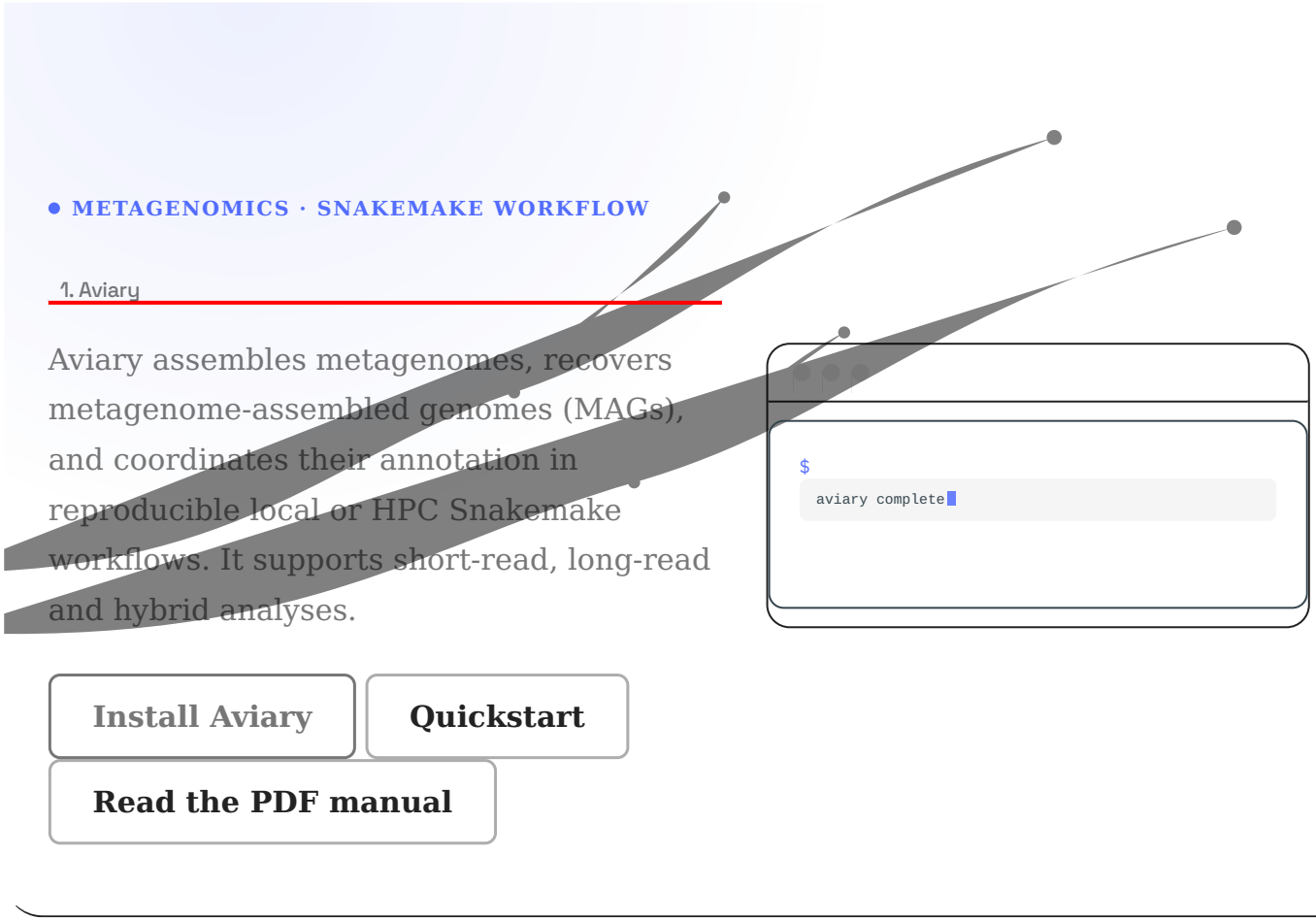
Aviary

Metagenomic assembly, genome recovery and annotation

Aviary contributors

Contents



1.1 A complete analysis in one command

```
aviary complete \
  -1 sample_R1.fastq.gz \
  -2 sample_R2.fastq.gz \
  --output sample_aviary \
  --max-threads 8 \
  --n-cores 8
```

See the first complete analysis walkthrough for what each stage does and where its outputs land.

