# Blacktie Documentation

***Release 0.2.1.2***

**Augustine Dunn**

July 21, 2013

# CONTENTS

Contents:

CHAPTER

ONE

# PROJECT SUMMARY

## 1.1 I want to collaborate with you!

Contact me at wadunn83@gmail.com if you are a Python coder and want to or already have made improvements on this code.

## 1.2 Introducing Blacktie: a simpler way to do RNA-seq using Tophat, Cufflinks, and CummeRbund

Leveraging multiple fastQ files full of RNA-seq reads into a coherent picture of gene expression and transcript models is a multi-step process. It requires the organization and coordination of many files of different types through many different program calls and output steps. Each step might take hours or days depending on your input data. Then, as you are writing up your work, sometimes weeks/months later, you see that a new version of the programs you use has come out. Do you need to re-run your analysis? What settings DID you use back then?

The Tophat/Cufflinks/CummeRbund group of programs makes quality RNA-seq analysis doable once you understand the process. But what about when its time for you to leave the lab and you need to "train" someone else to repeat your process? It can be a nightmare. Especially if the trainee is not yet comfortable with the command line.

This is why I wrote the Blacktie pipeline software. Its goals are to streamline and simplify the complex task of analyzing full RNA-seq experiments using these programs; to automatically record settings used and program output messages in a way that users can track them to data later; provide a base set of functions and classes that will allow users to create custom pipelines easily by editing a single file (or if they want: writing their own custom scripts).

### 1.2.1 Some of Blacktie's features include:

- simple installation
- simple command line interface that allows almost ANYBODY to fully automate and reliably repeat their analysis of RNA-seq data with Tophat/Cufflinks/CummeRbund
- send email updates to the user
- intelligently continue with the analysis if a single run fails
- run multiple, complex tophat/cufflinks experiments at once using a single command
- generates SGE qsub-able scripts for use with a computing cluster
- checks for R installation
- checks for cummeRbund library and walks user through installation if its not installed yet

- automatic preliminary CummeRbund Quality Control, Basic Differential Expression, and Basic Pattern Discovery plots using CummeRbund

Dedicated bioinformatics personnel can be few and far between. Blacktie aims to bring automated, reproducible RNA-seq with built-in record keeping to more labs so that your valuable data does not fester on your servers, and you can publish sooner.

## 1.3 Getting the code

The code is available from the Python Package Index or from its homepage: https://github.com/xguse/blacktie

Visit *Installation* for more detailed instructions on getting and building the package.

## 1.4 Issue tracking

If you find issues, bugs, or have feature requests, please go here to submit them: https://github.com/xguse/blacktie/issues

## 1.5 Blacktie Poster

To credit the use of blacktie please cite the poster using the DOI link provided.

> *Introducing Blacktie: a simpler way to do RNA-seq using Tophat/Cufflinks/CummeRbund*. Augustine Dunn. figshare. http://dx.doi.org/10.6084/m9.figshare.714149

CHAPTER

TWO

# INSTALLATION

## 2.1 Requirements

The following python modules must be installed for blacktie to function properly:

```
Mako>=0.7.3
PyYAML>=3.10
```

The following modules will provide useful but optional functionality:

```
pprocess>=0.5
rpy2
```

## 2.2 Installing the latest version from the git repository

**Note:** Git is a **very** useful tool to have installed and to know how to use. Learn more here and try it out here.

Clone the repo:

```
$ git clone git://github.com/xguse/blacktie.git
```

Install with any unmet requirements using `pip`:

```
$ [sudo] pip install -r blacktie/requirements.txt blacktie
```

Install using standard `setup.py` script:

```
$ cd blacktie
$ [sudo] python setup.py install
```

## 2.3 Use `pip` to obtain the package from PyPI

```
$ [sudo] pip install blacktie Mako PyYAML pprocess
```

## 2.4 Installing without using `git` or `pip` for the download

After installing the requirements:

```
$ wget https://github.com/xguse/blacktie/archive/master.zip
$ unzip master.zip
$ cd blacktie-master
$ [sudo] python setup.py install
```

