kb-python

Release 0.27.3

Kyung Hoi (Joseph) Min

Jul 08, 2022

CONTENTS:

This page contains **DEVELOPER** documentation for `kb-python` version `0.27.3`. For user documentation and tutorials, please go to kallisto | bustools.

CHAPTER

ONE

DEVELOPMENT PREREQUISITES

There are a couple of things you must set up on your machine so that all of your commits satisfy code quality and unit-testing requirements. First, install all necessary packages by running:

```
pip install -r requirements.txt
pip install -r dev-requirements.txt
```

Code qualty and unit tests are strictly enforced for every pull request via Github actions.

1.1 Code Quality

`kb-python` uses `flake8` and `yapf` to ensure code quality. The easiest way to set these up so that they run automatically for every commit is to install `pre-commit` hooks by running:

```
pre-commit install
```

at the root of the repository.

1.2 Unit-testing

`kb-python` uses `nose` to run unit tests. There is a convenient Makefile rule in place to run all tests.:

```
make test
```

CHAPTER

TWO

RELEASING NEW VERSIONS

This section walks you through, step-by-step, how to release a new version.

1. Make sure you are on the up-to-date `master` branch.
2. Run `make bump_patch`, `make bump_minor`, or `make bump_major` depending on what version you will be bumping.
3. Run `make push_release`. This will push the new commit and tag.
4. Go to the *releases* tab on Github.
5. Select the new release, edit the release description, and *Publish release*.
6. A Github action will automatically trigger to upload the new release to PyPi.

2.1 API Reference

This page contains auto-generated API reference documentation[1].

2.1.1 `kb_python`

Subpackages

`kb_python.dry`

Submodules

`kb_python.dry.count`

Module Contents

Functions

`stream_batch`(batch_path: str, temp_dir: str = 'tmp') → str	Dry version of *count.stream_batch*.
`write_smartseq3_capture`(capture_path: str) → str	Dry version of *count.write_smartseq3_capture*.

[1] Created with sphinx-autoapi

`kb_python.dry.count.`**`stream_batch`**(*batch_path: str*, *temp_dir: str = 'tmp'*) → str

Dry version of *count.stream_batch*.

`kb_python.dry.count.`**`write_smartseq3_capture`**(*capture_path: str*) → str

Dry version of *count.write_smartseq3_capture*.

`kb_python.dry.utils`

Module Contents

Functions

`run_executable`(command: List[str], quiet: bool = False, *args, **kwargs)	Dry version of *utils.run_executable*.
`make_directory`(path: str)	Dry version of *utils.make_directory*.
`remove_directory`(path: str)	Dry version of *utils.remove_directory*.
`stream_file`(url: str, path: str) → str	Dry version of *utils.stream_file*.
`move_file`(source: str, destination: str) → str	Dry version of *utils.move_file*.
`copy_whitelist`(technology: str, out_dir: str) → str	Dry version of *utils.copy_whitelist*.
`create_10x_feature_barcode_map`(out_path: str) → str	Dry version of *utils.create_10x_feature_barcode_map*.
`get_temporary_filename`(temp_dir: str) → str	Dry version of *utils.get_temporary_filename*.

`kb_python.dry.utils.`**`run_executable`**(*command: List[str]*, *quiet: bool = False*, **args*, ***kwargs*)

Dry version of *utils.run_executable*.

`kb_python.dry.utils.`**`make_directory`**(*path: str*)

Dry version of *utils.make_directory*.

`kb_python.dry.utils.`**`remove_directory`**(*path: str*)

Dry version of *utils.remove_directory*.

`kb_python.dry.utils.`**`stream_file`**(*url: str*, *path: str*) → str

Dry version of *utils.stream_file*.

`kb_python.dry.utils.`**`move_file`**(*source: str*, *destination: str*) → str

Dry version of *utils.move_file*.

`kb_python.dry.utils.`**`copy_whitelist`**(*technology: str*, *out_dir: str*) → str

Dry version of *utils.copy_whitelist*.

`kb_python.dry.utils.`**`create_10x_feature_barcode_map`**(*out_path: str*) → str

Dry version of *utils.create_10x_feature_barcode_map*.

`kb_python.dry.utils.`**`get_temporary_filename`**(*temp_dir: str*) → str

Dry version of *utils.get_temporary_filename*.

Package Contents

Functions

is_dry() → bool	Return whether the current run is a dry run.
dryable(dry_func: Callable) → Callable	Function decorator to set a function as dryable.
dummy_function(*args, **kwargs)	A dummy function that doesn't do anything and just returns.
undryable_function(*args, **kwargs)	A dummy function that raises an exception. For use when a particular

kb_python.dry.**is_dry**() → bool

Return whether the current run is a dry run.

Returns
Whether the current run is a dry run

kb_python.dry.**dryable**(*dry_func: Callable*) → Callable

Function decorator to set a function as dryable.

When this decorator is applied, the provided *dry_func* will be called instead of the actual function when the current run is a dry run.

Parameters
dry_func – Function to call when it is a dry run

Returns
Wrapped function

kb_python.dry.**dummy_function**(**args*, ***kwargs*)

A dummy function that doesn't do anything and just returns. Used for making functions dryable.

kb_python.dry.**undryable_function**(**args*, ***kwargs*)

A dummy function that raises an exception. For use when a particular function is not dryable.

Raises
Exception – Always

Submodules

kb_python.compile

Module Contents

Functions

get_latest_github_release_tag(releases_url: str) → str	Get the tag name of the latest GitHub release, given a url to the
get_filename_from_url(url: str) → str	Fetch the filename from a URL.
get_kallisto_url(ref: Optional[str] = None) → str	Get the tarball url of the specified or latest kallisto release.
get_bustools_url(ref: Optional[str] = None) → str	Get the tarball url of the specified or latest bustools release.
find_git_root(path: str) → str	Find the root directory of a git repo by walking.
compile_kallisto(source_dir: str, binary_path: str, cmake_arguments: Optional[str] = None) → str	Compile *kallisto* from source.
compile_bustools(source_dir: str, binary_path: str, cmake_arguments: Optional[str] = None) → str	Compile *bustools* from source.
compile(target: typing_extensions.Literal[kallisto, bustools, all], out_dir: Optional[str] = None, cmake_arguments: Optional[str] = None, url: Optional[str] = None, ref: Optional[str] = None, overwrite: bool = False, temp_dir: str = 'tmp') → Dict[str, str]	Compile *kallisto* and/or *bustools* binaries by downloading and compiling

exception kb_python.compile.**CompileError**

Bases: `Exception`

Common base class for all non-exit exceptions.

kb_python.compile.**get_latest_github_release_tag**(*releases_url: str*) → str

Get the tag name of the latest GitHub release, given a url to the releases API.

Parameters
releases_url – Url to the releases API

Returns
The tag name

kb_python.compile.**get_filename_from_url**(*url: str*) → str

Fetch the filename from a URL.

Parameters
url – The url

Returns
The filename

kb_python.compile.**get_kallisto_url**(*ref: Optional[str] = None*) → str

Get the tarball url of the specified or latest kallisto release.

Parameters
ref – Commit or release tag, defaults to *None*. By default, the most recent release is used.

Returns
Tarball url

kb_python.compile.**get_bustools_url**(*ref: Optional[str] = None*) → str

Get the tarball url of the specified or latest bustools release.

Parameters
ref – Commit or release tag, defaults to *None*. By default, the most recent release is used.

Returns
Tarball url

`kb_python.compile.`**`find_git_root`**`(`*path: str*`)` → str

Find the root directory of a git repo by walking.

Parameters
`path` – Path to start the search

Returns
Path to root of git repo

Raises
`CompileError` – If the git root could not be found

`kb_python.compile.`**`compile_kallisto`**`(`*source_dir: str*, *binary_path: str*, *cmake_arguments: Optional[str] = None*`)` → str

Compile *kallisto* from source.

Parameters

- **`source_dir`** – Path to directory containing root of kallisto git repo
- **`binary_path`** – Path to place compiled binary
- **`cmake_arguments`** – Additional arguments to pass to the cmake command

Returns
Path to compiled binary

`kb_python.compile.`**`compile_bustools`**`(`*source_dir: str*, *binary_path: str*, *cmake_arguments: Optional[str] = None*`)` → str

Compile *bustools* from source.

Parameters

- **`source_dir`** – Path to directory containing root of bustools git repo
- **`binary_path`** – Path to place compiled binary
- **`cmake_arguments`** – Additional arguments to pass to the cmake command

Returns
Path to compiled binary

`kb_python.compile.`**`compile`**`(`*target: typing_extensions.Literal[kallisto, bustools, all]*, *out_dir: Optional[str] = None*, *cmake_arguments: Optional[str] = None*, *url: Optional[str] = None*, *ref: Optional[str] = None*, *overwrite: bool = False*, *temp_dir: str = 'tmp'*`)` → Dict[str, str]

Compile *kallisto* and/or *bustools* binaries by downloading and compiling a source archive.

Parameters

- **`target`** – Which binary to compile. May be one of *kallisto*, *bustools* or *all*
- **`out_dir`** – Path to output directory, defaults to *None*
- **`cmake_arguments`** – Additional arguments to pass to the cmake command
- **`url`** – Download the source archive from this url instead, defaults to *None*
- **`ref`** – Commit hash or tag to use, defaults to *None*
- **`overwrite`** – Overwrite any existing results, defaults to *False*

- **temp_dir** – Path to temporary directory, defaults to *tmp*

Returns
Dictionary of results

kb_python.config

Module Contents

Classes

Technology	Typed version of namedtuple.

Functions

get_provided_kallisto_path() → Optional[str]	Finds platform-dependent kallisto binary included with the installation.
get_provided_bustools_path() → Optional[str]	Finds platform-dependent bustools binary included with the installation.
get_compiled_kallisto_path(alias: str = COMPILED_DIR) → Optional[str]	Finds platform-dependent kallisto binary compiled with *compile*.
get_compiled_bustools_path(alias: str = COMPILED_DIR) → Optional[str]	Finds platform-dependent bustools binary compiled with *compile*.
get_kallisto_binary_path() → str	Dummy function that simply returns the current value of *KALLISTO_PATH*.
get_bustools_binary_path() → str	Dummy function that simply returns the current value of *BUSTOOLS_PATH*.
set_kallisto_binary_path(path: str)	Helper function to set the *KALLISTO_PATH* variable. Automatically
set_bustools_binary_path(path: str)	Helper function to set the *BUSTOOLS_PATH* variable. Automatically
set_dry()	Set this run to be a dry run.
is_dry() → bool	Return whether the current run is a dry run.
no_validate()	Turn off validation.
is_validate() → bool	Return whether validation is turned on.

