Sunbeam Documentation

Release 1.0

Erik Clarke, Chunyu Zhao, Jesse Connell

Jun 27, 2022

Contents:

Sunbeam is a pipeline written in snakemake that simplifies and automates many of the steps in metagenomic sequencing analysis. It uses conda to manage dependencies, so it doesn't have pre-existing dependencies or admin privileges, and can be deployed on most Linux and Mac workstations and clusters.

Sunbeam currently automates the following tasks:

- Quality control, including adaptor trimming, host read removal, and quality filtering;
- Taxonomic assignment of reads to databases using Kraken;
- Assembly of reads into contigs using Megahit;
- Contig annotation using BLAST[n/p/x];
- Mapping of reads to target genomes; and
- ORF prediction using Prodigal

Sunbeam was designed to be modular and extensible. We have a few pre-built *Sunbeam Extensions* available that handle visualization tasks, including contig assembly graphs, read alignments, and taxonomic classifications.

To get started, see our *Quickstart Guide*!

CHAPTER 1

Quickstart Guide

Contents

1.1 Installation

Download a copy of Sunbeam from our GitHub repository, and install.

```
git clone -b stable https://github.com/eclarke/sunbeam sunbeam-stable
cd sunbeam-stable
./install.sh
tests/run_tests.bash -e sunbeam
```

This installs Sunbeam and all its dependencies, including the Conda environment manager, if required. It then runs some tests to make sure everything was installed correctly.

Tip: If you've never installed Conda before, you'll need to add it to your shell's path. If you're running Bash (the most common terminal shell), the following command will add it to your path: `echo 'export PATH=$PATH:$HOME/miniconda3/bin` > ~/.bashrc`

If you see "Tests failed", check out our *Troubleshooting* section or file an issue on our GitHub page.

1.2 Setup

Let's say your sequencing reads live in a folder called `/sequencing/project/reads`, with one or two files per sample (for single- and paired-end sequencing, respectively). These files *must* be in gzipped FASTQ format.

Let's create a new Sunbeam project (we'll call it `my_project`):

```
source activate sunbeam
sunbeam init my_project --data_fp /sequencing/project/reads
```

Sunbeam will create a new folder called `my_project` and put two files there:

- `sunbeam_config.yml` contains all the configuration parameters for each step of the Sunbeam pipeline.
- `samples.csv` is a comma-separated list of samples that Sunbeam found the given data folder, along with absolute paths to their FASTQ files.

Right now we have everything we need to do basic quality-control and contig assembly. However, let's go ahead and set up contaminant filtering and some basic taxonomy databases to make things interesting.

1.2.1 Contaminant filtering

Sunbeam can align your reads to an arbitrary number of contaminant sequences or host genomes and remove reads that map above a given threshold.

To use this, make a folder containing all the target sequences in FASTA format. The filenames should end in "fasta" to be recognized by Sunbeam. In your `sunbeam_config.yml` file, edit the `host_fp:` line in the `qc` section to point to this folder.

1.2.2 Taxonomic classification

Sunbeam can use Kraken to assign putative taxonomic identities to your reads. While creating a Kraken database is beyond the scope of this guide, pre-built ones are available at the Kraken homepage. Download or build one, then add the path to the database under `classify:kraken_db_fp:`.

1.2.3 Contig annotation

Sunbeam can automatically BLAST your contigs against any number of nucleotide or protein databases and summarize the top hits. Download or create your BLAST databases, then add the paths to your config file, following the instructions on here: *blastdbs*.

1.2.4 Reference mapping

If you'd like to map the reads against a set of reference genomes of interest, follow the same method as for the host/contaminant sequences above. Make a folder containing FASTA files for each reference genome, then add the path to that folder in `mapping:genomes_fp:`.

1.3 Running

After you've finished editing your config file, you're ready to run Sunbeam:

```
sunbeam run --configfile my_project/sunbeam_config.yml
```

By default, this will do a lot, including trimming and quality-controlling your reads, removing contaminant, host, and low-complexity sequences, assigning read-level taxonomy, assembling the reads in each sample into contigs, and then BLASTing those contigs against your databases. Each of these steps can also be run independently by adding arguments after the `sunbeam run` command. See *Running* for more info.

1.4 Viewing results

The output is stored by default under `my_project/sunbeam_output`. For more information on the output files and all of Sunbeam's different parts, see our full *User Guide*!