## 2.5 Test to see whether the install worked

To test whether your installation was successful, open a new terminal session and type the following command.

```
$ blacktie
```

You should see the help text for blacktie and it should look something like this:

```
usage: blacktie [-h] [--version]
                [--prog {tophat,cufflinks,cuffmerge,cuffdiff,cummerbund,all}]
                [--hide-logs] [--no-email]
                [--mode {analyze,dry_run,qsub_script}]
                config_file

This script reads options from a yaml formatted file and organizes the
execution of tophat/cufflinks runs for multiple condition sets.

positional arguments:
config_file           Path to a yaml formatted config file containing setup
                        options for the runs.

optional arguments:
-h, --help            show this help message and exit
--version             Print version number.
--prog {tophat,cufflinks,cuffmerge,cuffdiff,cummerbund,all}
                        Which program do you want to run? (default: tophat)
--hide-logs           Make your log directories hidden to keep a tidy
                        'looking' base directory. (default: False)
--no-email            Don't send email notifications. (default: False)
--mode {analyze,dry_run,qsub_script}
                        1) 'analyze': run the analysis pipeline. 2) 'dry_run':
                        walk through all steps that would be run and print out
                        the command lines; however, do not send the commands
                        to the system to be run. 3) 'qsub_script': generate
                        bash scripts suitable to be sent to a compute
                        cluster's SGE through the qsub command. (default:
                        analyze)
```

If this worked, great!

CHAPTER

THREE

# GETTING STARTED

**Note:** Make sure that you have successfully installed the blacktie module before trying the activities below.

To test whether your installation was successful, open a new terminal session and type the following command.

```
$ blacktie
```

You should see the help text for blacktie and it should look something like this:

```
$ blacktie

usage: blacktie [-h] [--version]
    [--prog {tophat,cufflinks,cuffmerge,cuffdiff,all}]
    [--hide-logs] [--no-email]
    [--mode {analyze,dry_run,qsub_script}]
    config_file

This script reads options from a yaml formatted file and organizes the
execution of tophat/cufflinks runs for multiple condition sets.

positional arguments:
    config_file           Path to a yaml formatted config file containing setup
        options for the runs.

optional arguments:
    -h, --help            show this help message and exit
    --version             Print version number.
    --prog {tophat,cufflinks,cuffmerge,cuffdiff,all}
        Which program do you want to run? (default: tophat)
    --hide-logs           Make your log directories hidden to keep a tidy
        'looking' base directory. (default: False)
    --no-email            Don't send email notifications. (default: False)
    --mode {analyze,dry_run,qsub_script}
        1) 'analyze': run the analysis pipeline. 2) 'dry_run':
        walk through all steps that would be run and print out
        the command lines; however, do not send the commands
        to the system to be run. 3) 'qsub_script': generate
        bash scripts suitable to be sent to a compute
        cluster's SGE through the qsub command. (default:
        analyze)
```

If this worked, great! Let's move on to what all that means.

## 3.1 The –prog option

This tells `blacktie` which part of the pipeline you would like to run. Any part can be run individually as long as the correct files exist. You can also run the whole thing from `tophat` to `cuffdiff` in one fell swoop if you like!

## 3.2 The –hide-logs option

This names your log files so that they are hidden in "*nix" systems.

## 3.3 The –modes option

`blacktie` can run in three modes. The first, `analyze`, actually runs the pipeline and does the analyses. However, it can be useful to simply view what WOULD be done to make sure that `blacktie` is producing command line calls that match what you expected. For this, use the `dry_run` mode.

Further, if you are working on a compute cluster running something like a "Sun Grid Engine" (SGE) to which you must submit jobs using `qsub`, it may not be a good idea to submit a job running all of `blacktie` as a single `qsub` job. For this it can be helpful to have `blacktie` write all of your `qsub` scripts for you based on a template. Each bash script represents a single program call to the tophat/cufflinks suite.

---

**Note:** A starter template for SGE submission can be found here: `blacktie/examples/qsub.template`. You will want to become familiar with how Mako processes templates if you plan to customize this much.