Attributes

PACKAGE_PATH
PLATFORM
BINS_DIR
COMPILED_DIR
TEMP_DIR
DRY
VALIDATE
GITHUB_API_URL
KALLISTO_REPO_URL
BUSTOOLS_REPO_URL
KALLISTO_RELEASES_URL
BUSTOOLS_RELEASES_URL
KALLISTO_TARBALL_URL
BUSTOOLS_TARBALL_URL
KALLISTO_PATH
BUSTOOLS_PATH
TECHNOLOGIES
TECHNOLOGIES_MAPPING
Reference
REFERENCES
REFERENCES_MAPPING

kb_python.config.**PACKAGE_PATH**

kb_python.config.**PLATFORM**

kb_python.config.**BINS_DIR**

kb_python.config.**COMPILED_DIR**

`kb_python.config.TEMP_DIR = tmp`

`kb_python.config.DRY = False`

`kb_python.config.VALIDATE = True`

`kb_python.config.GITHUB_API_URL = https://api.github.com`

`kb_python.config.KALLISTO_REPO_URL`

`kb_python.config.BUSTOOLS_REPO_URL`

`kb_python.config.KALLISTO_RELEASES_URL`

`kb_python.config.BUSTOOLS_RELEASES_URL`

`kb_python.config.KALLISTO_TARBALL_URL`

`kb_python.config.BUSTOOLS_TARBALL_URL`

`kb_python.config.get_provided_kallisto_path()` → Optional[str]

Finds platform-dependent kallisto binary included with the installation.

Returns
Path to the binary, *None* if not found

`kb_python.config.get_provided_bustools_path()` → Optional[str]

Finds platform-dependent bustools binary included with the installation.

Returns
Path to the binary, *None* if not found

`kb_python.config.get_compiled_kallisto_path`(*alias: str = COMPILED_DIR*) → Optional[str]

Finds platform-dependent kallisto binary compiled with *compile*.

Parameters
`Alias` – Alias of compiled binary.

Returns
Path to the binary, *None* if not found

`kb_python.config.get_compiled_bustools_path`(*alias: str = COMPILED_DIR*) → Optional[str]

Finds platform-dependent bustools binary compiled with *compile*.

Parameters
`Alias` – Alias of compiled binary.

Returns
Path to the binary, *None* if not found

`kb_python.config.KALLISTO_PATH`

`kb_python.config.BUSTOOLS_PATH`

`class kb_python.config.Technology`

Bases: `NamedTuple`

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

name :str

description :str

chemistry :ngs_tools.chemistry.Chemistry

show :bool = True

kb_python.config.**TECHNOLOGIES**

kb_python.config.**TECHNOLOGIES_MAPPING**

kb_python.config.**Reference**

kb_python.config.**REFERENCES**

kb_python.config.**REFERENCES_MAPPING**

exception kb_python.config.**UnsupportedOSError**

Bases: Exception

Common base class for all non-exit exceptions.

exception kb_python.config.**ConfigError**

Bases: Exception

Common base class for all non-exit exceptions.

kb_python.config.**get_kallisto_binary_path**() → str

Dummy function that simply returns the current value of *KALLISTO_PATH*.

kb_python.config.**get_bustools_binary_path**() → str

Dummy function that simply returns the current value of *BUSTOOLS_PATH*.

kb_python.config.**set_kallisto_binary_path**(*path: str*)

Helper function to set the *KALLISTO_PATH* variable. Automatically finds the full path to the executable and sets that as *KALLISTO_PATH*.

Parameters
path – Path to the kallisto binary

Raises

ConfigError – If *path* could not be resolved or if the executable is not executable.

`kb_python.config.set_bustools_binary_path(path: str)`

Helper function to set the *BUSTOOLS_PATH* variable. Automatically finds the full path to the executable and sets that as *BUSTOOLS_PATH*.

Parameters

path – Path to the bustools binary

Raises

ConfigError – If *path* could not be resolved or if the executable is not executable.

`kb_python.config.set_dry()`

Set this run to be a dry run.

`kb_python.config.is_dry()` → bool

Return whether the current run is a dry run.

Returns

Whether the current run is a dry run

`kb_python.config.no_validate()`

Turn off validation.

`kb_python.config.is_validate()` → bool

Return whether validation is turned on.

Returns

Whether validation is on

kb_python.constants

Module Contents

`kb_python.constants.INFO_FILENAME = info.txt`

`kb_python.constants.CDNA_FILENAME = cdna.fa`

`kb_python.constants.INTRON_FILENAME = introns.fa`

`kb_python.constants.SORTED_FASTA_FILENAME = sorted.fa`

`kb_python.constants.SORTED_GTF_FILENAME = sorted.gtf`

`kb_python.constants.COMBINED_FILENAME = combined.fa`

`kb_python.constants.INDEX_FILENAME = transcriptome.idx`

`kb_python.constants.WHITELIST_FILENAME = whitelist.txt`

`kb_python.constants.FILTER_WHITELIST_FILENAME = filter_barcodes.txt`

`kb_python.constants.INSPECT_FILENAME = inspect.json`

`kb_python.constants.BUS_FILENAME = output.bus`

`kb_python.constants.BUS_S_FILENAME = output.s.bus` kb_python.constants.**BUS_SC_FILENAME = output.s.c.bus**

kb_python.constants.**BUS_UNFILTERED_FILENAME = output.unfiltered.bus**

kb_python.constants.**BUS_FILTERED_FILENAME = output.filtered.bus**

kb_python.constants.**BUS_CDNA_PREFIX = spliced**

kb_python.constants.**BUS_INTRON_PREFIX = unspliced**

kb_python.constants.**ECMAP_FILENAME = matrix.ec**

kb_python.constants.**TXNAMES_FILENAME = transcripts.txt**

kb_python.constants.**KB_INFO_FILENAME = kb_info.json**

kb_python.constants.**KALLISTO_INFO_FILENAME = run_info.json**

kb_python.constants.**REPORT_NOTEBOOK_FILENAME = report.ipynb**

kb_python.constants.**REPORT_HTML_FILENAME = report.html**

kb_python.constants.**COUNTS_PREFIX = cells_x_genes**

kb_python.constants.**TCC_PREFIX = cells_x_tcc**

kb_python.constants.**FEATURE_PREFIX = cells_x_features**

kb_python.constants.**ADATA_PREFIX = adata**

kb_python.constants.**GENE_NAME = gene**

kb_python.constants.**FEATURE_NAME = feature**

kb_python.constants.**TRANSCRIPT_NAME = transcript**

kb_python.constants.**UNFILTERED_COUNTS_DIR = counts_unfiltered**

kb_python.constants.**FILTERED_COUNTS_DIR = counts_filtered**

kb_python.constants.**CELLRANGER_DIR = cellranger**

kb_python.constants.**CELLRANGER_MATRIX = matrix.mtx**

kb_python.constants.**CELLRANGER_BARCODES = barcodes.tsv**

kb_python.constants.**CELLRANGER_GENES = genes.tsv**

kb_python.constants.**BUS_UNFILTERED_SUFFIX = .unfiltered.bus**

kb_python.constants.**BUS_FILTERED_SUFFIX = .filtered.bus**

kb_python.constants.**FLENS_FILENAME = flens.txt**

kb_python.constants.**BATCH_FILENAME = batch.txt**

kb_python.constants.**ABUNDANCE_GENE_FILENAME = matrix.abundance.gene.mtx**

kb_python.constants.**ABUNDANCE_GENE_TPM_FILENAME = matrix.abundance.gene.tpm.mtx**

kb_python.constants.**ABUNDANCE_FILENAME = matrix.abundance.mtx** kb_python.constants.**ABUNDANCE_TPM_FILENAME = matrix.abundance.tpm.mtx**