1.2 The commands

Six analysis subcommands cover the workflow. `complete` runs the metagenome stages end to end; the others let you enter or leave at a specific point. Two utility commands, `configure` and `build`, manage reference paths and dependency environments.

`complete`

READS → ANNOTATED MAGS

Every stage in one command: assembly, binning, refinement, then annotation. Works from short reads, long reads, or both. Already have an assembly? Pass `--assembly` and Aviary picks up from binning instead of starting over.

`assemble`

READS → CONTIGS

Step-down hybrid assembly using long and short reads together, playing each to its strengths — long reads for contiguity, short reads for accuracy. Either type alone works too. Several short-read samples with no long reads are co-assembled with MEGAHIT or metaSPAdes.

`recover`

ASSEMBLY → MAGS

Sorts contigs into metagenome-assembled genomes using several binning algorithms rather than trusting any single one, then refines the result and reports quality and taxonomy per MAG. With no assembly it runs the assembly pipeline first; with several it enables SemiBin2 multi-sample binning.

`annotate`

MAGS → ANNOTATIONS

Adds functional annotations with EggNOG and taxonomic classifications with GTDB-Tk. Assemblies can be passed alongside when targeting QUAST QC. CheckM2 quality assessment belongs to the `recover` / `complete` binning path, not the default annotation target.

`cluster`

MANY RUNS → REPRESENTATIVES

Run enough samples and the same organism turns up repeatedly. Galah collapses those near-duplicates across finished Aviary runs and picks one representative genome per cluster — 97% ANI by default, adjustable with `--ani`.

`isolate`

PURE CULTURE → GENOME

A long-read-first Flye assembly and polishing path for a single organism from pure culture. Optional short reads add another polishing round. For metagenomic data, use `assemble` or `recover` instead.

1.3 Find what you need

Guide

How reads become assemblies, candidate genomes and annotated MAGs.

Worked analyses

End-to-end examples with real commands.

CLI reference

Every command, option, default and output.

Troubleshooting

Validation messages, failed rules, resource limits.

Using Aviary in research? See the citation guide for Aviary and its upstream tools. Aviary is released under GPL-3.0.

2. Installation

Aviary targets Linux and uses isolated dependency environments for its bioinformatics tools. Three install routes are documented below — most users only need one:

Which install route?

- **Most users:** Pixi — the recommended, supported route.
- **Already managing tool dependencies yourself:** pip.
- **Prefer a Conda-style environment:** Bioconda.

2.1 Install from source with Pixi

From a local checkout:

```
cd aviary
pixi run postinstall
pixi run aviary --version
```

This installs Aviary in editable mode. Dependency definitions live in `aviary/pixi.toml`, while `aviary/pixi.lock` pins the resolved environments. The `postinstall` task prepares both the main and development environments.

For a shared development system, database paths can be represented by symlinks in the repository's `db/` directory; `admin/set_env_vars.sh` documents the expected local names used by the activation hook.

2.2 Install from pip

The Python package name is `aviary-genome`, but pip alone does not provision the complete collection of external bioinformatics tools. Use this route only when you are deliberately managing those dependencies yourself:

```
conda env create -n aviary -f admin/environment.yml
conda activate aviary
pip install aviary-genome
```

2.3 Install from Bioconda

Create a dedicated environment:

```
conda create -n aviary -c conda-forge -c bioconda aviary
conda activate aviary
```

If your Conda channels are configured globally, keep this priority:

```
channels:
  - conda-forge
  - bioconda
  - defaults
```

Installing into a new environment avoids dependency conflicts with unrelated analysis software.