CHAPTER 2

User Guide

Contents

2.1 Installation

Clone the stable branch of Sunbeam and run the installation script:

```
git clone -b stable https://github.com/eclarke/sunbeam sunbeam-stable
cd sunbeam-stable
bash install.sh
```

The installer will check for and install the three components necessary for Sunbeam to work. The first is Conda, a system for downloading and managing software environments. The second is the Sunbeam environment, which will contain all the required software. The third is the Sunbeam library, which provides the necessary commands to run Sunbeam.

All of this is handled for you automatically. If Sunbeam is already installed, you can upgrade either or both the Sunbeam environment and library by passing `--upgrade [env/lib/all]` to the install script.

If you don't have Conda installed prior to this, you will need to add a line (displayed during install) to your config file (usually in `~/.bashrc` or `~/.profile`). Restart your terminal after installation for this to take effect.

2.1.1 Testing

We've included a test script that should verify all the dependencies are installed and Sunbeam can run properly. We strongly recommend running this after installing or updating Sunbeam:

```
bash tests/run_tests.bash
```

If the tests fail, you should either refer to our *troubleshooting* guide or file an issue on our Github page.

2.1.2 Updating

Sunbeam follows semantic versioning practices. In short, this means that the version has three numbers: major, minor and patch. For instance, a version number of 1.2.1 has 1 as the major version, 2 as the minor, and 1 as the patch.

When we update Sunbeam, if your config files and environment will work between upgrades, we will increment the patch or minor numbers (e.g. 1.0.0 -> 1.1.0). All you need to do is the following:

```
git pull
./install.sh --upgrade all
```

If we make a change that affects your config file (such as renaming keys or adding a new section), we will increase the major number (e.g. 1.1.0 -> 2.0.0). When this occurs, you can use the same update procedure as before, and then update your config file to the new format:

```
git pull
./install.sh --upgrade all
source activate sunbeam
sunbeam config upgrade --in_place /path/to/my_config.yml
```

It's a good idea to re-run the tests after this to make sure everything is working.

2.1.3 Uninstalling or reinstalling

If things go awry and updating doesn't work, simply uninstall and reinstall Sunbeam.

```
source deactivate
conda env remove sunbeam
rm -rf sunbeam-stable
```

Then follow the *installation* instructions above.

2.2 Setup

2.2.1 Activating Sunbeam

Almost all commands from this point forward require us to activate the Sunbeam conda environment:

```
source activate sunbeam
```

You should see '(sunbeam)' in your prompt when you're in the environment. To leave the environment, run `source deactivate` or close the terminal.

2.2.2 Creating a new project

We provide a utility, `sunbeam init`, to create a new config file and sample list for a project. The utility takes one required argument: a path to your project folder. This folder will be created if it doesn't exist. You can also specify the path to your gzipped fastq files, and Sunbeam will try to guess how your samples are named, and whether they're paired.

```
sunbeam init --data_fp /path/to/fastq/files /path/to/my_project
```

In this directory, a new config file and a new sample list were created (by default named `sunbeam_config.yml` and `samplelist.csv`, respectively). Edit the config file in your favorite text editor- all the keys are described below.

Note: Sunbeam will do its best to determine how your samples are named in the `data_fp` you specify. It assumes they are named something regular, like `MP66_S109_L008_R1_001.fastq.gz` and `MP66_S109_L008_R2_001.fastq.gz`. In this case, the sample name would be 'MP66_S109_L008' and the read pair indicator would be '1' and '2'. Thus, the filename format would look like `{sample}_R{rp}_001.fastq.gz`, where {sample} defines the sample name and {rp} defines the 1 or 2 in the read pair.

If you have single-end reads, you can pass `--single_end` to `sunbeam init` and it will not try to identify read pairs.

If the guessing doesn't work as expected, you can manually specify the filename format after the `--format` option in `sunbeam init`.

Finally, if you don't have your data ready yet, simply omit the `--data_fp` option. You can create a sample list later with `sunbeam list_samples`.

If some config values are always the same for all projects (e.g. paths to shared databases), you can put these keys in a file and auto-populate your config file with them during initialization. For instance, if your Kraken databases are located at `/shared/kraken/standard`, you could have a file containing the following called `common_values.yml`:

```
classify:
  kraken_db_fp: "/shared/kraken/standard"
```

When you make a new Sunbeam project, use the `--defaults common_values.yml` as part of the init command.

Further usage information is available by typing `sunbeam init --help`.

2.3 Configuration

Sunbeam has lots of configuration options, but most don't need individual attention. Below, each is described by section.