kb_python.constants.**FLD_FILENAME = matrix.fld.tsv**

kb_python.constants.**CELLS_FILENAME = matrix.cells**

kb_python.constants.**GENE_DIR = counts_gene**

kb_python.constants.**GENES_FILENAME = genes.txt**

kb_python.constants.**UNFILTERED_QUANT_DIR = quant_unfiltered**

kb_python.constants.**SAVED_INDEX_FILENAME = index.saved**

kb_python.constants.**INTERNAL_SUFFIX = _internal**

kb_python.constants.**UMI_SUFFIX = _umi**

kb_python.constants.**CAPTURE_FILENAME = capture_nonUMI.txt**

kb_python.constants.**INSPECT_INTERNAL_FILENAME = inspect_internal.json**

kb_python.constants.**INSPECT_UMI_FILENAME = inspect_umi.json**

kb_python.constants.**SORT_CODE = s**

kb_python.constants.**CORRECT_CODE = c**

kb_python.constants.**FILTERED_CODE = filtered**

kb_python.constants.**UNFILTERED_CODE = unfiltered**

kb_python.constants.**PROJECT_CODE = p**

kb_python.count

Module Contents

Functions

`kallisto_bus`(fastqs: Union[List[str], str], index_path: str, technology: str, out_dir: str, threads: int = 8, n: bool = False, k: bool = False, paired: bool = False, strand: Optional[typing_extensions.Literal[unstranded, forward, reverse]] = None) → Dict[str, str]	Runs *kallisto bus*.
`kallisto_quant_tcc`(mtx_path: str, saved_index_path: str, ecmap_path: str, t2g_path: str, out_dir: str, flens_path: Optional[str] = None, l: Optional[int] = None, s: Optional[int] = None, threads: int = 8) → Dict[str, str]	Runs *kallisto quant-tcc*.
`bustools_project`(bus_path: str, out_path: str, map_path: str, ecmap_path: str, txnames_path: str) → Dict[str, str]	Runs *bustools project*.
`bustools_sort`(bus_path: str, out_path: str, temp_dir: str = 'tmp', threads: int = 8, memory: str = '4G', flags: bool = False) → Dict[str, str]	Runs *bustools sort*.
`bustools_inspect`(bus_path: str, out_path: str, whitelist_path: Optional[str] = None, ecmap_path: Optional[str] = None) → Dict[str, str]	Runs *bustools inspect*.
`bustools_correct`(bus_path: str, out_path: str, whitelist_path: str) → Dict[str, str]	Runs *bustools correct*.
`bustools_count`(bus_path: str, out_prefix: str, t2g_path: str, ecmap_path: str, txnames_path: str, tcc: bool = False, mm: bool = False, cm: bool = False, umi_gene: bool = False, em: bool = False) → Dict[str, str]	Runs *bustools count*.
`bustools_capture`(bus_path: str, out_path: str, capture_path: str, ecmap_path: Optional[str] = None, txnames_path: Optional[str] = None, capture_type: typing_extensions.Literal[transcripts, umis, barcode] = 'transcripts', complement: bool = True) → Dict[str, str]	Runs *bustools capture*.
`bustools_whitelist`(bus_path: str, out_path: str, threshold: Optional[int] = None) → Dict[str, str]	Runs *bustools whitelist*.
`matrix_to_cellranger`(matrix_path: str, barcodes_path: str, genes_path: str, t2g_path: str, out_dir: str) → Dict[str, str]	Convert bustools count matrix to cellranger-format matrix.
`convert_matrix`(counts_dir: str, matrix_path: str, barcodes_path: str, genes_path: Optional[str] = None, ec_path: Optional[str] = None, t2g_path: Optional[str] = None, txnames_path: Optional[str] = None, name: str = 'gene', loom: bool = False, h5ad: bool = False, by_name: bool = False, tcc: bool = False, threads: int = 8) → Dict[str, str]	Convert a gene count or TCC matrix to loom or h5ad.
`convert_matrices`(counts_dir: str, matrix_paths: List[str], barcodes_paths: List[str], genes_paths: Optional[List[str]] = None, ec_paths: Optional[List[str]] = None, t2g_path: Optional[str] = None, txnames_path: Optional[str] = None, name: str = 'gene', loom: bool = False, h5ad: bool = False, by_name: bool = False, nucleus: bool = False, tcc: bool = False, threads: int = 8) → Dict[str, str]	Convert a gene count or TCC matrix to loom or h5ad.
`filter_with_bustools`(bus_path: str, ecmap_path: str, txnames_path: str, t2g_path: str, whitelist_path: str, filtered_bus_path: str, filter_threshold: Optional[int] = None, counts_prefix: Optional[str] = None, tcc: bool = False, mm: bool = False, kite: bool = False, temp_dir: str = 'tmp', threads: int = 8, memory: str = '4G',	Generate filtered count matrices with bustools.

Attributes

INSPECT_PARSER

`kb_python.count.`**`INSPECT_PARSER`**

`kb_python.count.`**`kallisto_bus`**(*fastqs: Union[List[str], str]*, *index_path: str*, *technology: str*, *out_dir: str*, *threads: int = 8*, *n: bool = False*, *k: bool = False*, *paired: bool = False*, *strand: Optional[typing_extensions.Literal[unstranded, forward, reverse]] = None*) → Dict[str, str]

Runs *kallisto bus*.

Parameters

- **`fastqs`** – List of FASTQ file paths, or a single path to a batch file
- **`index_path`** – Path to kallisto index
- **`technology`** – Single-cell technology used
- **`out_dir`** – Path to output directory
- **`threads`** – Number of threads to use, defaults to *8*
- **`n`** – Include number of read in flag column (used when splitting indices), defaults to *False*
- **`k`** – Alignment is done per k-mer (used when splitting indices), defaults to *False*
- **`paired`** – Whether or not to supply the *–paired* flag, only used for bulk and smartseq2 samples, defaults to *False*
- **`strand`** – Strandedness, defaults to *None*

Returns

Dictionary containing paths to generated files

`kb_python.count.`**`kallisto_quant_tcc`**(*mtx_path: str*, *saved_index_path: str*, *ecmap_path: str*, *t2g_path: str*, *out_dir: str*, *flens_path: Optional[str] = None*, *l: Optional[int] = None*, *s: Optional[int] = None*, *threads: int = 8*) → Dict[str, str]

Runs *kallisto quant-tcc*.

Parameters

- **`mtx_path`** – Path to counts matrix
- **`saved_index_path`** – Path to index.saved
- **`ecmap_path`** – Path to ecmap
- **`t2g_path`** – Path to T2G
- **`out_dir`** – Output directory path
- **`flens_path`** – Path to flens.txt, defaults to *None*
- **`l`** – Mean fragment length, defaults to *None*
- **`s`** – Standard deviation of fragment length, defaults to *None*
- **`threads`** – Number of threads to use, defaults to *8*

Returns

Dictionary containing path to output files `kb_python.count.`**`bustools_project`**(*bus_path: str*, *out_path: str*, *map_path: str*, *ecmap_path: str*, *txnames_path: str*) → Dict[str, str]

Runs *bustools project*.

bus_path: Path to BUS file to sort out_dir: Path to output directory map_path: Path to file containing source-to-destination mapping ecmap_path: Path to ecmap file, as generated by *kallisto bus* txnames_path: Path to transcript names file, as generated by *kallisto bus*

Returns
Dictionary containing path to generated BUS file

`kb_python.count.`**`bustools_sort`**(*bus_path: str*, *out_path: str*, *temp_dir: str = 'tmp'*, *threads: int = 8*, *memory: str = '4G'*, *flags: bool = False*) → Dict[str, str]

Runs *bustools sort*.

Parameters

- **`bus_path`** – Path to BUS file to sort
- **`out_dir`** – Path to output BUS path
- **`temp_dir`** – Path to temporary directory, defaults to *tmp*
- **`threads`** – Number of threads to use, defaults to *8*
- **`memory`** – Amount of memory to use, defaults to *4G*
- **`flags`** – Whether to supply the *–flags* argument to sort, defaults to *False*

Returns
Dictionary containing path to generated index

`kb_python.count.`**`bustools_inspect`**(*bus_path: str*, *out_path: str*, *whitelist_path: Optional[str] = None*, *ecmap_path: Optional[str] = None*) → Dict[str, str]

Runs *bustools inspect*.

Parameters

- **`bus_path`** – Path to BUS file to sort
- **`out_path`** – Path to output inspect JSON file
- **`whitelist_path`** – Path to whitelist
- **`ecmap_path`** – Path to ecmap file, as generated by *kallisto bus*

Returns
Dictionary containing path to generated index

`kb_python.count.`**`bustools_correct`**(*bus_path: str*, *out_path: str*, *whitelist_path: str*) → Dict[str, str]

Runs *bustools correct*.

Parameters

- **`bus_path`** – Path to BUS file to correct
- **`out_path`** – Path to output corrected BUS file
- **`whitelist_path`** – Path to whitelist

Returns
Dictionary containing path to generated index

`kb_python.count.`**`bustools_count`**`(`*bus_path: str*, *out_prefix: str*, *t2g_path: str*, *ecmap_path: str*, *txnames_path: str*, *tcc: bool = False*, *mm: bool = False*, *cm: bool = False*, *umi_gene: bool = False*, *em: bool = False*`)` → Dict[str, str]

Runs *bustools count*.

Parameters

- **`bus_path`** – Path to BUS file to correct
- **`out_prefix`** – Prefix of the output files to generate
- **`t2g_path`** – Path to output transcript-to-gene mapping
- **`ecmap_path`** – Path to ecmap file, as generated by *kallisto bus*
- **`txnames_path`** – Path to transcript names file, as generated by *kallisto bus*
- **`tcc`** – Whether to generate a TCC matrix instead of a gene count matrix, defaults to *False*
- **`mm`** – Whether to include BUS records that pseudoalign to multiple genes, defaults to *False*
- **`cm`** – Count multiplicities instead of UMIs. Used for chemitries without UMIs, such as bulk and Smartseq2, defaults to *False*
- **`umi_gene`** – Whether to use genes to deduplicate umis, defaults to *False*
- **`em`** – Whether to estimate gene abundances using EM algorithm, defaults to *False*

Returns
Dictionary containing path to generated index

`kb_python.count.`**`bustools_capture`**`(`*bus_path: str*, *out_path: str*, *capture_path: str*, *ecmap_path: Optional[str] = None*, *txnames_path: Optional[str] = None*, *capture_type: typing_extensions.Literal[transcripts, umis, barcode] = 'transcripts'*, *complement: bool = True*`)` → Dict[str, str]

Runs *bustools capture*.

Parameters

- **`bus_path`** – Path to BUS file to capture
- **`out_path`** – Path to BUS file to generate
- **`capture_path`** – Path transcripts-to-capture list
- **`ecmap_path`** – Path to ecmap file, as generated by *kallisto bus*
- **`txnames_path`** – Path to transcript names file, as generated by *kallisto bus*
- **`capture_type`** – The type of information in the capture list. Can be one of *transcripts*, *umis*, *barcode*.
- **`complement`** – Whether or not to complement, defaults to *True*

Returns
Dictionary containing path to generated index

`kb_python.count.`**`bustools_whitelist`**`(`*bus_path: str*, *out_path: str*, *threshold: Optional[int] = None*`)` → Dict[str, str]

Runs *bustools whitelist*.

Parameters

- **`bus_path`** – Path to BUS file generate the whitelist from
- **`out_path`** – Path to output whitelist

- `threshold` – Barcode threshold to be included in whitelist

Returns
Dictionary containing path to generated index

`kb_python.count.`**`matrix_to_cellranger`**(*matrix_path: str*, *barcodes_path: str*, *genes_path: str*, *t2g_path: str*, *out_dir: str*) → Dict[str, str]

Convert bustools count matrix to cellranger-format matrix.

Parameters

- `matrix_path` – Path to matrix
- `barcodes_path` – List of paths to barcodes.txt
- `genes_path` – Path to genes.txt
- `t2g_path` – Path to transcript-to-gene mapping
- `out_dir` – Path to output matrix

Returns
Dictionary of matrix files

`kb_python.count.`**`convert_matrix`**(*counts_dir: str*, *matrix_path: str*, *barcodes_path: str*, *genes_path: Optional[str] = None*, *ec_path: Optional[str] = None*, *t2g_path: Optional[str] = None*, *txnames_path: Optional[str] = None*, *name: str = 'gene'*, *loom: bool = False*, *h5ad: bool = False*, *by_name: bool = False*, *tcc: bool = False*, *threads: int = 8*) → Dict[str, str]

Convert a gene count or TCC matrix to loom or h5ad.

