported since NGLess 1.3"
```

6.17.4 Internal Modules

This is very advanced as it requires writing Haskell code which can then interact very deeply with the rest of ngless.

For an example, you can look at the example internal module. If you want to get started, you can ask about details on the ngless user mailing list.

6.18 Counting in NGLess

The `count()` function takes a `MappedReadSet` (the logical equivalent of a SAM/BAM file) and summarizes the information therein. The `count()` function can perform three types of operation, depending on the `features` argument:

1. `seqname`: only the `MappedReadSet` is necessary.
2. Using a GFF/GTF file.
3. Using a `functional_map` (TSV) file

Option #1 is the simplest to understand: just summarize based on the names of the sequences. This is appropriate for obtaining per-gene abundances from gene catalogs. Options #2 and #3 are similar: for each `MappedRead`, `count()` will use the passed reference to map to a set of features and summarize those. Using the GFF format is much more flexible and allows for a lot more filtering, but also *significantly costlier in time and memory*. At a high-level, the process is similar:

For example, if you have an insert (either a paired-end or single-end short-read) that mapped to a gene called `G1` and that gene is annotated with two gene ontology terms, both will be considered and their counts adjusted. If the insert mapped to multiple genes, then all the terms will be considered, but as a set: if an insert is mappped to `G1`, which has two GO annotations and also to `G2` which has the same GO annotations (which is very frequent), then those annotations will be counted a single time.

How counts are adjusted in the presence of multiple annotations is defined by the `multiple` argument. Generally, for obtaining gene abundances, distribution of multiple mappers is the best (using `multiple={dist1}`), while for *functional annotations*, you want to count them all (using `multiple={all1}`). This implies that the functional annotations will sum to a higher value than the number of reads. This may seem strange at first, but it is the intended behaviour.

See also the full description of all count arguments in the API docs.

6.18.1 A TSV (tab-separated values) file for use in the `functional_map` argument

The file consists of a header and content.

TSV header

The simplest header is just a single line of tab separated column headers. That line *may* start with a `#` sign, which is ignored. Alternatively, a multi-line header consists of multiple lines starting with `#`. The last one of these will be parsed as the header.

Examples of TSV headers

ALl the

1. Simple, 1 line header, with a `#` sign

```
#geneID        feat1        feat2        feat3
G1        ann        ann        ann
G2        ann        ann        ann
```

1. Simple, 1 line header, without a `#` sign

```
geneID        feat1        feat2        feat3
G1        ann        ann        ann
G2        ann        ann        ann
```

1. Multi-line header, `#` signs are required

```
# My comment can span multiple lines
# The last one of these is the header!
#geneID        feat1        feat2        feat3
G1        ann        ann        ann
G2        ann        ann        ann
```

Note: format #3 is only supported in NGLess version 1.1 and above

TSV content

Values can be

1. empty.
2. lists of entries, separated by either `,` or `|` characters.

```
#geneID        feat1        feat2        feat3
G1        a1,a2        b        c
G2        a1|a3                c
```

In this case, the mappings are:

- `G1` has the properties `a1` and `a2` in the `feat1` feature; `b` in the `feat2` feature; and `c` in the `col3`
- `G2` has the properties `a1` and `a3` in the `feat1` feature; and `c` in the `col3`. There is no `feat2` associated with `G2` and, from `feat2`'s point-of-view, inserts mapped to `G2` are considered unmapped

As of NGLess 1.1, *spaces are not allowed*: i.e., `a,  b` is the feature `a` and the feature `b` (space followed by b). This is arguably sub-optimal.

6.19 NGLess Constants

In NGLess, any variable written in uppercase is a constant, i.e., can only be assigned to once. In addition, there are builtin constants defined by NGLess.

6.19.1 Built in constants

- ARGV

This is string array which contains the arguments passed to the script

- STDIN

Use in place of a filename to read from standard input

- STDOUT

Use in place of a filename to write to standard output

For example:

```
ngless '0.9'

input = samfile(STDIN)
input = select(input) using |mr|:
    if mr.flag({mapped}):
        discard
write(input, ofile=STDOUT, format={bam})
```

This file reads a sam stream from stdin, filters it (using the `select` call) and writes to standard output in `bam` format.

6.20 Available Reference Genomes

NGLess provides builtin support for the most widely used model organisms (human, mouse, yeast, C. elegans, . . . ; see the full table below). This makes it easier to use the tool when using these organisms as some knowledge is already built in.

6.20.1 Genome references available

NGLess provides archives containing data sets of organisms. Is also provided gene annotations that provide information about protein-coding and non-coding genes, splice variants, cDNA and protein sequences, non-coding RNAs.

The following table represents organisms provided by default:

Name	Description	Assembly	Ensembl
bosTau4	bos_taurus	UMD3.1	75
ce10	caenorhabditis_elegans	WBcel235	75
canFam3	canis_familiaris	CanFam3.1	75
dm6	drosophila_melanogaster	BDGP6	90
dm5	drosophila_melanogaster	BDGP5	75
gg5	gallus_gallus	Gallus_gallus-5.0	90
gg4	gallus_gallus	GalGal4	75
hg38.p10	homo_sapiens	GRCh38.p10	90
hg38.p7	homo_sapiens	GRCh38.p7	85
hg19	homo_sapiens	GRCh37	75
mm10.p5	mus_musculus	GRCm38.p5	90
mm10.p2	mus_musculus	GRCm38.p2	75
rn6	rattus_norvegicus	Rnor_6.0	90
rn5	rattus_norvegicus	Rnor_5.0	75
sacCer3	saccharomyces_cerevisiae	R64-1-1	75
susScr11	sus_scrofa	Sscrofa11.1	90

These archives are all created using versions 75, 85 and 90 of Ensembl.

6.20.2 Automatic installation

The builtin datasets are downloaded the first time they are used. They are downloaded to the user home directory and stored in **home/**.ngless/genomes.

6.20.3 Manual installation

Is possible to install data sets locally, before running any script. They can be installed in **User** mode or in **Root** mode.

To install locally (organism bos taurus), use the following command:

```
$ ngless --install-reference-data bosTau4
```

If you install as a super-user, then the dataset will be available for all users:

```
$ sudo ngless --install-reference-data bosTau4
```

When attempting to install an organism if is returned **True** it means that the organism is already installed, and there is no reason to install again. Otherwise, a progress bar is displayed to provide information on the download.

6.21 Configuration

Note: NGLess' results do not change because of configuration or command line options. **The NGLess script always has complete information on what is computed**. What configuration options change are details of *how* the results are computed such as where to store intermediate files and how many CPU cores to use.

Ngless gets its configuration options from the following sources:

1. Defaults/auto-configuration

2. A global configuration file
3. A user configuration file (typically `$HOME/.config/ngless.conf`)
4. A configuration file present in the current directory‘
5. A configuration file specified on the command line
6. Command line options

In case an option is specified more than once, the order above determines priority: later options take precedence.

6.21.1 Configuration file format

NGLess configuration files are text files using assignment syntax. Here is a simple example, setting the temporary directory and enabling auto-detection of the number of threads:

```
temporary-directory = "/local/ngless-temp/"
jobs = "auto"
```

6.21.2 Options

`jobs`: number of CPUs to use. You can use the keyword `auto` to attempt auto-detection (see below).

`strict-threads`: by default, NGLess will, in certain conditions, use more CPUs than specified by the `jobs` argument (in bursts of activity). This happens, for example, when it calls an external short-read-mapper (such as bwa). By default, it will pass the threads argument through to `bwa`. However, it will still be processing `bwa`'s output using its own threads. This will results in small bursts of activity where the CPU usage is above `jobs`. If you specify `--strict-threads`, however, then this behavior is curtailed and it will never use more threads than specified (in particular, it will call `bwa` using one thread fewer than specified, while restricting itself to a single thread, thus even peak usage is at most the number of specified threads).

`temporary-directory`: where to keep temporary files. By default, this is the system defined temporary directory (either `/tmp` or the value of the `$TEMPDIR` environment variable on Unix).

`color`: whether to use color output. Defaults to `auto` (i.e., print color if the output is a terminal), `no` (never use color), `force` (use color even if writing to a file or pipe), `yes` (synonym of `force`).

`print-header`: whether to print ngless banner (version info...).

`user-directory`: user writable directory to cache downloads (default is system dependent, on Linux, typically it is `$HOME/.local/share/ngless/`).

`user-data-directory`: user writable directory to cache data (default is a `data` directory inside the `user-directory` [see above]).

`index-path`: user writable directory to store indices and similar data.

`global-data-directory`: global data directory.

Debug options

`keep-temporary-files`: whether to keep temporary files after the end of the programme.

`trace` (only command line): print a lot of internal information.

Auto-detection of the number of CPUs

If the option `auto` is passed as the number of jobs (either on the command line or in the configuration file), ngless will inspect the environment looking for a small set of clues as to how many CPUs to use. In particular, it will make use of these variables:

- `OMP_NUM_THREADS`
- `NSLOTS`
- `LSB_DJOB_NUMPROC`
- `SLURM_CPUS_PER_TASK`

If none are found (or they do not contain a single number), an error is produced.

6.22 Search path expansion

Note: Search path expansion is a very powerful feature. It can be abused to defeat NGLess' reproducibility mechanisms and to obsfucate which reference information is being used. However, if used correctly, it can greatly simplify file management and **enhance** reproducibility.

NGLess supports a search path system to find references. Certain functions (such as map()) support *search path expansion*. For example, you can write:

```
map(input, fafile="<>/my-reference.fa")
```

Then if the search path consists of `"/opt/ngless-references/"`, the expanded version will be `"/opt/ngless-references/my-reference.fa"`.

Named and unnamed search paths

You can have named and unnamed paths in your search path. The rules are a bit complex (see below), but it makes sense if you see examples:

```
map(input, fafile="<references>/my-reference.fa")
```

With the search path `['references=/opt/ngless-refs']` will result in `'/opt/ngless-refs/my-reference.fa'`.

With the search path `['internal=/opt/ngless-internal', 'references=/opt/ngless-refs']` will also result in `'/opt/ngless-refs/my-reference.fa'` as the *internal* path will not be matched.

With the search path `['internal=/opt/ngless-internal', 'references=/opt/ngless-refs', '/opt/ngless-all']` now it will result in `['/opt/ngless-refs/my-reference.fa', '/opt/ngless-all/my-reference.fa']` as the unnamed path will always match. Since there is more than one result, both are checked (in order).

Using <> (as in the example above) will use only unnamed paths.

Setting the search path

The search path can be passed on the command line:

```
ngless script.ngl --search-path "references=/opt/ngless"
```

Alternatively, you can set it on the ngless configuration file:

```
search-path = ["references=/opt/ngless"]
```

Note that **the search path is a list**, even if it contains a single element.

Rules

1. If a path matches <([^>]*)>, then it is path expanded.
2. The search path (which is a list of named and unnamed search paths) if filter. A path is kept on the list if it is an unnamed paht or if the name matches the requested pattern (<references> requests "references"; <> never matches so that only unnamed paths are kept).
3. Paths are tested in order and the first path referring to an existing file is kept.

Similarly

6.23 Reproducible Computation With NGLess

NGLess has several builtin features to make it easier to achieve reproducible research.

All information that is needed to run a result is contained in the NGLess script. There is no command line or configuration option which changes the results: they only change the way in which the computation was run (what information was printed on the console, where intermediate files were saved, &c).

The version annotations that NGLess requires also enhance reproducibility while allowing us to update NGLess going forward.

6.23.1 Annotate results with input script

The `write()` function call supports the argument `auto_comments` which will add (as comments) meta information to the output. In particular, you can use the `{script}` auto comment to add the script to your output. For example:

```
ngless '1.2'
mapped = samfile('input.bam')

counted = count(mapped, features=['seqname'])
write(counted,
        ofile='output.txt',
        auto_comments=[{script}]) # <<<< ADD SCRIPT
```

This will add the script to your output. Thus, it will be easy to see how the output was generated.

You can also use `{date}`, which will output a string with the date in which the script was run (note that the result is no longer reproducible at the Byte level as each run will contain a different date/time). Finally, the `comment` argument allows for any free text string:

```
write(counted,
        ofile="output.txt",
        comment="For my awesome Science publication",
        auto_comments=[{script}])
```

Finally, you can use the magical `{hash}` auto comment:

```
write(counted,
        ofile="output.txt",
        comment="For my awesome Science publication",
        auto_comments=[{script}, {hash}])
```

This will add a hash string to the output describing the computational path to compute the result. This is smarter than a simple hash of the script as it does not consider code that is not necessary to generate the script or elements such as variable names (i.e., if you change the variable names, the hash will stay the same as it is the same computational path).

The `collect()` function also support the same arguments.

6.24 Frequently Asked Questions

This is a list of questions we have regularly gotten on the project. See below for questions about the ngless language.

6.24.1 Why a new domain-specific language instead of a library in Python (or another existing language)?

First of all, you can actually use NGLess through Python, using NGLessPy.

However, the native mode of NGLess is using its internal DSL (domain specific langugage). There are several advantages to this approach:

1. Fast error checking which can speed up the development process. For example, static type checking, which is known to many programmer. In general, we do a lot of error checking before even starting interpretation. We perform syntax and error checking, but we can also check some conditions that can be tricky to express with simple types only (e.g., certain parameter combinations can be illegal). We also pre-check all the input files (so even if you only use a particular file in step 5 of your process, we check if it exists even before running steps 1 through 4). We even do some things like: if you use step 1 to compute to name of the input file that will be used in step 5, we will check it immediately after step 1. Same for output files. If you issue a `write()` call using `output/results.txt` as your output filename, we will check if a directory named output exists and is writable. We also try to be helpful in the error messages (mispelled a parameter value? Here's an error, but also my best guess of what you meant + all legal values). I really care about error messages.
2. By controlling the environment more than would be typical with a Python library (or any other language), we can also get some reproducibility guarantees. Note too that we declare the version of every script so that we can update the interpreter in the future without silently changing the behaviour of older ones.
3. Using a domain specific language makes the resulting scripts very readable even for non-experts as there is little boilerplate.
4. Finally, we needed the result to be fast and languages such as Python often add a lot of overhead.

6.24.2 Is the language extensible?

Yes.

While the basic types and syntax of the language are fixed, it is not hard to add external modules that introduce new functions. These can be described with a YAML file and can use any command line tool.

Add new model organisms can similarly be done with simple YAML file.

More advanced extensions can be done in Haskell, but this is considered a solution for advanced users.

6.24.3 Couldn't you just use Docker/Bioboxes?

Short answer: Bioboxes gets us part of the way there, but not all of it; however, if we think of these technologies as complements, we might get more out of them.

Longer answer:

Several of the goals of ngless can be fulfilled with a technology such as bioboxes. Namely, we can obtain reproducibility of computation, including across platforms using bioboxes without having to bother with ngless. However, the result is less readable than an ngless script, which is another important goal of ngless. An ngless script can be easily be submitted as supplemented methods to a journal publication and even be easily scrutinized by a knowledgeable reviewer in an easier way than a Docker container.

Furthermore, the fact that we work with a smaller domain than a Docker-based solution (we only care about NGS) means that we can provide the users a better experience than is possible with a generic tool. In particular, when the user makes a mistake (and all users will make mistakes), we can diagnose it faster and provide a better error message than is possible to do with Bioboxes.

6.24.4 What is the relationship of ngless to the Common Workflow Language?

Like for the question above, we consider ngless to be related to but not overlapping with the CWL (Common Workflow Language).

In particular, much of functionality of ngless can also be accessed in CWL workflow, using our command line wrappers all of which have CWL wrappers.

Additionally, (with some limitations), you can embedded a generic NGLess script within a larger CWL workflow by using the `--export-cwl` functionality. For example, to automatically generate a wrapper for a script called `my-script.ngl`, call:

```
ngless --export-cwl=wrapper.cwl my-script.ngl
```

The automatically generated `wrapper.cwl` file can now be used as a CWL tool within a larger pipeline. See more in the CWL page.

6.24.5 How does ngless interact with job schedulers and HPC clusters?

Generally speaking, it does not. It can be used with HPC clusters, whereby you simply run a job that calls the ngless binary.

The parallel module can be used to split large jobs in many tasks, so that you can run multiple ngless instances and they collaborate. It is written such that does not depend on the HPC scheduler and can, thus, be used in any HPC system (or even, for smaller jobs, on a single machine).

6.24.6 Questions about the ngless language

6.24.7 Can I pass command line arguments to a script?

Yes, you can. Just add them as additional arguments and they will be available inside your script as `ARGV`.

6.24.8 What are symbols (in the ngless language)?

If you are familiar with the concept, you can think of them as `enums` in other languages.

Whenever a symbol is used in the argument to a function, this means that that function takes only one of a small number of possible symbols for that argument. This improves error checking and readibility.

6.24.9 Does the `select` function work on inserts (considering both mates) or per-read (treating the data as single-ended)?

By default, `select` considers the insert as a whole, but you can have it consider each read as single-end by using setting the `paired` argument to `False`.

6.24.10 Should I use a TSV file or a GFF for the `count` function?

Short answer: If you have a choice, use TSV; if you must, use GFF.

Long answer: The TSV format is much more limited, annotating each reference sequence with a set of annotation terms. This is appropriate for gene catalogues. With the GFF format, you can annotate areas of the reference with different annotations. This is appropriate for (1) mapping metagenomes against reference genomes (where, due to strain variability, different areas of the genome may be differentially present in your samples) and (2) mapping (meta)transcriptomes against reference genomes.

The GFF format is, thus, more powerful than the TSV format (this is meant in the strict sense: everything that you can do with the TSV file can also be done with a GFF by setting the coordinates to cover the whole sequence). However, this implies a significantly higher computational cost (both in terms of time and memory usage), which is why you should not use the functionality unless you need it.

6.25 NGLessPy: NGLess in Python

Note As of Oct 2017, NGLess is considered beta software (we believe it works, but there may still be a few rough edges), while NGLessPy is alpha software (very experimental).

6.25.1 Install

```
pip install NGLessPy
```

(or from source, using the standard `python setup.py install`)

6.25.2 Basic Tutorial

This tutorial expects a certain familiarity with general ngless concepts and functions.

We start by importing the `NGLess` object:

```
from ngless import NGLess
```

We now build an `NGLess.NGLess` object, giving it the version of ngless we wish (this is like the version declaration at the top of an NGLess file:

```
sc = NGLess.NGLess('0.8')
```

To simplify the rest of the script, we are going to use the short name `e` to refer to the environment of the script we are generating. The environment is what will hold the ngless variables we will use:

```
e = sc.env
```

We can import ngless modules using the `import_` function (using `name` and `version`):

```
sc.import_('mocat', '0.0')
```

Now, we can use all NGLesss functionality. Functions get an underscore at the end, like this:

```
e.sample = sc.load_mocat_sample_('testing')
```

`preprocess_` is special because it takes a block in ngless, which maps to it taking a function in Python. We can use decorator syntax to do it all compactly:

```
@sc.preprocess_(e.sample, using='r')
def proc(bk):
    # bk is the block environment, where `r` is defined
    bk.r = sc.substrim_(bk.r, min_quality=25)
```

Now, we map against `hg19` and filter it. As you can see, ngless functions are called with an extra underscore and variables are kept in the `e` object:

```
e.mapped = sc.map_(e.sample, reference='hg19')
e.mapped = sc.select_(e.mapped, keep_if=['{mapped}'])

sc.write_(e.mapped, ofile='ofile.sam')
```

Finally, we execute the resulting script:

```
sc.run(auto_install=True)
```

This will even install NGLess if it is not available in the PATH.

6.25.3 Full script

```
from ngless import NGLess

sc = NGLess.NGLess('0.8')
e = sc.env

sc.import_('mocat', '0.0')

e.sample = sc.load_mocat_sample_('testing')
@sc.preprocess_(e.sample, using='r')
def proc(bk):
    # bk is the block environment, where `r` is defined
    bk.r = sc.substrim_(bk.r, min_quality=25)

e.mapped = sc.map_(e.sample, reference='hg19')
e.mapped = sc.select_(e.mapped, keep_if=['{mapped}'])
```

(continues on next page)

(continued from previous page)