---

Here is what the starter template looks like:

```
#!/bin/bash
#$ -S /bin/bash                                          # Use a real BASH shell on the worker no
#$ -q ${queues}                                          # What queues do you want to submit to
#$ -M ${email_addy}                                      # Send email updates to this address
#$ -m beas                                               # When to send an email update
#$ -e /data/users/dunnw/logs/${call_id}.e                # Write standard error to this file
#$ -o /data/users/dunnw/logs/${call_id}.o                # Write standard out to this file
#$ -N ${job_name}                                        # Name my job this
#$ -R y                                                  # Reserve cores for me until there are t
#$ -pe openmp ${core_range}                              # Use openmp for multiprocessor use and

LD_LIBRARY_PATH="${ld_library_path}$${}{LD_LIBRARY_PATH}"   # Make sure worker's LD_LIBRARY_PATH con


# HPC clusters frequently use a module system to provide system wide access to
# certain programs.  The following makes sure that the tools needed are loaded
# for **MY** cluster. You will need alter this to make sure your cluster is set up
# based on its system.

module load bowtie2/2.0.2
module load tophat/2.0.6
module load cufflinks/2.0.2
module load samtools/0.1.18


# basic staging stuff
DATAHOME="${datahome}"
```

```
MYSCRATCH="/scratch/$${}{USER}"


mkdir -p $MYSCRATCH
cd $MYSCRATCH


# Remind me what will be done
echo ''
echo "${cmd_str}"
echo ''

# Run my job
${cmd_str}


# Pack up results and send it home to log-in node
tar -zcvf ${call_id}.tar.gz ${out_dir}
cp ${call_id}.tar.gz $${}{DATAHOME}/

# Back into the shadows
cd $HOME
rm -rf $MYSCRATCH
```

## 3.4 The configuration file

The configuration file is a YAML-based document that is where we will store all of the complexity of the options, input and output files of the typical tophat/cufflinks workflow. This way we have though about what we want to do with our RNA-seq data from start to finish before we actually start the analysis. Also, this config file acts as a check on our poor memory. If you get strange results you don't have to worry about whether you entered the samples backwards since you can go back to this config file and see exactly what files and settings were used.

---

**Note:** If you are running blacktie in `analyze` mode, you will have many more files created that document every step of the process where the output files are actually placed as well as central log files.

---

Here is a dummy example of a config file:

---

**Note:** A copy of this file can be found here: `blacktie/examples/blacktie_config_example.yaml`

---

```
# The document starts after the '---'

# By the way: everything after a '#' on a line
# will be ignored by the program and acts as a
# comment or note to explain things.

---
# run_options is a dictionary that contains variables that will be needed for
# many or all stages of the run
run_options:
    base_dir: /path/to/project/base_dir
    run_id: False            #  name your run: if false; uses current date/time for uniqe run_id eve
    bowtie_indexes_dir: /path/to/bowtie2_indexes
    email_info:
```

```
        sender: from_me@gmail.com
        to: to_you@email.com
        li: /path/to/file/containing/base64_encoded/login_info      # base64_encoded pswrd for from_
    custom_smtp:
        host: smtp.gmail.com   # or what ever your email smtp server is
        port: 587              # or which ever port your smtp server uses



# 'tophat_options':
# -----------------
# This is a dictionary that contains variables needed for all the tophat runs.
# The names of the key:value combinations are taken directly from the tophat
# option names but have the leading '-' removed.

# -o becomes o; --library-type becomes library-type

# **This is true for the cufflinks, cuffmerge, cuffdiff option dictionaries.**

# 'from_conditions':
# ------------------
# This is a special value that tells blacktie that you don't want to name a single
# value for this option but would rather set the value individually for each of
# your samples/conditions.  If you set the 'o' value here:

#    **all of your different sample results would
#      be written to the same output directory and
#      each would overwrite the next!**
# Hence: from_conditions

# However if you made all of your libraries the same way, things like 'r' and
# 'mate-std-dev' can be set here to avoid writing the same values over and over
# and perhaps making a mistake or two.

# 'positional_args':
# ------------------
# This is a dictionary inside of the 'tophat_options' dictionary.
# It is where you put the arguments to tophat that do not have 'flags' to make
# their identity explicit like '-o path/to/output_dir' or '--library-type fr-unstranded'

# For tophat, these values are
#     [1] the bowtie index name
#     [2] the fastq files containing the left_reads
#     [3] the fastq files containing the right_reads

# They will be different for cufflinks, cuffmerge, cuffdiff so consult the
# respective help text or manuals, but you should be fine if you just use what
# I have set up in this file already.

tophat_options:
    o: from_conditions
    library-type: fr-unstranded
    p: 6
    r: 125
    mate-std-dev: 25
    G: from_conditions
    no-coverage-search: True
    positional_args:
```

```
        bowtie2_index: from_conditions
        left_reads: from_conditions
        right_reads: from_conditions

cufflinks_options:
    o: from_conditions
    p: 7
    GTF-guide: from_conditions  # If you want to use annotation as *TRUTH* set this to False and set
    GTF: False                  # if an option set to false, it will be ommited from the command str
    3-overhang-tolerance: 5000
    frag-bias-correct: from_conditions
    multi-read-correct: True
    upper-quartile-norm: True
    positional_args:
        accepted_hits: from_conditions

cuffmerge_options:
    o: from_conditions # output directory
    ref-gtf: from_conditions
    p: 6
    ref-sequence: from_conditions
    positional_args:
        assembly_list: from_conditions # file with path to cufflinks gtf files to be merged

cuffdiff_options:
    o: from_conditions
    labels: from_conditions
    p: 6
    time-series: True
    upper-quartile-norm: True
    frag-bias-correct: from_conditions
    multi-read-correct: True
    positional_args:
        transcripts_gtf: from_conditions
        sample_bams: from_conditions


cummerbund_options:
    cuffdiff-dir: from_conditions
    gtf-path: from_conditions
    out: from_conditions
    file-type: pdf


# options for --mode qsub_script
# If you are not using --mode qsub_script, then set all to 'None'
qsub_options:
  queues: 'queue1,queue3,queue5'
  datahome: '/path/to/baseDirectory/on/cluster/'
  core_range: 40-64 # how many cpus do you want
  ld_library_path: ''  # leave this blank unless you know what it is and need it
  template: /path/to/your/altered/version/of/qsub.template


# 'condition_queue':
# ------------------
# This is a list of info related to each sample/condition contained in your RNA-sequence
# experiment(s)
```

```
131
132 # 'name': the name of this condition program. Usually something like a time-point
133 #         ID or treatment type. Should be as short as possible while still being a useful label.
134
135 # 'experiment_id': this is how you group different experiments to be included in a
136 #             single cuffmerge/cuffdiff program call.  All conditions in a time
137 #             series should share the same 'experiment_id' and be placed in
138 #             'condition_queue' in the order that you want them to be sent to
139 #             cuffdiff.
140
141 # 'replicate_id': this is how you group data for biological replicates of a single
142 #             experimental condition experiments to be included in a cuffdiff program
143 #             call.  Each replicate of a condition should have a unique 'experiment_id'.
144
145 # 'left_reads': a list of the paths to fastq files containing left reads for
146 #               each condition.
147
148 # 'right_reads': list of fastqs containing the right mates for the fastqs in
149 #                'left_reads'.
150 #                 **NOTE** right mate file must be in same order as provided to 'left_reads'
151
152 condition_queue:
153     -
154         name: exp1_control
155         experiment_id: 0
156         replicate_id: 0
157         left_reads:
158             - /path/to/exp1_control/techRep1.left_reads.fastq
159             - /path/to/exp1_control/techRep2.left_reads.fastq
160         right_reads:
161             - /path/to/exp1_control/techRep1.right_reads.fastq
162             - /path/to/exp1_control/techRep2.right_reads.fastq
163         genome_seq: /path/to/species/genome.fa
164         gtf_annotation: /path/to/species/annotation.gtf
165         bowtie2_index: species.bowtie2_index.basename
166
167     -
168         name: exp1_control
169         experiment_id: 0
170         replicate_id: 1
171         left_reads:
172             - /path/to/exp1_control/techRep1.left_reads.fastq
173             - /path/to/exp1_control/techRep2.left_reads.fastq
174         right_reads:
175             - /path/to/exp1_control/techRep1.right_reads.fastq
176             - /path/to/exp1_control/techRep2.right_reads.fastq
177         genome_seq: /path/to/species/genome.fa
178         gtf_annotation: /path/to/species/annotation.gtf
179         bowtie2_index: species.bowtie2_index.basename
180
181     -
182         name: exp1_treatment
183         experiment_id: 0
184         replicate_id: 0
185         left_reads:
186             - /path/to/exp1_treatment/techRep1.left_reads.fastq
187             - /path/to/exp1_treatment/techRep2.left_reads.fastq
188         right_reads:
```

```
            - /path/to/exp1_treatment/techRep1.right_reads.fastq
            - /path/to/exp1_treatment/techRep2.right_reads.fastq
        genome_seq: /path/to/species/genome.fa
        gtf_annotation: /path/to/species/annotation.gtf
        bowtie2_index: species.bowtie2_index.basename

    -
        name: exp2_control
        experiment_id: 1
        replicate_id: 0
        left_reads:
            - /path/to/exp2_control/techRep1.left_reads.fastq
            - /path/to/exp2_control/techRep2.left_reads.fastq
        right_reads:
            - /path/to/exp2_control/techRep1.right_reads.fastq
            - /path/to/exp2_control/techRep2.right_reads.fastq
        genome_seq: /path/to/species/genome.fa
        gtf_annotation: /path/to/species/annotation.gtf
        bowtie2_index: species.bowtie2_index.basename

    -
        name: exp2_treatment
        experiment_id: 1
        replicate_id: 0
        left_reads:
            - /path/to/exp2_treatment/techRep1.left_reads.fastq
            - /path/to/exp2_treatment/techRep2.left_reads.fastq
        right_reads:
            - /path/to/exp2_treatment/techRep1.right_reads.fastq
            - /path/to/exp2_treatment/techRep2.right_reads.fastq
        genome_seq: /path/to/species/genome.fa
        gtf_annotation: /path/to/species/annotation.gtf
        bowtie2_index: species.bowtie2_index.basename


...
```