Parameters

- `counts_dir` – Path to counts directory
- `matrix_path` – Path to matrix
- `barcodes_path` – List of paths to barcodes.txt
- `genes_path` – Path to genes.txt, defaults to *None*
- `ec_path` – Path to ec.txt, defaults to *None*
- `t2g_path` – Path to transcript-to-gene mapping. If this is provided, the third column of the mapping is appended to the anndata var, defaults to *None*
- `txnames_path` – Path to transcripts.txt, defaults to *None*
- `name` – Name of the columns, defaults to "gene"
- `loom` – Whether to generate loom file, defaults to *False*
- `h5ad` – Whether to generate h5ad file, defaults to *False*
- `by_name` – Aggregate counts by name instead of ID. Only affects when *tcc=False*.
- `tcc` – Whether the matrix is a TCC matrix, defaults to *False*
- `threads` – Number of threads to use, defaults to *8*

Returns
Dictionary of generated files

kb_python.count.**convert_matrices**(*counts_dir: str*, *matrix_paths: List[str]*, *barcodes_paths: List[str]*, *genes_paths: Optional[List[str]] = None*, *ec_paths: Optional[List[str]] = None*, *t2g_path: Optional[str] = None*, *txnames_path: Optional[str] = None*, *name: str = 'gene'*, *loom: bool = False*, *h5ad: bool = False*, *by_name: bool = False*, *nucleus: bool = False*, *tcc: bool = False*, *threads: int = 8*) → Dict[str, str]

Convert a gene count or TCC matrix to loom or h5ad.

Parameters

- **counts_dir** – Path to counts directory
- **matrix_paths** – List of paths to matrices
- **barcodes_paths** – List of paths to barcodes.txt
- **genes_paths** – List of paths to genes.txt, defaults to *None*
- **ec_paths** – List of path to ec.txt, defaults to *None*
- **t2g_path** – Path to transcript-to-gene mapping. If this is provided, the third column of the mapping is appended to the anndata var, defaults to *None*
- **txnames_path** – List of paths to transcripts.txt, defaults to *None*
- **name** – Name of the columns, defaults to "gene"
- **loom** – Whether to generate loom file, defaults to *False*
- **h5ad** – Whether to generate h5ad file, defaults to *False*
- **by_name** – Aggregate counts by name instead of ID. Only affects when *tcc=False*.
- **nucleus** – Whether the matrices contain single nucleus counts, defaults to *False*
- **tcc** – Whether the matrix is a TCC matrix, defaults to *False*
- **threads** – Number of threads to use, defaults to *8*

Returns
Dictionary of generated files

kb_python.count.**filter_with_bustools**(*bus_path: str*, *ecmap_path: str*, *txnames_path: str*, *t2g_path: str*, *whitelist_path: str*, *filtered_bus_path: str*, *filter_threshold: Optional[int] = None*, *counts_prefix: Optional[str] = None*, *tcc: bool = False*, *mm: bool = False*, *kite: bool = False*, *temp_dir: str = 'tmp'*, *threads: int = 8*, *memory: str = '4G'*, *count: bool = True*, *loom: bool = False*, *h5ad: bool = False*, *by_name: bool = False*, *cellranger: bool = False*, *umi_gene: bool = False*, *em: bool = False*) → Dict[str, str]

Generate filtered count matrices with bustools.

Parameters

- **bus_path** – Path to sorted, corrected, sorted BUS file
- **ecmap_path** – Path to matrix ec file
- **txnames_path** – Path to list of transcripts
- **t2g_path** – Path to transcript-to-gene mapping
- **whitelist_path** – Path to filter whitelist to generate
- **filtered_bus_path** – Path to filtered BUS file to generate

- **filter_threshold** – Barcode filter threshold for bustools, defaults to *None*
- **counts_prefix** – Prefix of count matrix, defaults to *None*
- **tcc** – Whether to generate a TCC matrix instead of a gene count matrix, defaults to *False*
- **mm** – Whether to include BUS records that pseudoalign to multiple genes, defaults to *False*
- **kite** – Whether this is a KITE workflow
- **temp_dir** – Path to temporary directory, defaults to *tmp*
- **threads** – Number of threads to use, defaults to *8*
- **memory** – Amount of memory to use, defaults to *4G*
- **count** – Whether to run *bustools count*, defaults to *True*
- **loom** – Whether to convert the final count matrix into a loom file, defaults to *False*
- **h5ad** – Whether to convert the final count matrix into a h5ad file, defaults to *False*
- **by_name** – Aggregate counts by name instead of ID. Only affects when *tcc=False*.
- **cellranger** – Whether to convert the final count matrix into a cellranger-compatible matrix, defaults to *False*
- **umi_gene** – Whether to perform gene-level UMI collapsing, defaults to *False*
- **em** – Whether to estimate gene abundances using EM algorithm, defaults to *False*

Returns
Dictionary of generated files

kb_python.count.**stream_fastqs**(*fastqs: List[str]*, *temp_dir: str = 'tmp'*) → List[str]

Given a list of fastqs (that may be local or remote paths), stream any remote files. Internally, calls utils.

Parameters

- **fastqs** – List of (remote or local) fastq paths
- **temp_dir** – Temporary directory

Returns
All remote paths substituted with a local path

kb_python.count.**stream_batch**(*batch_path: str*, *temp_dir: str = 'tmp'*) → str

Given a path to a batch file, produce a new batch file where all the remote FASTQs are being streamed.

Parameters

- **fastqs** – List of (remote or local) fastq paths
- **temp_dir** – Temporary directory

Returns
New batch file with all remote paths substituted with a local path

kb_python.count.**copy_or_create_whitelist**(*technology: str*, *bus_path: str*, *out_dir: str*) → str

Copies a pre-packaged whitelist if it is provided. Otherwise, runs *bustools whitelist* to generate a whitelist.

Parameters

- **technology** – Single-cell technology used
- **bus_path** – Path to BUS file generate the whitelist from
- **out_dir** – Path to output directory

Returns
Path to copied or generated whitelist

`kb_python.count.`**`convert_transcripts_to_genes`**(*txnames_path: str*, *t2g_path: str*, *genes_path: str*) → str

Convert a textfile containing transcript IDs to another textfile containing gene IDs, given a transcript-to-gene mapping.

Parameters

- **`txnames_path`** – Path to transcripts.txt
- **`t2g_path`** – Path to transcript-to-genes mapping
- **`genes_path`** – Path to output genes.txt

Returns
Path to written genes.txt

`kb_python.count.`**`write_smartseq3_capture`**(*capture_path: str*) → str

Write the capture sequence for smartseq3.

Parameters
`capture_path` – Path to write the capture sequence

Returns
Path to written file

`kb_python.count.`**`count`**(*index_path: str*, *t2g_path: str*, *technology: str*, *out_dir: str*, *fastqs: List[str]*, *whitelist_path: Optional[str] = None*, *tcc: bool = False*, *mm: bool = False*, *filter: Optional[typing_extensions.Literal[bustools]] = None*, *filter_threshold: Optional[int] = None*, *kite: bool = False*, *FB: bool = False*, *temp_dir: str = 'tmp'*, *threads: int = 8*, *memory: str = '4G'*, *overwrite: bool = False*, *loom: bool = False*, *h5ad: bool = False*, *by_name: bool = False*, *cellranger: bool = False*, *inspect: bool = True*, *report: bool = False*, *fragment_l: Optional[int] = None*, *fragment_s: Optional[int] = None*, *paired: bool = False*, *strand: Optional[typing_extensions.Literal[unstranded, forward, reverse]] = None*, *umi_gene: bool = False*, *em: bool = False*) → Dict[str, Union[str, Dict[str, str]]]

Generates count matrices for single-cell RNA seq.

Parameters

- **`index_path`** – Path to kallisto index
- **`t2g_path`** – Path to transcript-to-gene mapping
- **`technology`** – Single-cell technology used
- **`out_dir`** – Path to output directory
- **`fastqs`** – List of FASTQ file paths or a single batch definition file
- **`whitelist_path`** – Path to whitelist, defaults to *None*
- **`tcc`** – Whether to generate a TCC matrix instead of a gene count matrix, defaults to *False*
- **`mm`** – Whether to include BUS records that pseudoalign to multiple genes, defaults to *False*
- **`filter`** – Filter to use to generate a filtered count matrix, defaults to *None*
- **`filter_threshold`** – Barcode filter threshold for bustools, defaults to *None*
- **`kite`** – Whether this is a KITE workflow
- **`FB`** – Whether 10x Genomics Feature Barcoding technology was used, defaults to *False*

- `temp_dir` – Path to temporary directory, defaults to *tmp*
- `threads` – Pumber of threads to use, defaults to *8*
- `memory` – Amount of memory to use, defaults to *4G*
- `overwrite` – Overwrite an existing index file, defaults to *False*
- `loom` – Whether to convert the final count matrix into a loom file, defaults to *False*
- `h5ad` – Whether to convert the final count matrix into a h5ad file, defaults to *False*
- `by_name` – Aggregate counts by name instead of ID. Only affects when *tcc=False*.
- `cellranger` – Whether to convert the final count matrix into a cellranger-compatible matrix, defaults to *False*
- `inspect` – Whether or not to inspect the output BUS file and generate the inspect.json
- `report` – Generate an HTMl report, defaults to *False*
- `fragment_l` – Mean length of fragments, defaults to *None*
- `fragment_s` – Standard deviation of fragment lengths, defaults to *None*
- `paired` – Whether the fastqs are paired. Has no effect when a single batch file is provided. Defaults to *False*
- `strand` – Strandedness, defaults to *None*
- `umi_gene` – Whether to perform gene-level UMI collapsing, defaults to *False*
- `em` – Whether to estimate gene abundances using EM algorithm, defaults to *False*

Returns
Dictionary containing paths to generated files

`kb_python.count.`**`count_smartseq3`**(*index_path: str*, *t2g_path: str*, *out_dir: str*, *fastqs: List[str]*, *whitelist_path: Optional[str] = None*, *tcc: bool = False*, *mm: bool = False*, *temp_dir: str = 'tmp'*, *threads: int = 8*, *memory: str = '4G'*, *overwrite: bool = False*, *loom: bool = False*, *h5ad: bool = False*, *by_name: bool = False*, *inspect: bool = True*, *strand: Optional[typing_extensions.Literal[unstranded, forward, reverse]] = None*) → Dict[str, Union[str, Dict[str, str]]]

Generates count matrices for Smartseq3.