```
    sc.write_(e.mapped, ofile='ofile.sam')

    sc.run(auto_install=True)
```

6.26 Common Workflow Language Integrations

6.26.1 Simple operations

Simple NGLess operations can be performed through the command line wrappers, all of which have a CWL tool description.

6.26.2 Automatic CWL export of NGLess scripts

An NGLess script that conforms to certain rules can be exported as a CWL tool using the `--export-cwl` option:

```
ngless script.ngl --export-cwl=tool.cwl
```

The rules are simple: the script must use `ARGV` for its inputs and outputs. For example, this is a conforming script:

```
ngless "0.8"

mapped = samfile(ARGV[1])

mapped = select(mapped, drop_if=[{mapped}])

write(mapped,
        ofile=ARGV[2])
```

The resulting tool will take two arguments, specifying its input and output.

6.27 Advanced options

6.27.1 Subsample mode

Subsample mode simply *throws away >90% of the data*. This allows you to quickly check whether your pipeline works as expected and the output files have the expected format. Subsample mode should never be used in production. To use it, pass the option `--subsample` on the command line:

```
ngless --subsample script.ngl
```

will run `script.ngl` in subsample mode, which will probably run much faster than the full pipeline, allowing to quickly spot any issues with your code. A 10 hour pipeline will finish in a few minutes (sometimes in just seconds) when run in subsample mode.

Note: subsample mode is also a way to make sure that all indices exist. Any `map()` calls will check that the necessary indices are present: if a `fafile` argument is used, then the index will be built if necessary; if a `reference` argument is used, then the necessary datasets are downloaded if they have not previously been obtained.

Subsample mode also changes all your `write()` so that the output files include the `subsample` extension. That is, a call such as:

```
write(output, ofile='results.txt')
```

will automatically get rewritten to:

```
write(output, ofile='results.txt.subsample')
```

This ensures that you do not confuse subsampled results with the real thing.

6.28 NGLess Language

This document describes the NGLess language.

6.28.1 Tokenization

Tokenization follows the standard C-family rules. A word is anything that matches `[A-Za-z_][A-Za-z_0-9]*`. The language is case-sensitive. All files are assumed to be in UTF-8.

Both LF and CRLF are accepted as line endings (Unix-style LF is preferred).

A semicolon (;) can be used as an alternative to a new line. Any spaces (and only space characters) following a semicolon are ignored. *This feature is intended for inline scripts at the command line (passed with the `-e` option), its use for scripts is heavily discouraged and may trigger an error in the future.*

Script-style (# to EOL), C-style (/* to */) and C++-style (// to EOL) comments are all recognised. Comments are effectively removed prior to any further parsing as are empty lines.

Strings are denoted with single or double quotes and standard backslashed escapes apply (\n for newline, ...).

A symbol is denoted as a token surrounded by curly braces (e.g., `{symbol}` or `{gene}`).

Integers are specified as decimals `[0-9]+` or as hexadecimals `0x[0-9a-fA-F]+`.

6.28.2 Version declaration

The first line (ignoring comments and empty lines) of an NGLess file MUST be a version declaration:

```
ngless "0.9"
```

6.28.3 Module Import Statements

Following the version statement, optional import statements are allowed, using the syntax `import "<MODULE>" version "<VERSION>"`. For example:

```
import "batch" version "1.0"
```

This statement indicates that the `batch` module, version `1.0` should be used in this script. Module versions are independent of NGLess versions.

Only a predefined set of modules can be imported (these are shipped with NGLess). To import user-written modules, the user MUST use the *local import* statement, e.g.:

```
local import "batch" version "1.0"
```

Import statements MUST immediately follow the version declaration

Blocks

Blocks are defined by indentation in multiples of 4 spaces. To avoid confusion, TAB characters are not allowed.

Blocks are used for conditionals and `using` statements.

6.28.4 Data types

NGless supports the following basic types:

- String
- Integer
- Double
- Bool
- Symbol
- Filename
- Shortread
- Shortreadset
- Mappedread
- Mappedreadset

In addition, it supports the composite type List of X where X is a basic type. Lists are built with square brackets (e.g., [1,2,3]). All elements of a list must have the same data type.

String

A string can start with either a quote `(U+0022, ")` or a single quote `(U+0027,')` or and end with the same character. They can contain any number of characters.

Special sequences start with `\`. Standard backslashed escapes can be used as `LF` and `CR` (`\n` and `\r` respectively), quotation marks (`\'`) or slash (`\\`).

Integer

Integers are specified as decimals `[0-9]+` or as hexadecimals `0x[0-9a-fA-F]+`. The prefix `-` denotes a negative number.

Double

Doubles are specified as decimals `[0-9]+` with the decimal point serving as a separator. The prefix `-` denotes a negative number.

Doubles and Integers are considered numeric types.

Boolean

The two boolean constants are `True` and `False` (which can also be written `true` or `false`).

Symbol

A symbol is denoted as a token surrounded by curly braces (e.g., `{symbol}` or `{drop}`). Symbols are used as function arguments to indicate that there is only a limited set of allowed values for that argument. Additionally, unlike Strings, no operations can be performed with Symbols.

6.28.5 Variables

NGless is a statically typed language and variables are typed. Types are automatically inferred from context.

Assignment is performed with = operator:

```
variable = value
```

A variable that is all uppercase is a constant and can only be assigned to once.

6.28.6 Operators

Unary

The operator `(-)` returns the symmetric of its numeric argument.

The operator `len` returns the length of a ShortRead.

The operator `not` negates its boolean argument

6.28.7 Binary

All operators can only be applied to numeric types. Mixing integers and doubles returns a double. The following binary operators are used for arithmetic:

```
+ - < > >= <= == !=
```

The + operator can also perform concatenation of String objects.

The </> operator is used to concatenate two Strings while also adding a '/' character between them. This is useful for concatenating file paths.

6.28.8 Indexing

Can be used to access only one element or a range of elements in a ShortRead. To access one element, is required an identifier followed by an expression between brackets. (e.g, x[10]).

To obtain a range, is required an identifier and two expressions separated by a ':' and between brackets. Example:

- `x[:]` - from position 0 until length of variable x
- `x[10:]` - from position 10 until length of variable x
- `x[:10]` - from position 0 until 10

6.28.9 Conditionals

Conditionals work as in Python. For example:

```
if 5 > 10:
   val = 10
else:
   val = 20
```

6.28.10 Functions

Functions are called with parentheses:

```
result = f(arg, arg1=2)
```

Functions have a single positional parameter, all other must be given by name:

```
unique(reads, max_copies=2)
```

The exception is constructs which take a block: they take a single positional parameter and a block. The block is passed using the `using` keyword:

```
reads = preprocess(reads) using |read|:
    block
    ...
```

The `|read|` syntax defines an unnamed (lambda) function, which takes a variable called `read`. The function body is the following block.

There is no possibility of defining new functions within the language. Only built-in functions or those added by modules can be used.

6.28.11 Methods

Methods are called using the syntax `object . methodName ( <ARGS> )`. As with functions, one argument may be unnamed, all others must be passed by name.

6.28.12 Grammar

This is the extended Backus-Naur form grammar for the NGLess language (using the ISO 14977 conventions). Briefly, the comma (`,`) is used for concatenation, `[x]` denotes *optional*, and `{x}` denotes *zero or more of* `x`.

```
string = ? a quoted string, produced by the tokenizer ? ;
word = ? a word produced by the tokenizer ? ;

eol =
    ';'
    | '\n' {'\n'}
    ;


ngless = [header], body;
```

(continues on next page)

(continued from previous page)

```
header = {eol}, ngless_version, {eol}, {import}, {eol}

ngless_version = "ngless", string, eol ;

import = ["local"],  "import", string, "version", string, eol ;

body = {expression, eol} ;

expression =
            conditional
            | "discard"
            | "continue"
            | assignment
            | innerexpression
            ;

innerexpression = left_expression, binop, innerexpression
                    | left_expression
                    ;

left_expression =  uoperator
                    | method_call
                    | indexexpr
                    | base_expression
                    ;

base_expression = pexpression
                   | funccall
                   | listexpr
                   | constant
                   | variable
                   ;

pexpression = '(', innerexpression, ')' ;

constant =
        "true"
        | "True"
        | "false"
        | "False"
        | double
        | integer
        | symbol
        ;

double = integer, '.', integer ;
integer = digit, {digit} ;
digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' ;
symbol = '{', word, '}' ;


indentation = ' ', {' '} ;
binop = '+' | '-' | '*' | "!=" | "==" | "</>" | "<=" | "<" | ">=" | ">" | "+" | "-" ;

uoperator =
        lenop
        | unary_minus
```

(continues on next page)

(continued from previous page)