**Todo**

Add the slots for custom email server options.

## 3.5 Using e-mail notifications

Changed in version v0.2.0rc1: any smtp server should now be usable if you code the host and port into the yaml config file. *Any* email can be used as the recipient.New in version v0.2.0rc1: added `--no-email` option.

> **Warning:** gmail's 2-step authentication will NOT work. Sorry. I will look into how to deal with that eventually.

You will need to provide your password in order to use the email notifications but it is not a good idea to store human readable passwords lying around your system. So the file that is used to store your password must contain a version of your password that has been encoded in base64. This will scramble your password beyond most people's ability to read it as a password as long as you don't name it something silly like `password_file.txt`.

The help text for `blacktie-encode` is:

```
$ blacktie-encode -h

usage: blacktie-encode [-h] input_file

This script takes a path to a file where you have placed your password for the
email you want blacktie to use as the "sender" in its notification emails. It
will replace the file with one containing your password once it has encoded it
out of human readable plain-text into seemingly meaningless text. **THIS IS
NOT FOOLPROOF:** If someone knows exactly what to look for they might figure
it out. ALWAYS use good password practices and never use the same password for
multiple important accounts!

positional arguments:
    input_file  Path to a file where you have placed your password for the email
        you want blacktie to use as the "sender" in its notification
        emails.

optional arguments:
    -h, --help  show this help message and exit
```

CHAPTER

FOUR

# TUTORIAL

A more detailed tutorial is under development, so *watch this space!*

CHAPTER

FIVE

# BLACKTIE AUTO-GENERATED CODE DOCUMENTATION

**Todo**

**DONE** Convert docstring style from (given,does,returns) to (:param a: format)

## 5.1 calls.py

Code defining classes to represent and excute pipeline program calls.