Parameters

- `index_path` – Path to kallisto index
- `t2g_path` – Path to transcript-to-gene mapping
- `out_dir` – Path to output directory
- `fastqs` – List of FASTQ file paths
- `whitelist_path` – Path to whitelist, defaults to *None*
- `tcc` – Whether to generate a TCC matrix instead of a gene count matrix, defaults to *False*
- `mm` – Whether to include BUS records that pseudoalign to multiple genes, defaults to *False*
- `temp_dir` – Path to temporary directory, defaults to *tmp*
- `threads` – Pumber of threads to use, defaults to *8*
- `memory` – Amount of memory to use, defaults to *4G*
- `overwrite` – Overwrite an existing index file, defaults to *False*

- `loom` – Whether to convert the final count matrix into a loom file, defaults to *False*
- `h5ad` – Whether to convert the final count matrix into a h5ad file, defaults to *False*
- `by_name` – Aggregate counts by name instead of ID. Only affects when *tcc=False*.
- `inspect` – Whether or not to inspect the output BUS file and generate the inspect.json
- `strand` – Strandedness, defaults to *None*

Returns
Dictionary containing paths to generated files

`kb_python.count.count_velocity`(*index_path: str*, *t2g_path: str*, *cdna_t2c_path: str*, *intron_t2c_path: str*, *technology: str*, *out_dir: str*, *fastqs: List[str]*, *whitelist_path: Optional[str] = None*, *tcc: bool = False*, *mm: bool = False*, *filter: Optional[typing_extensions.Literal[bustools]] = None*, *filter_threshold: Optional[int] = None*, *temp_dir: str = 'tmp'*, *threads: int = 8*, *memory: str = '4G'*, *overwrite: bool = False*, *loom: bool = False*, *h5ad: bool = False*, *by_name: bool = False*, *cellranger: bool = False*, *inspect: bool = True*, *report: bool = False*, *nucleus: bool = False*, *fragment_l: Optional[int] = None*, *fragment_s: Optional[int] = None*, *paired: bool = False*, *strand: Optional[typing_extensions.Literal[unstranded, forward, reverse]] = None*, *umi_gene: bool = False*, *em: bool = False*) → Dict[str, Union[Dict[str, str], str]]

Generates RNA velocity matrices for single-cell RNA seq.

Parameters

- `index_path` – Path to kallisto index
- `t2g_path` – Path to transcript-to-gene mapping
- `cdna_t2c_path` – Path to cDNA transcripts-to-capture file
- `intron_t2c_path` – Path to intron transcripts-to-capture file
- `technology` – Single-cell technology used
- `out_dir` – Path to output directory
- `fastqs` – List of FASTQ file paths or a single batch definition file
- `whitelist_path` – Path to whitelist, defaults to *None*
- `tcc` – Whether to generate a TCC matrix instead of a gene count matrix, defaults to *False*
- `mm` – Whether to include BUS records that pseudoalign to multiple genes, defaults to *False*
- `filter` – Filter to use to generate a filtered count matrix, defaults to *None*
- `filter_threshold` – Barcode filter threshold for bustools, defaults to *None*
- `temp_dir` – Path to temporary directory, defaults to *tmp*
- `threads` – Number of threads to use, defaults to *8*
- `memory` – Amount of memory to use, defaults to *4G*
- `overwrite` – Overwrite an existing index file, defaults to *False*
- `loom` – Whether to convert the final count matrix into a loom file, defaults to *False*
- `h5ad` – Whether to convert the final count matrix into a h5ad file, defaults to *False*
- `by_name` – Aggregate counts by name instead of ID. Only affects when *tcc=False*.

- `cellranger` – Whether to convert the final count matrix into a cellranger-compatible matrix, defaults to *False*
- `inspect` – Whether or not to inspect the output BUS file and generate the inspect.json
- `report` – Generate HTML reports, defaults to *False*
- `nucleus` – Whether this is a single-nucleus experiment. if *True*, the spliced and unspliced count matrices will be summed, defaults to *False*
- `fragment_l` – Mean length of fragments, defaults to *None*
- `fragment_s` – Standard deviation of fragment lengths, defaults to *None*
- `paired` – Whether the fastqs are paired. Has no effect when a single batch file is provided. Defaults to *False*
- `strand` – Strandedness, defaults to *None*
- `umi_gene` – Whether to perform gene-level UMI collapsing, defaults to *False*
- `em` – Whether to estimate gene abundances using EM algorithm, defaults to *False*

Returns

Dictionary containing path to generated index

`kb_python.count.`**`count_velocity_smartseq3`**`(`*index_path: str*, *t2g_path: str*, *cdna_t2c_path: str*, *intron_t2c_path: str*, *out_dir: str*, *fastqs: List[str]*, *whitelist_path: Optional[str] = None*, *tcc: bool = False*, *mm: bool = False*, *temp_dir: str = 'tmp'*, *threads: int = 8*, *memory: str = '4G'*, *overwrite: bool = False*, *loom: bool = False*, *h5ad: bool = False*, *by_name: bool = False*, *inspect: bool = True*, *strand: Optional[typing_extensions.Literal[unstranded, forward, reverse]] = None*`)` → Dict[str, Union[str, Dict[str, str]]]

Generates count matrices for Smartseq3.

Parameters

- `index_path` – Path to kallisto index
- `t2g_path` – Path to transcript-to-gene mapping
- `out_dir` – Path to output directory
- `fastqs` – List of FASTQ file paths
- `whitelist_path` – Path to whitelist, defaults to *None*
- `tcc` – Whether to generate a TCC matrix instead of a gene count matrix, defaults to *False*
- `mm` – Whether to include BUS records that pseudoalign to multiple genes, defaults to *False*
- `temp_dir` – Path to temporary directory, defaults to *tmp*
- `threads` – Pumber of threads to use, defaults to *8*
- `memory` – Amount of memory to use, defaults to *4G*
- `overwrite` – Overwrite an existing index file, defaults to *False*
- `loom` – Whether to convert the final count matrix into a loom file, defaults to *False*
- `h5ad` – Whether to convert the final count matrix into a h5ad file, defaults to *False*
- `by_name` – Aggregate counts by name instead of ID. Only affects when *tcc=False*.

- **inspect** – Whether or not to inspect the output BUS file and generate the inspect.json
- **strand** – Strandedness, defaults to *None*

Returns
Dictionary containing paths to generated files

kb_python.logging

Module Contents

kb_python.logging.**logger**

kb_python.main

Module Contents

Functions

test_binaries() → Tuple[bool, bool]	Test whether kallisto and bustools binaries are executable.
get_binary_info() → str	Get information on the binaries that will be used for commands.
display_info()	Displays kb, kallisto and bustools version + citation information, along
display_technologies()	Displays a list of supported technologies along with whether kb provides
parse_compile(parser: argparse.ArgumentParser, args: argparse.Namespace, temp_dir: str = 'tmp')	Parser for the *compile* command.
parse_ref(parser: argparse.ArgumentParser, args: argparse.Namespace, temp_dir: str = 'tmp')	Parser for the *ref* command.
parse_count(parser: argparse.ArgumentParser, args: argparse.Namespace, temp_dir: str = 'tmp')	Parser for the *count* command.
setup_info_args(parser: argparse.ArgumentParser, parent: argparse.ArgumentParser) → argparse.ArgumentParser	Helper function to set up a subparser for the *info* command.
setup_compile_args(parser: argparse.ArgumentParser, parent: argparse.ArgumentParser) → argparse.ArgumentParser	Helper function to set up a subparser for the *compile* command.
setup_ref_args(parser: argparse.ArgumentParser, parent: argparse.ArgumentParser) → argparse.ArgumentParser	Helper function to set up a subparser for the *ref* command.
setup_count_args(parser: argparse.ArgumentParser, parent: argparse.ArgumentParser) → argparse.ArgumentParser	Helper function to set up a subparser for the *count* command.
main()	Command-line entrypoint.

Attributes

COMMAND_TO_FUNCTION

`kb_python.main.`**`test_binaries`**`()` → Tuple[bool, bool]

Test whether kallisto and bustools binaries are executable.

Internally, this function calls `utils.get_kallisto_version()` and `utils.get_bustools_version()`, both of which return *None* if there is something wrong with their respective binaries.

Returns
A tuple of two booleans indicating kallisto and bustools binaries.

`kb_python.main.`**`get_binary_info`**`()` → str

Get information on the binaries that will be used for commands.

Returns
kallisto and *bustools* binary versions and paths.

`kb_python.main.`**`display_info`**`()`

Displays kb, kallisto and bustools version + citation information, along with a brief description and examples.

`kb_python.main.`**`display_technologies`**`()`

Displays a list of supported technologies along with whether kb provides a whitelist for that technology and the FASTQ argument order for kb count.

`kb_python.main.`**`parse_compile`**`(`*parser: argparse.ArgumentParser*, *args: argparse.Namespace*, *temp_dir: str = 'tmp'*`)`

Parser for the *compile* command.

Parameters

- **`parser`** – The argument parser
- **`args`** – Parsed command-line arguments

`kb_python.main.`**`parse_ref`**`(`*parser: argparse.ArgumentParser*, *args: argparse.Namespace*, *temp_dir: str = 'tmp'*`)`

Parser for the *ref* command.

Parameters

- **`parser`** – The argument parser
- **`args`** – Parsed command-line arguments

`kb_python.main.`**`parse_count`**`(`*parser: argparse.ArgumentParser*, *args: argparse.Namespace*, *temp_dir: str = 'tmp'*`)`

Parser for the *count* command.

Parameters

- **`parser`** – The argument parser
- **`args`** – Parsed command-line arguments

`kb_python.main.`**`COMMAND_TO_FUNCTION`** `kb_python.main.`**`setup_info_args`**(*parser: argparse.ArgumentParser*, *parent: argparse.ArgumentParser*) → argparse.ArgumentParser

Helper function to set up a subparser for the *info* command.

Parameters

- **`parser`** – Parser to add the *info* command to
- **`parent`** – Parser parent of the newly added subcommand. used to inherit shared commands/flags

Returns
The newly added parser

`kb_python.main.`**`setup_compile_args`**(*parser: argparse.ArgumentParser*, *parent: argparse.ArgumentParser*) → argparse.ArgumentParser

Helper function to set up a subparser for the *compile* command.

Parameters

- **`parser`** – Parser to add the *compile* command to
- **`parent`** – Parser parent of the newly added subcommand. used to inherit shared commands/flags

Returns
The newly added parser

`kb_python.main.`**`setup_ref_args`**(*parser: argparse.ArgumentParser*, *parent: argparse.ArgumentParser*) → argparse.ArgumentParser

Helper function to set up a subparser for the *ref* command.

Parameters

- **`parser`** – Parser to add the *ref* command to
- **`parent`** – Parser parent of the newly added subcommand. used to inherit shared commands/flags

Returns
The newly added parser

`kb_python.main.`**`setup_count_args`**(*parser: argparse.ArgumentParser*, *parent: argparse.ArgumentParser*) → argparse.ArgumentParser

Helper function to set up a subparser for the *count* command.