```
        | not_expr
        ;

lenop = "len", '(', expression, ')'
unary_minus = '-', base_expression ;
not_expr = "not", innerexpression ;

funccall = paired
        | word,  '(', innerexpression, kwargs, ')', [ funcblock ]
        ;

(* paired is a special-case function with two arguments *)
paired = "paired", '(', innerexpression, ',', innerexpression,  kwargs ;

funcblock = "using", '|', [ variablelist ], '|', ':', block ;


kwargs = {',', variable, '=', innerexpression} ;

assignment = variable, '=', expression ;

method_call = base_expression, '.', word, '(', [ method_args ], ')';
method_args =
        innerexpression, kwargs
        | variable, '=', innerexpression, kwargs
        ; (* note that kwargs is defined as starting with a comma *)

indexexpr = base_expression, '[', [ indexing ], ']' ;

indexing = [ innerexpression  ], ':', [ innerexpression ] ;

listexpr = '[', [ list_contents ] , ']' ;
list_contents = innerexpression, {',', innerexpression } ;

conditional = "if",  innerexpression, ':',  block, [ elseblock ] ;
elseblock = "else", ':', block ;
block = eol, indentation, expression, eol, {indentation, expression, eol} ;

variablelist = variable, {',', variable} ;
variable = word ;
```

6.29 Mapping

Mapping is one of the major functions of NGLess. Here we describe, in more detail, some of its functionality.

Mapping is implemented using bwa. As of version 1.4, NGLess uses *bwa 0.7.17* by default.

By default, bwa is called with default parameters. If the `mode_all` argument is set to true, then `-a` is passed to `bwa`.

6.29.1 Low memory mode

As databases get very large, memory requirements can grow very large. In order to make large databases accessible to users without access to large memory machines, NGLess implements a simple heuristic: it splits the input database into smaller blocks, processes each one in turn and combines the results at the end.

To enable low-memory mode, use the `block_size_megabases` in the script. Set it to a value that is less than the available memory. Note that this *does change* the results (although the impact is limited).

A FAQ is why the memory requirements are not a configuration option and must be specified in the script. As low memory mode is heuristic, it can potentially *change* results. As NGLess aims to capture all parameters that can change the result **inside** the script, it must be specified as an argument to `map()`.

6.29.2 Using SOAPAligner

Note: Support for SOAPAligner is experimental (as of version 0.6)

You can use SOAPAligner as an alternative to bwa using the following code:

```
import "soap" version "0.0"

input = ....

mapped = map(input, mapper="soap")
```

Note that, unlike the case for bwa, SOAPAligner is not bundled with NGLess and must be in the PATH to be used.

6.30 Contacts

How to get in touch

6.30.1 I have some questions comments

The best is probably the user mailing list. Both developers and other users are there.

6.30.2 I want to report a bug

The best is to report it using Github issues. This way, it will not get lost even if everyone happens to be busy at the moment.

If you are not sure if it's a bug, feel free to ask on the user mailing list first.

6.30.3 I want to discuss the internals of development

There are two major channels for that:

- Developer mailing list
- Gitter channel

6.31 Software used by NGLess

NGLess internally uses a few other packages to implement specific functionality. As we believe in giving appropriate credit, these pacakges are printed in the citation list of any script that uses them.

NGLess version 1.4 uses the following software tools:

- Samtools (used for SAM/BAM handling as well as in the samtools module: version 2.13
- BWA (used for map): version 0.7.17
- Minimap (used for map as an alternative to bwa): version 2.24
- Prodigal (used for orf_find): version 2.6.3
- Megahit (used for assemble): version 1.2.9