2.4 Verify the command

```
aviary --version
aviary complete --full-help
```

Before downloading large databases, use `aviary complete --full-help` to confirm that the installed release exposes the options used by this documentation.

2.5 Build analysis environments

After installation, Aviary can prepare its per-tool environments:

```
aviary build
```

GPU environments are optional and require compatible hardware and drivers:

```
aviary build --gpu
```

2.6 Configure reference data

Aviary can use local data for GTDB-Tk, EggNOG-mapper, CheckM2, SingleM and Metabuli. Which resources are required depends on the workflow stages and options selected. Place shared databases outside individual run directories; on an HPC system, coordinate their location with the system administrator.

Configure existing resources explicitly:

```
aviary configure \
  --gtdb-path /shared/db/gtdb/release232 \
  --eggnog-db-path /shared/db/eggnog \
  --checkm2-db-path /shared/db/checkm2/uniref100.KO.1.dmnd \
  --singlem-metapackage-path /shared/db/singlem/package.smpkg.zb \
  --metabuli-db-path /shared/db/metabuli
```

The corresponding environment variables are:

Variable	Consumer
`GTDBTK_DATA_PATH`	GTDB-Tk taxonomy
`EGGNOG_DATA_DIR`	EggNOG functional annotation
`CHECKM2DB`	CheckM2 genome quality assessment
`SINGLEM_METAPACKAGE_PATH`	SingleM community/recovery analysis
`METABULI_DB_PATH`	Metabuli classification used by applicable workflows
`TMPDIR`	Temporary-file location

To ask Aviary to download configured resources, add `--download` followed by one or more of `gtdb`, `eggnog`, `singlem`, `checkm2` and `metabuli`.

```
aviary configure \
  --gtdb-path /shared/db/gtdb \
  --checkm2-db-path /shared/db/checkm2 \
  --download gtdb checkm2
```

Large downloads

Database downloads are large and should not be duplicated per user or run. Confirm available storage and release requirements before starting them. **Supplying `--download` with no values requests all five databases at once** — always list the specific databases you need.

Continue with the quickstart. For production clusters, see HPC and cluster submission.

3. Quickstart

This example runs the complete short-read metagenome workflow. It assumes Aviary is installed and the required databases are configured.

3.1 1. Check the installation

```
aviary --version
aviary complete --full-help
```

3.2 2. Prepare paired reads

Use one forward and one reverse FASTQ file for the same sample:

```
reads/
├── sample_R1.fastq.gz
└── sample_R2.fastq.gz
```

Compressed FASTQ input is accepted. Forward and reverse files must be supplied in matching order when more than one pair is given.

3.3 3. Preview the workflow

```
aviary complete \
  -1 reads/sample_R1.fastq.gz \
  -2 reads/sample_R2.fastq.gz \
  --output sample_aviary \
  --max-threads 8 \
  --n-cores 8 \
  --dry-run
```

The dry run resolves the workflow without executing analysis tools. Remove `--dry-run` after checking the planned jobs and configured database paths.

3.4 4. Run

```
aviary complete \
  -1 reads/sample_R1.fastq.gz \
  -2 reads/sample_R2.fastq.gz \
  --output sample_aviary \
  --max-threads 8 \
  --n-cores 8
```

`--max-threads` limits an individual tool. `--n-cores` is the total CPU capacity available to Snakemake, which may schedule several compatible jobs at once.

3.5 5. Inspect the results

Start with:

```
sample_aviary/
├── assembly/final_contigs.fasta
├── bins/bin_info.tsv
├── bins/final_bins/
├── benchmarks/
└── logs/
```

Read `bin_info.tsv` and the rest of the output layout before using recovered genomes downstream. If a rule fails, find its log in `sample_aviary/logs/` and see troubleshooting.

3.6 Next step

The first complete analysis explains how the stages behave and which controls are most important.

4. First complete analysis

`aviary complete` joins assembly, genome recovery and annotation into one Snakemake run. Use it when you want Aviary to carry raw reads through to a curated set of metagenome-assembled genomes (MAGs).