Parameters

- **`parser`** – Parser to add the *count* command to
- **`parent`** – Parser parent of the newly added subcommand. used to inherit shared commands/flags

Returns
The newly added parser

`kb_python.main.`**`main`**()

Command-line entrypoint.

kb_python.ref

Module Contents

Functions

generate_kite_fasta(feature_path: str, out_path: str, no_mismatches: bool = False) → Tuple[str, int]	Generate a FASTA file for feature barcoding with the KITE workflow.
create_t2g_from_fasta(fasta_path: str, t2g_path: str) → Dict[str, str]	Parse FASTA headers to get transcripts-to-gene mapping.
create_t2c(fasta_path: str, t2c_path: str) → Dict[str, str]	Creates a transcripts-to-capture list from a FASTA file.
kallisto_index(fasta_path: str, index_path: str, k: int = 31) → Dict[str, str]	Runs *kallisto index*.
split_and_index(fasta_path: str, index_prefix: str, n: int = 2, k: int = 31, temp_dir: str = 'tmp') → Dict[str, str]	Split a FASTA file into *n* parts and index each one.
download_reference(reference: kb_python.config.Reference, files: Dict[str, str], temp_dir: str = 'tmp', overwrite: bool = False) → Dict[str, str]	Downloads a provided reference file from a static url.
decompress_file(path: str, temp_dir: str = 'tmp') → str	Decompress the given path if it is a .gz file. Otherwise, return the
get_gtf_attribute_include_func(include: List[Dict[str, str]]) → Callable[[ngs_tools.gtf.GtfEntry], bool]	Helper function to create a filtering function to include certain GTF
get_gtf_attribute_exclude_func(exclude: List[Dict[str, str]]) → Callable[[ngs_tools.gtf.GtfEntry], bool]	Helper function to create a filtering function to exclude certain GTF
ref(fasta_paths: Union[List[str], str], gtf_paths: Union[List[str], str], cdna_path: str, index_path: str, t2g_path: str, n: int = 1, k: Optional[int] = None, include: Optional[List[Dict[str, str]]] = None, exclude: Optional[List[Dict[str, str]]] = None, temp_dir: str = 'tmp', overwrite: bool = False) → Dict[str, str]	Generates files necessary to generate count matrices for single-cell RNA-seq.
ref_kite(feature_path: str, fasta_path: str, index_path: str, t2g_path: str, n: int = 1, k: Optional[int] = None, no_mismatches: bool = False, temp_dir: str = 'tmp', overwrite: bool = False) → Dict[str, str]	Generates files necessary for feature barcoding with the KITE workflow.
ref_lamanno(fasta_paths: Union[List[str], str], gtf_paths: Union[List[str], str], cdna_path: str, intron_path: str, index_path: str, t2g_path: str, cdna_t2c_path: str, intron_t2c_path: str, n: int = 1, k: Optional[int] = None, flank: Optional[int] = None, include: Optional[List[Dict[str, str]]] = None, exclude: Optional[List[Dict[str, str]]] = None, temp_dir: str = 'tmp', overwrite: bool = False) → Dict[str, str]	Generates files necessary to generate RNA velocity matrices for single-cell RNA-seq.

exception `kb_python.ref.`**`RefError`**

Bases: `Exception`

Common base class for all non-exit exceptions.

`kb_python.ref.`**`generate_kite_fasta`**(*feature_path: str*, *out_path: str*, *no_mismatches: bool = False*) → Tuple[str, int]

Generate a FASTA file for feature barcoding with the KITE workflow.

This FASTA contains all sequences that are 1 hamming distance from the provided barcodes. The file of barcodes must be a 2-column TSV containing the barcode sequences in the first column and their corresponding feature name in the second column. If hamming distance 1 variants collide for any pair of barcodes, the hamming distance 1 variants for those barcodes are not generated.

Parameters

- **`feature_path`** – Path to TSV containing barcodes and feature names
- **`out_path`** – Path to FASTA to generate
- **`no_mismatches`** – Whether to generate hamming distance 1 variants, defaults to *False*

Returns
Path to generated FASTA, smallest barcode length

Raises
`RefError` – If there are barcodes of different lengths or if there are duplicate barcodes

`kb_python.ref.`**`create_t2g_from_fasta`**(*fasta_path: str*, *t2g_path: str*) → Dict[str, str]

Parse FASTA headers to get transcripts-to-gene mapping.

Parameters

- **`fasta_path`** – Path to FASTA file
- **`t2g_path`** – Path to output transcript-to-gene mapping

Returns
Dictionary containing path to generated t2g mapping

`kb_python.ref.`**`create_t2c`**(*fasta_path: str*, *t2c_path: str*) → Dict[str, str]

Creates a transcripts-to-capture list from a FASTA file.

Parameters

- **`fasta_path`** – Path to FASTA file
- **`t2c_path`** – Path to output transcripts-to-capture list

Returns
Dictionary containing path to generated t2c list

`kb_python.ref.`**`kallisto_index`**(*fasta_path: str*, *index_path: str*, *k: int = 31*) → Dict[str, str]

Runs *kallisto index*.

Parameters

- **`fasta_path`** – path to FASTA file
- **`index_path`** – path to output kallisto index
- **`k`** – k-mer length, defaults to 31

Returns
Dictionary containing path to generated index

`kb_python.ref.`**`split_and_index`**(*fasta_path: str*, *index_prefix: str*, *n: int = 2*, *k: int = 31*, *temp_dir: str = 'tmp'*) → Dict[str, str]

Split a FASTA file into *n* parts and index each one.

Parameters

- `fasta_path` – Path to FASTA file
- `index_prefix` – Prefix of output kallisto indices
- `n` – Split the index into *n* files, defaults to *2*
- `k` – K-mer length, defaults to 31
- `temp_dir` – Path to temporary directory, defaults to *tmp*

Returns
Dictionary containing path to generated index

`kb_python.ref.`**`download_reference`**(*reference: kb_python.config.Reference*, *files: Dict[str, str]*, *temp_dir: str = 'tmp'*, *overwrite: bool = False*) → Dict[str, str]

Downloads a provided reference file from a static url.

The configuration for provided references is in *config.py*.

Parameters

- `reference` – A Reference object
- `files` – Dictionary that has the command-line option as keys and the path as values. used to determine if all the required paths to download the given reference have been provided
- `temp_dir` – Path to temporary directory, defaults to *tmp*
- `overwrite` – Overwrite an existing index file, defaults to *False*

Returns
Dictionary containing paths to generated file(s)

Raises
`RefError` – If the required options are not provided

`kb_python.ref.`**`decompress_file`**(*path: str*, *temp_dir: str = 'tmp'*) → str

Decompress the given path if it is a .gz file. Otherwise, return the original path.

Parameters
`path` – Path to the file

Returns

Unaltered *path* if the file is not a .gz file, otherwise path to the
uncompressed file

`kb_python.ref.`**`get_gtf_attribute_include_func`**(*include: List[Dict[str, str]]*) → Callable[[ngs_tools.gtf.GtfEntry], bool]

Helper function to create a filtering function to include certain GTF entries while processing. The returned function returns *True* if the entry should be included.

Parameters
`include` – List of dictionaries representing key-value pairs of attributes to include

Returns
Filter function

`kb_python.ref.`**`get_gtf_attribute_exclude_func`**(*exclude: List[Dict[str, str]]*) → Callable[[ngs_tools.gtf.GtfEntry], bool]

Helper function to create a filtering function to exclude certain GTF entries while processing. The returned function returns *False* if the entry should be excluded.

Parameters
`exclude` – List of dictionaries representing key-value pairs of attributes to exclude

Returns
Filter function

`kb_python.ref.`**`ref`**(*fasta_paths: Union[List[str], str]*, *gtf_paths: Union[List[str], str]*, *cdna_path: str*, *index_path: str*, *t2g_path: str*, *n: int = 1*, *k: Optional[int] = None*, *include: Optional[List[Dict[str, str]]] = None*, *exclude: Optional[List[Dict[str, str]]] = None*, *temp_dir: str = 'tmp'*, *overwrite: bool = False*) → Dict[str, str]

Generates files necessary to generate count matrices for single-cell RNA-seq.

Parameters

- **`fasta_paths`** – List of paths to genomic FASTA files
- **`gtf_paths`** – List of paths to GTF files
- **`cdna_path`** – Path to generate the cDNA FASTA file
- **`t2g_path`** – Path to output transcript-to-gene mapping
- **`n`** – Split the index into *n* files
- **`k`** – Override default kmer length 31, defaults to *None*
- **`include`** – List of dictionaries representing key-value pairs of attributes to include
- **`exclude`** – List of dictionaries representing key-value pairs of attributes to exclude
- **`temp_dir`** – Path to temporary directory, defaults to *tmp*
- **`overwrite`** – Overwrite an existing index file, defaults to *False*

Returns
Dictionary containing paths to generated file(s)

`kb_python.ref.`**`ref_kite`**(*feature_path: str*, *fasta_path: str*, *index_path: str*, *t2g_path: str*, *n: int = 1*, *k: Optional[int] = None*, *no_mismatches: bool = False*, *temp_dir: str = 'tmp'*, *overwrite: bool = False*) → Dict[str, str]

Generates files necessary for feature barcoding with the KITE workflow.

Parameters

- **`feature_path`** – Path to TSV containing barcodes and feature names
- **`fasta_path`** – Path to generate fasta file containing all sequences that are 1 hamming distance from the provide barcodes (including the actual sequence)
- **`t2g_path`** – Path to output transcript-to-gene mapping
- **`n`** – Split the index into *n* files
- **`k`** – Override calculated optimal kmer length, defaults to *None*
- **`no_mismatches`** – Whether to generate hamming distance 1 variants, defaults to *False*
- **`temp_dir`** – Path to temporary directory, defaults to *tmp*
- **`overwrite`** – Overwrite an existing index file, defaults to *False*

Returns
Dictionary containing paths to generated file(s)

`kb_python.ref.`**`ref_lamanno`**(*fasta_paths: Union[List[str], str]*, *gtf_paths: Union[List[str], str]*, *cdna_path: str*, *intron_path: str*, *index_path: str*, *t2g_path: str*, *cdna_t2c_path: str*, *intron_t2c_path: str*, *n: int = 1*, *k: Optional[int] = None*, *flank: Optional[int] = None*, *include: Optional[List[Dict[str, str]]] = None*, *exclude: Optional[List[Dict[str, str]]] = None*, *temp_dir: str = 'tmp'*, *overwrite: bool = False*) → Dict[str, str]

Generates files necessary to generate RNA velocity matrices for single-cell RNA-seq.