**class** `blacktie.utils.calls.`**`BaseCall`**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode='analyze'*)

Defines common methods for all program call types.

**`__init__`**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode='analyze'*)

initializes a `BaseCall` object

**Parameters**

- **yargs** – argument tree generated by parsing the yaml config file
- **email_info** – Bunch() object containing keys: `email_from`, `email_to`, `email_li`
- **run_id** – id for the whole set of calls
- **run_logs** – the directory where log file should be put
- **conditions** – one or a list of condition-dictionaries from `yargs.condition_queue`
- **mode** – choices = ['analyze','dry_run','qsub_script']

**Returns** an initialized `BaseCall` object

**`_flag_out_dir`**()

renames out directory, prepending 'FAILED' flag: equivalent of `mv tophat_Aa0 FAILED.tophat_Aa0`

**`build_out_dir_path`**()

builds correct `out_dir` path based on state of `self`

**Returns** `out_dir`

**build_qsub**()
Builds and writes this CallObject's qsub script to current working directory using options provided under the "qsub_options" sub-tree in the yaml config file.

**construct_options_list**()
converts `opt_dict` into list encoding proper options to send to the current program: saves to `self`.

**execute**()
calls correct program, records results, and manages errors

**get_condition_id**(*condition_dict*)
Constructs condition ID :param condition_dict: a dictionary containing consition info like name, replicate_id, etc. :returns: an ID used to construct the call_id of a call.

**init_log_file**()
creates empty log file for this call and stores its path in `self.log_file`

**init_opt_dict**()
builds a dict with non-job-specific values set and job-specific values set to False based on option names in the yaml file for this phase

**Returns** partially populated `opt_dict`

**log_end**()
records command string used, program output, and the end of call in `self.log_file`

**log_msg**(*log_msg=''*)

- opens `self.log_file`
- writes `log_msg`
- closes `self.log_file`

**log_start**()
records start of call in `self.log_file`

**notify_end_of_call**()
sends notification email informing user that `self.call_id` has exited

**notify_start_of_call**()
sends notification email informing user that `self.call_id` has been initiated

**purge_progress_bars**(*stderr_str*)
removes the dynamic progress bars included in some output in case user did not turn them off

**set_call_id**()
builds and stores this call's call ID in `self.call_id`

**class** `blacktie.utils.calls.`**CuffdiffCall**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode*)
Manage a single call to cuffdiff and store associated run data.

**__init__**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode*)
initializes the `CuffdiffCall` object

**Parameters**

- **yargs** – argument tree generated by parsing the yaml config file
- **email_info** – Bunch() object containing keys: `email_from`, `email_to`, `email_li`
- **run_id** – id for the whole set of calls
- **run_logs** – the directory where log file should be put

- **conditions** – one or a list of condition-dictionaries from `yargs.condition_queue`
- **mode** – choices = ['analyze','dry_run','qsub_script']

**Returns** an initialized `CuffdiffCall` object

**get_bam_path**(*condition*)
Supports `self.get_sample_bams()`.

**get_cuffmerge_gtf**()
Handles `yaml_config.cuffdiff_options.positional_args.transcripts_gtf: from_conditions`.

**get_genome**()
Handles `yaml_config.cuffdiff_options.frag-bias-correct: from_conditions`.

**get_labels**()
Handles `yaml_config.cuffdiff_options.labels: from_conditions`.

**get_mask_file**()
Handles `yaml_config.cuffdiff_options.mask-file: from_conditions`.

**get_out_dir**()
Handles `yaml_config.cuffdiff_options.o: from_conditions`.

**get_sample_bams**()
Handles `yaml_config.cuffdiff_options.positional_args.sample_bams: from_conditions`.

**class** `blacktie.utils.calls.`**CufflinksCall**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode*)
Manage a single call to cufflinks and store associated run data.