2.3.1 Sections

all

- `root`: The root project folder, used to resolve any relative paths in the rest of the config file.
- `output_fp`: Path to where the Sunbeam outputs will be stored.
- `samplelist_fp`: Path to a comma-separated file where each row contains a sample name and one or two paths (if single- or paired-end) to raw gzipped fastq files. This can be created for you by `sunbeam init` or `sunbeam list_samples`.
- `paired_end`: 'true' or 'false' depending on whether you are using paired- or single-end reads.
- `version`: Automatically added for you by `sunbeam init`. Ensures compatibility with the right version of Sunbeam.

qc

- `suffix`: the name of the subfolder to hold outputs from the quality-control steps
- `threads`: the number of threads to use for rules in this section
- `java_heapsize`: the memory available to Trimmomatic
- `leading`: (trimmomatic) remove the leading bases of a read if below this quality
- `trailing`: (trimmomatic) remove the trailing bases of a read if below this quality
- `slidingwindow`: (trimmomatic) the [width, avg. quality] of the sliding window
- `minlength`: (trimmomatic) drop reads smaller than this length
- `adapter_fp`: (trimmomatic) path to the Illumina paired-end adaptors (autofilled)
- `fwd_adaptors`: (cutadapt) custom forward adaptor sequences to remove using cutadapt. Replace with "" to skip.
- `rev_adaptors`: (cutadapt) custom reverse adaptor sequences to remove using cutadapt. Replace with "" to skip.
- `mask_low_complexity`: [true/false] mask low-complexity sequences with Ns
- `kz_threshold`: a value between 0 and 1 to determine the low-complexity boundary (1 is most stringent). Ignored if not masking low-complexity sequences.
- `kz_window`: window size to use (in bp) for local complexity assessment. Ignored if not masking low-complexity sequences.
- `pct_id`: (decontaminate) minimum percent identity to host genome to consider match

- `frac`: (decontaminate) minimum fraction of the read that must align to consider match
- `host_fp`: the path to the folder with host/contaminant genomes (ending in *.fasta)

classify

- `suffix`: the name of the subfolder to hold outputs from the taxonomic classification steps
- `threads`: threads to use for Kraken
- `kraken_db_fp`: path to Kraken database

assembly

- `suffix`: the name of the folder to hold outputs from the assembly steps
- `min_len`: the minimum contig length to keep
- `threads`: threads to use for the MEGAHIT assembler

annotation

- `suffix`: the name of the folder to hold contig annotation results
- `min_contig_length`: minimum length of contig to annotate (shorter contigs are skipped)
- `circular_kmin`: smallest length of kmers used to search for circularity
- `circular_kmax`: longest length of kmers used to search for circularity
- `circular_min_length`: smallest length of contig to check for circularity

blast

- `threads`: number of threads provided to all BLAST programs

blastdbs

- `root_fp`: path to a directory containing BLAST databases (if they're all in the same place)
- `nucleotide`: the section to define any nucleotide BLAST databases (see tip below for syntax)
- `protein`: the section to define any protein BLAST databases (see tip below)

Tip: The structure for this section allows you to specify arbitrary numbers of BLAST databases of either type. For example, if you had a local copy of nt and a couple of custom protein databases, your section here would look like this (assuming they're all in the same parent directory):

```
blastdbs:
  root_fp: "/local/blast_databases"
  nucleotide:
    nt: "nt/nt"
  protein:
    vfdb: "virulence_factors/virdb"
    card: "/some/other/path/card_db/card"
```

This tells Sunbeam you have three BLAST databases, two of which live in `/local/blast_databases` and a third that lives in `/some/other/path`. It will run nucleotide blast on the nucleotide databases and BLASTX and BLASTP on the protein databases.

mapping

- `suffix`: the name of the subfolder to create for mapping output (bam files, etc)
- `genomes_fp`: path to a directory with an arbitrary number of target genomes upon which to map reads. Genomes should be in FASTA format, and Sunbeam will create the indexes if necessary.
- `threads`: number of threads to use for alignment to the target genomes
- `samtools_opts`: a string added to the `samtools view` command during mapping. This is a good place to add '-F4' to keep only mapped reads and decrease the space these files occupy.