Parameters

- **`fasta_paths`** – List of paths to genomic FASTA files
- **`gtf_paths`** – List of paths to GTF files
- **`cdna_path`** – Path to generate the cDNA FASTA file
- **`intron_path`** – Path to generate the intron FASTA file
- **`t2g_path`** – Path to output transcript-to-gene mapping
- **`cdna_t2c_path`** – Path to generate the cDNA transcripts-to-capture file
- **`intron_t2c_path`** – Path to generate the intron transcripts-to-capture file
- **`n`** – Split the index into *n* files
- **`k`** – Override default kmer length (31), defaults to *None*
- **`flank`** – Number of bases to include from the flanking regions when generating the intron FASTA, defaults to *None*, which sets the flanking region to be k - 1 bases.
- **`include`** – List of dictionaries representing key-value pairs of attributes to include
- **`exclude`** – List of dictionaries representing key-value pairs of attributes to exclude
- **`temp_dir`** – Path to temporary directory, defaults to *tmp*
- **`overwrite`** – Overwrite an existing index file, defaults to *False*

Returns

Dictionary containing paths to generated file(s)

`kb_python.report`

Module Contents

Functions

dict_to_table(d: Dict[str, Any], column_ratio: List[int] = [3, 7], column_align: List[str] = ['right', 'left']) → plotly.graph_objects.Figure	Convert a dictionary to a Plot.ly table of key-value pairs.
knee_plot(n_counts: List[int]) → plotly.graph_objects.Figure	Generate knee plot card.
genes_detected_plot(n_counts: List[int], n_genes: List[int]) → plotly.graph_objects.Figure	Generate genes detected plot card.
elbow_plot(pca_variance_ratio: List[float]) → plotly.graph_objects.Figure	Generate elbow plot card.
pca_plot(pc: numpy.ndarray) → plotly.graph_objects.Figure	Generate PCA plot card.
write_report(stats_path: str, info_path: str, inspect_path: str, out_path: str, matrix_path: Optional[str] = None, barcodes_path: Optional[str] = None, genes_path: Optional[str] = None, t2g_path: Optional[str] = None) → str	Render the Jupyter notebook report with Jinja2.
execute_report(execute_path: str, nb_path: str, html_path: str) → Tuple[str, str]	Execute the report and write the results as a Jupyter notebook and HTML.
render_report(stats_path: str, info_path: str, inspect_path: str, nb_path: str, html_path: str, matrix_path: Optional[str] = None, barcodes_path: Optional[str] = None, genes_path: Optional[str] = None, t2g_path: Optional[str] = None, temp_dir: str = 'tmp') → Dict[str, str]	Render and execute the report.

Attributes

REPORT_DIR
BASIC_TEMPLATE_PATH
MATRIX_TEMPLATE_PATH
MARGIN

kb_python.report.**REPORT_DIR**

kb_python.report.**BASIC_TEMPLATE_PATH**

kb_python.report.**MATRIX_TEMPLATE_PATH**

kb_python.report.**MARGIN**

kb_python.report.**dict_to_table**(*d: Dict[str, Any]*, *column_ratio: List[int] = [3, 7]*, *column_align: List[str] = ['right', 'left']*) → plotly.graph_objects.Figure

Convert a dictionary to a Plot.ly table of key-value pairs.

Parameters

- `d` – Dictionary to convert
- `column_ratio` – Relative column widths, represented as a ratio, defaults to *[3, 7]*
- `column_align` – Column text alignments, defaults to *['right', 'left']*

Returns
Figure

`kb_python.report.`**`knee_plot`**`(`*n_counts: List[int]*`)` → plotly.graph_objects.Figure

Generate knee plot card.

Parameters
`n_counts` – List of UMI counts

Returns
Figure

`kb_python.report.`**`genes_detected_plot`**`(`*n_counts: List[int]*, *n_genes: List[int]*`)` → plotly.graph_objects.Figure

Generate genes detected plot card.

Parameters

- `n_counts` – List of UMI counts
- `n_genes` – List of gene counts

Returns
Figure

`kb_python.report.`**`elbow_plot`**`(`*pca_variance_ratio: List[float]*`)` → plotly.graph_objects.Figure

Generate elbow plot card.

Parameters
`pca_variance_ratio` – List PCA variance ratios

Returns
Figure

`kb_python.report.`**`pca_plot`**`(`*pc: numpy.ndarray*`)` → plotly.graph_objects.Figure

Generate PCA plot card.

Parameters
`pc` – Embeddings

Returns
Figure

`kb_python.report.`**`write_report`**`(`*stats_path: str*, *info_path: str*, *inspect_path: str*, *out_path: str*, *matrix_path: Optional[str] = None*, *barcodes_path: Optional[str] = None*, *genes_path: Optional[str] = None*, *t2g_path: Optional[str] = None*`)` → str

Render the Jupyter notebook report with Jinja2.

Parameters

- `stats_path` – Path to kb stats JSON
- `info_path` – Path to run_info.json
- `inspect_path` – Path to inspect.json
- `out_path` – Path to Jupyter notebook to generate
- `matrix_path` – Path to matrix

- **barcodes_path** – List of paths to barcodes.txt
- **genes_path** – Path to genes.txt, defaults to *None*
- **t2g_path** – Path to transcript-to-gene mapping

Returns
Path to notebook generated

kb_python.report.**execute_report**(*execute_path: str*, *nb_path: str*, *html_path: str*) → Tuple[str, str]

Execute the report and write the results as a Jupyter notebook and HTML.

Parameters

- **execute_path** – Path to Jupyter notebook to execute
- **nb_path** – Path to Jupyter notebook to generate
- **html_path** – Path to HTML to generate

Returns
Tuple containing executed notebook and HTML

kb_python.report.**render_report**(*stats_path: str*, *info_path: str*, *inspect_path: str*, *nb_path: str*, *html_path: str*, *matrix_path: Optional[str] = None*, *barcodes_path: Optional[str] = None*, *genes_path: Optional[str] = None*, *t2g_path: Optional[str] = None*, *temp_dir: str = 'tmp'*) → Dict[str, str]

Render and execute the report.

Parameters

- **stats_path** – Path to kb stats JSON
- **info_path** – Path to run_info.json
- **inspect_path** – Path to inspect.json
- **nb_path** – Path to Jupyter notebook to generate
- **html_path** – Path to HTML to generate
- **matrix_path** – Path to matrix
- **barcodes_path** – List of paths to barcodes.txt
- **genes_path** – Path to genes.txt, defaults to *None*
- **t2g_path** – Path to transcript-to-gene mapping
- **temp_dir** – Path to temporary directory, defaults to *tmp*

Returns
Dictionary containing notebook and HTML paths

kb_python.stats

Module Contents

Classes

Stats	Class used to collect kb run statistics.

Attributes

STATS

class kb_python.stats.**Stats**

Class used to collect kb run statistics.

start(*self*)

Start collecting statistics.

Sets start time, the command line call, and the commands array to an empty list. Additionally, sets the kallisto and bustools paths and versions.

command(*self*, *command: List[str]*, *runtime: Optional[float] = None*)

Report a shell command was run.

Parameters

- **command** – A shell command, represented as a list
- **runtime** – Command runtime

end(*self*)

End collecting statistics.

save(*self*, *path: str*) → str

Save statistics as JSON to path.

Parameters
path – Path to JSON

Returns
Path to saved JSON

to_dict(*self*) → Dict[str, Union[str, float]]

Convert statistics to dictionary, so that it is easily parsed by the report-rendering functions.

Returns
Statistics dictionary

kb_python.stats.**STATS**

`kb_python.utils`

Module Contents

Functions

update_filename(filename: str, code: str) → str	Update the provided path with the specified code.
make_directory(path: str)	Quietly make the specified directory (and any subdirectories).
remove_directory(path: str)	Quietly make the specified directory (and any subdirectories).
run_executable(command: List[str], stdin: Optional[int] = None, stdout: int = sp.PIPE, stderr: int = sp.PIPE, wait: bool = True, stream: bool = True, quiet: bool = False, returncode: int = 0, alias: bool = True, record: bool = True) → Union[Tuple[subprocess.Popen, str, str], subprocess.Popen]	Execute a single shell command.
get_kallisto_version() → Optional[Tuple[int, int, int]]	Get the provided Kallisto version.
get_bustools_version() → Optional[Tuple[int, int, int]]	Get the provided Bustools version.
parse_technologies(lines: List[str]) → Set[str]	Parse a list of strings into a list of supported technologies.
get_supported_technologies() → Set[str]	Runs 'kallisto bus --list' to fetch a list of supported technologies.
whitelist_provided(technology: str) → bool	Determine whether or not the whitelist for a technology is provided.
move_file(source: str, destination: str) → str	Move a file from source to destination, overwriting the file if the
copy_whitelist(technology: str, out_dir: str) → str	Copies provided whitelist for specified technology.
create_10x_feature_barcode_map(out_path: str) → str	Create a feature-barcode map for the 10x Feature Barcoding technology.
stream_file(url: str, path: str) → str	Creates a FIFO file to use for piping remote files into processes.
read_t2g(t2g_path: str) → Dict[str, Tuple[str, Ellipsis]]	Given a transcript-to-gene mapping path, read it into a dictionary.
collapse_anndata(adata: anndata.AnnData, by: Optional[str] = None) → anndata.AnnData	Collapse the given Anndata by summing duplicate rows. The *by* argument
import_tcc_matrix_as_anndata(matrix_path: str, barcodes_path: str, ec_path: str, txnames_path: str, threads: int = 8) → anndata.AnnData	Import a TCC matrix as an Anndata object.
import_matrix_as_anndata(matrix_path: str, barcodes_path: str, genes_path: str, t2g_path: Optional[str] = None, name: str = 'gene', by_name: bool = False) → anndata.AnnData	Import a matrix as an Anndata object.
overlay_anndatas(adata_spliced: anndata.AnnData, adata_unspliced: anndata.AnnData) → anndata.AnnData	'Overlays' anndata objects by taking the intersection of the obs and var
sum_anndatas(adata_spliced: anndata.AnnData, adata_unspliced: anndata.AnnData) → anndata.AnnData	Sum the counts in two anndata objects by taking the intersection of
restore_cwd(func: Callable) → Callable	Function decorator to decorate functions that change the current working

Attributes

TECHNOLOGY_PARSER
VERSION_PARSER
open_as_text
decompress_gzip
compress_gzip
concatenate_files
download_file
get_temporary_filename

kb_python.utils.**TECHNOLOGY_PARSER**

kb_python.utils.**VERSION_PARSER**

kb_python.utils.**open_as_text**

kb_python.utils.**decompress_gzip**

kb_python.utils.**compress_gzip**

kb_python.utils.**concatenate_files**

kb_python.utils.**download_file**

kb_python.utils.**get_temporary_filename**

kb_python.utils.**update_filename**(*filename: str*, *code: str*) → str

Update the provided path with the specified code.