**__init__**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode*)
initializes the `CufflinksCall` object

**Parameters**

- **yargs** – argument tree generated by parsing the yaml config file
- **email_info** – Bunch() object containing keys: `email_from`, `email_to`, `email_li`
- **run_id** – id for the whole set of calls
- **run_logs** – the directory where log file should be put
- **conditions** – one or a list of condition-dictionaries from `yargs.condition_queue`
- **mode** – choices = ['analyze','dry_run','qsub_script']

**Returns** an initialized `CufflinksCall` object

---

**Todo**

**DONE** add support for –GTF in addition to currently supported –GTF-guide

---

**get_accepted_hits**()
Handles `yaml_config.cufflinks_options.positional_args.accepted_hits: from_conditions`.

**get_bam_path**()
Supports `self.get_accepted_hits()`.

**get_genome**()
Handles yaml_config.cufflinks_options.frag-bias-correct: from_conditions.

**get_gtf_anno**()
Handles yaml_config.cufflinks_options.GTF: from_conditions.

**get_gtf_anno_guide**()
Handles yaml_config.cufflinks_options.GTF-guide: from_conditions.

**get_mask_file**()
Handles yaml_config.cufflinks_options.mask-file: from_conditions.

**get_out_dir**()
Handles yaml_config.cufflinks_options.o: from_conditions.

**verify_options**()
Makes sure that conflicting options were not imported from yaml config file.

---

**Todo**

**DONE** GTF and GTF-guide should not be used together but both can be ommited

---

**class** blacktie.utils.calls.**CuffmergeCall**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode*)
Manage a single call to cuffmerge and store associated run data.

**__init__**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode*)
initializes the CuffmergeCall object

**Parameters**

- **yargs** – argument tree generated by parsing the yaml config file
- **email_info** – Bunch() object containing keys: email_from, email_to, email_li
- **run_id** – id for the whole set of calls
- **run_logs** – the directory where log file should be put
- **conditions** – one or a list of condition-dictionaries from yargs.condition_queue
- **mode** – choices = ['analyze','dry_run','qsub_script']

**Returns** an initialized CuffmergeCall object

**get_cuffGTF_path**(*condition*)
Supports self.get_cufflinks_gtfs().

**get_cufflinks_gtfs**()
Handles yaml_config.cuffmerge_options.positional_args.assembly_list: from_conditions.

**get_genome**()
Handles yaml_config.cuffmerge_options.ref-sequence: from_conditions.

**get_gtf_anno**()
Handles yaml_config.cuffmerge_options.ref-gtf: from_conditions.

**get_out_dir**()
Handles yaml_config.cuffmerge_options.o: from_conditions.

**class** blacktie.utils.calls.**CummerbundCall**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode*)

Manage a single call to blacktie-cummerbund script and store associated run data.

**__init__**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode*)

initializes the CummerbundCall object

**Parameters**

- **yargs** – argument tree generated by parsing the yaml config file
- **email_info** – Bunch() object containing keys: email_from, email_to, email_li
- **run_id** – id for the whole set of calls
- **run_logs** – the directory where log file should be put
- **conditions** – one or a list of condition-dictionaries from yargs.condition_queue
- **mode** – choices = ['analyze','dry_run','qsub_script']

**Returns** an initialized CummerbundCall object

**get_cuffdiff_dir**()

Handles yaml_config.cummerbund_options.cuffdiff-dir: from_conditions.

**get_cuffmerge_gtf**()

Handles yaml_config.cummerbund_options.gtf-path: from_conditions.

**get_out_dir**()

Handles yaml_config.cummerbund_options.out: from_conditions.

**class** blacktie.utils.calls.**TophatCall**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode*)

Manage a single call to tophat and store associated run data.

**__init__**(*yargs*, *email_info*, *run_id*, *run_logs*, *conditions*, *mode*)

initializes the TophatCall object

**Parameters**

- **yargs** – argument tree generated by parsing the yaml config file
- **email_info** – Bunch() object containing keys: email_from, email_to, email_li
- **run_id** – id for the whole set of calls
- **run_logs** – the directory where log file should be put
- **conditions** – one or a list of condition-dictionaries from yargs.condition_queue
- **mode** – choices = ['analyze','dry_run','qsub_script']

**Returns** an initialized TophatCall object

**get_bt_idx**()

Handles yaml_config.tophat_options.positional_args.bowtie2_index: from_conditions.