2.4 Running

To run Sunbeam, make sure you've activated the sunbeam environment. Then run:

```
sunbeam run --configfile ~/path/to/config.yml
```

There are many options that you can use to determine which outputs you want. By default, if nothing is specified, this runs the entire pipeline. However, each section is broken up into subsections that can be called individually, and will only execute the steps necessary to get their outputs. These are specified after the command above and consist of the following:

- `all_qc`: basic quality control on all reads (no host read removal)
- `all_decontam`: quality control and host read removal on all samples
- `all_mapping`: align reads to target genomes
- `all_classify`: classify taxonomic provenance of all qc'd, decontaminated reads
- `all_assembly`: build contigs from all qc'd, decontaminated reads
- `all_annotate`: annotate contigs using defined BLAST databases

To use one of these options, simply run it like so:

```
sunbeam run -- --configfile ~/path/to/config.yml all_classify
```

In addition, since Sunbeam is really just a set of snakemake rules, all the (many) snakemake options apply here as well. Some useful ones are:

- `-n` performs a dry run, and will just list which rules are going to be executed without actually doing so.
- `-k` allows the workflow to continue with unrelated rules if one produces an error (useful for malformed samples, which can also be added to the `exclude` config option).
- `-p` prints the actual shell command executed for each rule, which is very helpful for debugging purposes.

2.4.1 Cluster options

Sunbeam inherits its cluster abilities from Snakemake. There's nothing special about installing Sunbeam on a cluster, but in order to distribute work to cluster nodes, you have to use the `--cluster` and `--jobs` flags. For example, if we wanted each rule to run on a 12-thread node, and a max of 100 rules executing in parallel, we would use the following command on our cluster:

```
sunbeam run -- --configfile ~/path/to/config.yml --cluster "bsub -n 12" -j 100 -w 90
```

The `-w 90` flag is provided to account for filesystem latency that often causes issues on clusters. It asks Snakemake to wait for 90 seconds before complaining that an expected output file is missing.

2.5 Outputs

This section describes all the outputs from Sunbeam. Here is an example output directory, where we had two samples (sample1 and sample2), and two BLAST databases, one nucleotide ('bacteria') and one protein ('card').

```
sunbeam_output
     ├── annotation
     │   ├── blastn
     │   │   └── bacteria
     │   │       └── contig
     │   ├── blastp
     │   │   └── card
     │   │       └── prodigal
     │   ├── blastx
     │   │   └── card
     │   │       └── prodigal
     │   ├── genes
     │   │   └── prodigal
     │   │       └── log
     │   └── summary
     ├── assembly
     │   ├── contigs
     ├── classify
     │   └── kraken
     │       └── raw
     ├── mapping
     │   └── genome1
     └── qc
         ├── cleaned
         ├── decontam
         ├── log
         │   ├── decontam
         │   ├── cutadapt
         │   └── trimmomatic
         └── reports
```

In order of appearance, the folders contain the following:

2.5.1 Contig annotation

```
sunbeam_output
      ├── annotation
      │   ├── blastn
      │   │   └── bacteria
      │   │       └── contig
      │   ├── blastp
      │   │   └── card
      │   │       └── prodigal
      │   ├── blastx
      │   │   └── card
      │   │       └── prodigal
      │   ├── genes
      │   │   └── prodigal
      │   │       └── log
      │   └── summary
```

This contains the BLAST results in XML from the assembled contigs. `blastn` contains the results from directly BLASTing the contig nucleotide sequences against the nucleotide databases. `blastp` and `blastx` use genes identified by the ORF finding program Prodigal to search for hits in the protein databases.

The genes found from Prodigal are available in the `genes` folder.

Finally, the `summary` folder contains an aggregated report of the number and types of hits of each contig against the BLAST databases, as well as length and circularity.

2.5.2 Contig assembly

```
├── assembly
│   ├── contigs
```

This contains the assembled contigs for each sample under 'contigs'.

2.5.3 Taxonomic classification

```
├── classify
│   └── kraken
│       └── raw
```

This contains the taxonomic outputs from Kraken, both the raw output as well as summarized results. The primary output file is `all_samples.tsv`, which is a BIOM-style format with samples as columns and taxonomy IDs as rows, and number of reads assigned to each in each cell.

2.5.4 Alignment to genomes

```
├── mapping
│   └── genome1
```

Alignment files (in BAM format) to each target genome are contained in subfolders named for the genome, such as 'genome1'.

2.5.5 Quality control

```
└── qc
    ├── cleaned
    ├── decontam
    ├── log
    │   ├── decontam
    │   ├── cutadapt
    │   └── trimmomatic
    └── reports
```

This folder contains the trimmed, low-complexity filtered reads in `cleaned`. The `decontam` folder contains the cleaned reads that did not map to any contaminant or host genomes. In general, most downstream steps should reference the `decontam` reads.

2.6 Troubleshooting

Coming soon. For now we suggest browsing the closed issues tab on Github.

CHAPTER 3

Sunbeam Extensions