For instance, if the *path* is 'output.bus' and *code* is *s* (for sort), this function returns *output.s.bus*.

Parameters

- **filename** – filename (NOT path)
- **code** – code to append to filename

Returns

Path updated with provided code

kb_python.utils.**make_directory**(*path: str*)

Quietly make the specified directory (and any subdirectories).

This function is a wrapper around os.makedirs. It is used so that the appropriate mkdir command can be printed for dry runs.

Parameters

path – Path to directory to make

`kb_python.utils.`**`remove_directory`**`(`*path: str*`)`

Quietly make the specified directory (and any subdirectories).

This function is a wrapper around shutil.rmtree. It is used so that the appropriate rm command can be printed for dry runs.

Parameters
`path` – Path to directory to remove

`kb_python.utils.`**`run_executable`**`(`*command: List[str]*, *stdin: Optional[int] = None*, *stdout: int = sp.PIPE*, *stderr: int = sp.PIPE*, *wait: bool = True*, *stream: bool = True*, *quiet: bool = False*, *returncode: int = 0*, *alias: bool = True*, *record: bool = True*`)` → Union[Tuple[subprocess.Popen, str, str], subprocess.Popen]

Execute a single shell command.

Parameters

- **`command`** – A list representing a single shell command
- **`stdin`** – Object to pass into the *stdin* argument for *subprocess.Popen*, defaults to *None*
- **`stdout`** – Object to pass into the *stdout* argument for *subprocess.Popen*, defaults to *subprocess.PIPE*
- **`stderr`** – Object to pass into the *stderr* argument for *subprocess.Popen*, defaults to *subprocess.PIPE*
- **`wait`** – Whether to wait until the command has finished, defaults to *True*
- **`stream`** – Whether to stream the output to the command line, defaults to *True*
- **`quiet`** – Whether to not display anything to the command line and not check the return code, defaults to *False*
- **`returncode`** – The return code expected if the command runs as intended, defaults to *0*
- **`alias`** – Whether to use the basename of the first element of *command*, defaults to *True*
- **`record`** – Whether to record the call statistics, defaults to *True*

Returns

(the spawned process, list of strings printed to stdout,
list of strings printed to stderr) if *wait=True*. Otherwise, the spawned process

`kb_python.utils.`**`get_kallisto_version`**`()` → Optional[Tuple[int, int, int]]

Get the provided Kallisto version.

This function parses the help text by executing the included Kallisto binary.

Returns
Major, minor, patch versions

`kb_python.utils.`**`get_bustools_version`**`()` → Optional[Tuple[int, int, int]]

Get the provided Bustools version.

This function parses the help text by executing the included Bustools binary.

Returns
Major, minor, patch versions

`kb_python.utils.`**`parse_technologies`**`(`*lines: List[str]*`)` → Set[str]

Parse a list of strings into a list of supported technologies.

This function parses the technologies printed by running *kallisto bus –list*.

Parameters
`lines` – The output of *kallisto bus –list* split into lines

Returns
Set of technologies

`kb_python.utils.`**`get_supported_technologies`**`()` → Set[str]

Runs 'kallisto bus –list' to fetch a list of supported technologies.

Returns
Set of technologies

`kb_python.utils.`**`whitelist_provided`**`(`*technology: str*`)` → bool

Determine whether or not the whitelist for a technology is provided.

Parameters
`technology` – The name of the technology

Returns
Whether the whitelist is provided

`kb_python.utils.`**`move_file`**`(`*source: str*, *destination: str*`)` → str

Move a file from source to destination, overwriting the file if the destination exists.

Parameters

- `source` – Path to source file
- `destination` – Path to destination

Returns
Path to moved file

`kb_python.utils.`**`copy_whitelist`**`(`*technology: str*, *out_dir: str*`)` → str

Copies provided whitelist for specified technology.

Parameters

- `technology` – The name of the technology
- `out_dir` – Directory to put the whitelist

Returns
Path to whitelist

`kb_python.utils.`**`create_10x_feature_barcode_map`**`(`*out_path: str*`)` → str

Create a feature-barcode map for the 10x Feature Barcoding technology.

Parameters
`out_path` – Path to the output mapping file

Returns
Path to map

`kb_python.utils.`**`stream_file`**`(`*url: str*, *path: str*`)` → str

Creates a FIFO file to use for piping remote files into processes.

This function spawns a new thread to download the remote file into a FIFO file object. FIFO file objects are only supported on unix systems.

Parameters

- `url` – Url to the file
- `path` – Path to place FIFO file

Returns
Path to FIFO file

Raises
`UnsupportedOSError` – If the OS is Windows

`kb_python.utils.`**`read_t2g`**`(`*t2g_path: str*`)` → Dict[str, Tuple[str, Ellipsis]]

Given a transcript-to-gene mapping path, read it into a dictionary. The first column is always assumed to tbe the transcript IDs.

Parameters
`t2g_path` – Path to t2g

Returns

Dictionary containing transcript IDs as keys and all other columns
as a tuple as values

`kb_python.utils.`**`collapse_anndata`**`(`*adata: anndata.AnnData*, *by: Optional[str] = None*`)` → anndata.AnnData

Collapse the given Anndata by summing duplicate rows. The *by* argument specifies which column to use. If not provided, the index is used.

Note: This function also collapses any existing layers. Additionally, the returned AnnData will have the values used to collapse as the index.

Parameters

- **`adata`** – The Anndata to collapse
- **`by`** – The column to collapse by. If not provided, the index is used. When this column contains missing values (i.e. nan or None), these columns are removed.

Returns
A new collapsed Anndata object. All matrices are sparse, regardless of whether or not they were in the input Anndata.

`kb_python.utils.`**`import_tcc_matrix_as_anndata`**`(`*matrix_path: str*, *barcodes_path: str*, *ec_path: str*, *txnames_path: str*, *threads: int = 8*`)` → anndata.AnnData

Import a TCC matrix as an Anndata object.

Parameters

- **`matrix_path`** – Path to the matrix ec file
- **`barcodes_path`** – Path to the barcodes txt file
- **`genes_path`** – Path to the ec txt file
- **`txnames_path`** – Path to transcripts.txt generated by *kallisto bus*

Returns
A new Anndata object

`kb_python.utils.`**`import_matrix_as_anndata`**`(`*matrix_path: str*, *barcodes_path: str*, *genes_path: str*, *t2g_path: Optional[str] = None*, *name: str = 'gene'*, *by_name: bool = False*`)` → anndata.AnnData

Import a matrix as an Anndata object.

Parameters

- **`matrix_path`** – Path to the matrix ec file
- **`barcodes_path`** – Path to the barcodes txt file
- **`genes_path`** – Path to the genes txt file
- **`t2g_path`** – Path to transcript-to-gene mapping. If this is provided, the third column of the mapping is appended to the anndata var, defaults to *None*
- **`name`** – Name of the columns, defaults to "gene"
- **`by_name`** – Aggregate counts by name instead of ID. *t2g_path* must be provided and contain names.

Returns
A new Anndata object

`kb_python.utils.`**`overlay_anndatas`**(*adata_spliced: anndata.AnnData*, *adata_unspliced: anndata.AnnData*) → anndata.AnnData

'Overlays' anndata objects by taking the intersection of the obs and var of each anndata.

Note: Matrices generated by kallisto | bustools always contain all genes, even if they have zero counts. Therefore, taking the intersection is not entirely necessary but is done as a sanity check.

Parameters

- **`adata_spliced`** – An Anndata object
- **`adata_unspliced`** – An Anndata object

Returns
A new Anndata object

`kb_python.utils.`**`sum_anndatas`**(*adata_spliced: anndata.AnnData*, *adata_unspliced: anndata.AnnData*) → anndata.AnnData

Sum the counts in two anndata objects by taking the intersection of both matrices and adding the values together.

Note: Matrices generated by kallisto | bustools always contain all genes, even if they have zero counts. Therefore, taking the intersection is not entirely necessary but is done as a sanity check.

Parameters

- **`adata_spliced`** – An Anndata object
- **`adata_unspliced`** – An Anndata object

Returns
A new Anndata object

`kb_python.utils.`**`restore_cwd`**(*func: Callable*) → Callable

Function decorator to decorate functions that change the current working directory. When such a function is decorated with this function, the current working directory is restored to its previous state when the function exits.

`kb_python.validate`

Module Contents

Functions

`validate_bus`(path: str)	Verify if the provided BUS file is valid.
`validate_mtx`(path: str)	Verify if the provided Matrix Market (.mtx) file is valid.
`validate`(path: str)	Validate a file.
`validate_files`(pre: bool = True, post: bool = True) → Callable	Function decorator to validate input/output files.

Attributes

`BUSTOOLS_INSPECT_PARSER`
`VALIDATORS`

`kb_python.validate.`**`BUSTOOLS_INSPECT_PARSER`**

`exception` `kb_python.validate.`**`ValidateError`**

Bases: `Exception`

Common base class for all non-exit exceptions.

`kb_python.validate.`**`validate_bus`**(*path: str*)

Verify if the provided BUS file is valid.

A BUS file is considered valid when *bustools inspect* can read the file + it has > 0 BUS records.

Parameters
`path` – Path to BUS file

Raises

- ***`ValidateError`*** – If the file failed verification
- **`subprocess.CalledProcessError`** – If the bustools command failed

`kb_python.validate.`**`validate_mtx`**(*path: str*)

Verify if the provided Matrix Market (.mtx) file is valid.

A BUS file is considered valid when the file can be read with *scipy.io.mmread*.

Parameters
`path` – Path to mtx file

Raises
`ValidateError` – If the file failed verification

`kb_python.validate.`**`VALIDATORS`**

`kb_python.validate.`**`validate`**(*path: str*)

Validate a file.

This function is a wrapper around all validation functions. Given a path, it chooses the correct validation function. This function assumes the file exists.

Parameters
`path` – Path to file

Raises
`ValidateError` – If the file failed verification

`kb_python.validate.`**`validate_files`**(*pre: bool = True*, *post: bool = True*) → Callable

Function decorator to validate input/output files.

This function does not validate when the current run is a dry run. The decorated function is expected to return a dictionary of paths as values.

Parameters

- `pre` – Whether to validate input files, defaults to *True*
- `post` – Whether to validate output files, defaults to *True*

Returns
Wrapped function

Package Contents

`kb_python.`**`__version__ = 0.27.3`**

PYTHON MODULE INDEX

INDEX