4.1 Data flow

```
paired short reads ─┐
                    ├─ quality control ─ assembly ─ read mapping
optional long reads ┘                          │
                                               ▼
                          binning ─ refinement ─ quality assessment
                                               │
                                               ▼
                                taxonomy and functional annotation
```

Aviary orchestrates these stages; specialised dependencies perform the underlying assembly, binning, quality assessment and annotation algorithms.

4.2 Choose the input mode

Data available	Required input	Typical use
Paired short reads	`-1`, `-2`	Short-read metagenome assembly and recovery
Long reads	`--longreads`, `--long-read-type`	Long-read assembly and recovery
Both	short- and long-read options	Hybrid assembly and recovery
Existing assembly	`--assembly` plus reads for coverage	Skip de novo assembly and recover MAGs

See the input reference for pairing rules and accepted long-read type identifiers.

4.3 Run in a dedicated output directory

```
aviary complete \
  -1 reads/sample_R1.fastq.gz \
  -2 reads/sample_R2.fastq.gz \
  --longreads reads/sample_ont.fastq.gz \
  --long-read-type ont \
  --output sample_aviary \
  --max-threads 16 \
  --n-cores 32 \
  --max-memory 250
```

The default maximum memory value in the workflow configuration is 250 GB. Set the option to the actual hard limit available to the run; it is not a prediction of typical memory consumption.

4.4 Monitor and resume

Rule-specific messages are written beneath `logs/`, while Snakemake benchmark records are written beneath `benchmarks/`. Re-running the same command and output directory normally resumes from existing valid outputs. Do not delete `.snakemake/` or intermediate files while diagnosing an interrupted run.

For scheduler submission, resource caps and retries, see HPC and cluster submission. For precise command options, see the `complete` CLI reference.

5. Guide

5.1 Guide

The Aviary workflow follows sequencing data through three principal stages:

```
reads → assembly → genome recovery → annotation
```

Read core concepts first, then use the stage guide matching your analysis. The workflow-control guide explains dry runs, targeted rules, resuming and cleanup.

5.2 Core concepts

Aviary is a workflow orchestrator for metagenomic assembly and genome recovery. Understanding four distinctions makes its commands and outputs easier to use.

5.2.1 Workflow commands describe stopping points

`assemble`, `recover` and `annotate` are composable stages. `complete` runs the full path permitted by the supplied inputs, while `isolate` uses an assembly path intended for a cultured isolate rather than a mixed community.

Command	Begins with	Principal result
`assemble`	sequencing reads	assembled contigs
`recover`	assembly plus reads used for coverage	recovered MAGs
`annotate`	MAG FASTA files	taxonomy and functional annotations
`complete`	reads or an existing assembly	results through annotation
`cluster`	completed Aviary runs	dereplicated representative genomes
`isolate`	reads from a cultured isolate	isolate assembly

5.2.2 Assemblies, bins and MAGs

An assembly joins overlapping sequence evidence into contigs. Metagenomic binning then groups contigs that appear to originate from the same population. A final bin is treated as a metagenome-assembled genome (MAG), but its completeness, contamination and taxonomy remain estimates. Use `bins/bin_info.tsv` to assess each recovered genome rather than treating all FASTA files as equivalent.

5.2.3 Aviary and upstream tools

Aviary owns workflow decisions: it validates command-line input, writes the run configuration, selects Snakemake targets, passes resource limits and links final products into stable output locations. Upstream programs perform the underlying read filtering, assembly, mapping, binning, quality estimation, taxonomic classification and functional annotation. The exact tools invoked depend on command options and available data.

5.2.4 Requested resources have different scopes

`--max-threads` is the maximum made available to one tool. `--n-cores` is the total capacity Snakemake may schedule concurrently. `--local-cores` limits work kept on the coordinator node when using a cluster profile. `--max-memory` is a hard workflow cap in gigabytes, not a guarantee that every tool uses that amount.