**get_gtf_anno**()

Handles yaml_config.tophat_options.G: from_conditions.

**get_lt_reads**()

Handles yaml_config.tophat_options.positional_args.left_reads: from_conditions.

**get_out_dir**()

Handles yaml_config.tophat_options.o: from_conditions.

**get_rt_reads**()
Handles yaml_config.tophat_options.positional_args.right_reads: from_conditions.

## 5.2 errors.py

Code defining custom base error classes to provide a foundation for graceful error handling.

**exception** blacktie.utils.errors.**BlacktieError**
Base class for exceptions in the blacktie package.

**exception** blacktie.utils.errors.**InvalidFileFormatError**
When errors occur due to malformed file formats.

**exception** blacktie.utils.errors.**MissingArgumentError**(*errMsg*)
When a required argument is missing from the parsed command line options.

**__init__**(*errMsg*)

**exception** blacktie.utils.errors.**SanityCheckError**
When a 'state check' comes back as conflicting or nonsensical.

**exception** blacktie.utils.errors.**SystemCallError**(*errno*, *strerror*, *filename=None*)
Error raised when a problem occurs while attempting to run an external system call.

**Attributes:**

errno – return code from system call
filename – file in volved if any
strerror – error msg

**__init__**(*errno*, *strerror*, *filename=None*)

**exception** blacktie.utils.errors.**UnexpectedValueError**
When values that "should" not be possible happen; like if a variable was changed unexpectedly.

## 5.3 externals.py

Code facilitating the execution of external system calls.

blacktie.utils.externals.**mkdirp**(*path*)
Create new dir while creating any parent dirs in the path as needed.

blacktie.utils.externals.**runExternalApp**(*progName*, *argStr*)
Convenience func to handle calling and monitoring output of external programs.

**Parameters**

- **progName** – name of system program command
- **argStr** – string containing command line options for progName

**Returns** subprocess.communicate object

blacktie.utils.externals.**whereis**(*program*)
returns path of program if it exists in your $PATH variable or None otherwise

## 5.4 misc.py

Code facilitating random aspects of this package.

**class** blacktie.utils.misc.**Bunch**(**args*, ***kwds*)
A dict like class to facilitate setting and access to tree-like data. Allows access to dictionary keys through 'dot' notation: "yourDict.key = value".

**__init__**(**args*, ***kwds*)

blacktie.utils.misc.**bunchify**(*dict_tree*)
Traverses a dictionary tree and converts all sub-dictionaries to Bunch() objects.

blacktie.utils.misc.**email_notification**(*sender*, *to*, *subject*, *txt*, *pw*, *server_info*)
Sends email to recipient using GMAIL server by default but will now accept `server_info` to customize this.

**Parameters**

- **sender** – email address of sender
- **to** – email addres of recipient
- **subject** – subject text
- **txt** – body text
- **pw** – password of `sender`
- **server_info** – dictionary = {'host':`str`,'port':`int`}

**Returns** None

---

**Todo**

**DONE** make `email_notification()` adjustable for other email servers

---

blacktie.utils.misc.**get_time**()
Return system time formatted as 'YYYY:MM:DD-hh:mm:ss'.

blacktie.utils.misc.**get_version_number**(*path_to_setup*)
Provides access to current version info contained in setup.py

blacktie.utils.misc.**map_condition_groups**(*yargs*)
creates a Bunch obj `groups` with key='experiment_id' from `yargs`, value=list(condition_queue objects with 'experiment_id')

**Parameters** **yargs** – argument object generated from the yaml config file

**Returns** `groups`

blacktie.utils.misc.**whoami**()
Returns the name of the currently active function.

CHAPTER

SIX

# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

# PYTHON MODULE INDEX

## b