5.2.5 Workflow state and resumability

Snakemake determines whether an output is current from its inputs, rules and metadata. A repeated Aviary command against the same output directory normally continues incomplete work. Options such as `--rerun-triggers`, `--clean` and `--unlock` deliberately change this behaviour; use them only after reading the workflow-control guide.

5.2.6 Inputs and outputs

Aviary accepts FASTQ sequencing reads and FASTA assemblies or genomes. BAM files are produced internally when reads are mapped to assemblies. Final results are presented through stable directories such as `assembly/`, `bins/`, `taxonomy/` and `annotation/`; working files remain in `data/`.

Continue with the assembly guide, genome-recovery guide, or annotation guide.

5.3 Metagenome assembly

Assembly transforms quality-controlled reads into contigs for genome recovery or other downstream analysis. Aviary supports short-read, long-read and hybrid inputs and chooses applicable workflow rules from the data supplied.

5.3.1 Select an input strategy

- Paired short reads require matching `-1` and `-2` lists.
- Long reads require `--longreads` and a `--long-read-type` value.
- Hybrid assembly supplies both short and long reads.
- Multiple samples can be co-assembled where the selected options support it.

Input quality control happens before assembly unless it is explicitly skipped. Host references supplied through `--host-filter` are used to remove matching reads before the assembly stage.

5.3.2 Basic usage

```
aviary assemble \
  -1 sample_R1.fastq.gz \
  -2 sample_R2.fastq.gz \
  --output assembly_run \
  --max-threads 16 \
  --n-cores 16
```

For a hybrid dataset, add long reads and their platform type:

```
aviary assemble \
  -1 sample_R1.fastq.gz \
  -2 sample_R2.fastq.gz \
  --longreads sample_ont.fastq.gz \
  --long-read-type ont \
  --output hybrid_run
```

5.3.3 Important controls

`--long-read-assembler` selects the supported long-read assembly implementation. Quality-control thresholds affect which reads reach the assembler and should be chosen for the sequencing data, not copied mechanically between projects. `--min-contig-size` is accepted by `assemble` through its shared parser, but it is a downstream binning filter and does not alter the default assembly target; set it on `recover` or `complete` when filtering contigs for binning.

5.3.4 Result

The canonical assembly is stored in `data/final_contigs.fasta` and exposed as `assembly/final_contigs.fasta`. Reports are written under `www/`; rule logs and benchmarks are kept in `logs/` and `benchmarks/`.

See the `assemble` reference for every option and the output reference for the run layout.

5.4 Genome recovery

Genome recovery groups assembled contigs into candidate genomes, refines the candidate sets and evaluates the resulting MAGs. Reads are still important: mapping them back to the assembly provides coverage information used by binning and abundance calculations.

5.4.1 Basic usage

```
aviary recover \
  --assembly assembly/final_contigs.fasta \
  -1 sample_R1.fastq.gz \
  -2 sample_R2.fastq.gz \
  --output recovery_run \
  --max-threads 16 \
  --n-cores 32
```

5.4.2 How the stage behaves

Aviary prepares coverage inputs, invokes the enabled binners, refines candidate bins and combines evidence into a final set. Quality assessment and taxonomy then add evidence needed to interpret each MAG. A binning algorithm producing a FASTA file does not by itself establish that the genome is complete, uncontaminated or correctly classified.

5.4.3 Important controls

`--min-bin-size` excludes candidate genomes smaller than the configured number of bases from later processing. Binner selection options change which upstream algorithms contribute candidates. GPU variants require a compatible GPU and the corresponding Aviary environments; enabling a GPU flag does not make every stage GPU accelerated.

SemiBin2 multi-sample mode accepts multiple assemblies and should be used only with inputs organised for that mode. See the exact validation and accepted values in the `recover` reference.

5.4.4 Interpreting results

Use `bins/final_bins/` as the recovered FASTA collection and `bins/bin_info.tsv` as its main summary. Retain the run's database versions and software environment because taxonomy and quality estimates can change with reference data and dependency versions.

Continue with annotation, the output reference, or reproducibility guidance.