drmr Documentation

Release 1.0.2

The Parker Lab

May 09, 2017

Contents

CHAPTER 1

Introduction

Drmr (pronounced 'drummer') lets you write computational pipelines in simple shell scripts. It's designed to work with common distributed resource management (DRM) systems (Slurm and PBS so far).

What's in the box?

- drmrc

  A tool to generate the drmr configuration file, `.drmrc`. It will try to detect your DRM automatically. You can also specify default accounts and destinations for running jobs.

- drmr

  The most capable way of submitting jobs, drmr scripts can specify job parameters for each command, and also express dependencies, where the pipeline should wait for previous jobs to complete before continuing.

- drmrarray

  For simple scripts whose every command can run concurrently with the same job parameters. If commands have different requirements, use drmr.

- drmrm

  A utility to make it easier to delete jobs from your DRM.

License

GPLv3 or any later version.

CHAPTER 2

Installation

Requirements

- Python. We've run it successfully under versions 2.7.10 and 3.5.
- Jinja2 (If you install drmr with pip, Jinja2 should be installed automatically.)
- lxml (Again, pip should install it for you.)

How to install

At the command line:

```
git clone https://github.com/ParkerLab/drmr
pip install ./drmr
```

Or in one step:

```
pip install git+https://github.com/ParkerLab/drmr
```

The dependencies should be installed automatically via pip's requirements.

CHAPTER 3

Usage

Configuration

The first thing you'll want to do is configure drmr for your local workload manager, by running `drmrc`. As long as you only have one workload manager installed, it should be able to detect it and create a reasonable configuration.

If you have a default account or destination (Slurm partition or PBS queue) you want to use for jobs that don't specify one, you can add that to the configuration. See *drmrc* in the *Command reference* section below for details.

Writing and submitting scripts

Once you've configured drmr, you're ready to write and submit a drmr script. Try putting this script in a file called *hello*:

```
echo "hello world"
```

Then run `drmr hello`. Almost nothing will happen. You should just see a number printed before the shell prompt comes back. This is the job ID of a success job, which drmr submits at the end of every script. You can monitor that job ID to see when your jobs finish.

The *hello* script has probably finished by the time you've read this far. The only indication will be a new directory named *.drmr*. In there you'll find a surprising number of files for a simple "hello world" example. The signal-to-noise ratio does improve as your scripts grow in size. The contents of *.drmr* should look something like this:

```
$ ls -1 .drmr
hello.1_174.out
hello.1.slurm
hello.cancel
hello.finish_176.out
hello.finished
hello.finish.slurm
hello.success
```

```
hello.success_175.out
hello.success.slurm
```

All of them start with *hello*, the job name derived automatically from your drmr script's filename. We could have explicitly set the job name instead, by submitting the script with drmr's `--job-name` option.

The file *hello.1.slurm* contains the actual job script. The name consists of the job prefix *hello*, the job number of the command in the drmr script (*1*), and a suffix indicating the DRM in use (*.slurm*, because I'm using Slurm for this example). The job script looks like this:

```
#!/bin/bash

####  Slurm preamble

#SBATCH --export=ALL
#SBATCH --job-name=hello.1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=4000
#SBATCH --output "/home/john/tmp/.drmr/hello.1_%j.out"
#SBATCH --workdir=/home/john/tmp


####  End Slurm preamble


####  Environment setup
. /home/john/.virtualenvs/drmr/bin/activate

####  Commands

echo "hello world"
```

It's pretty much what you'd write to submit any Slurm job. If you're using PBS, it will have a *.pbs* extension, and contain a PBS-specific preamble.

You'll notice that the last line is the command from your *hello* script.

In between is a nicety for Python users: if you have a virtual environment active when you submit a script, it will be activated before running your commands.

Each job script's standard output and error will be in a file named after the job, containing its DRM job ID. Here it's *hello.1_174.out*, and it contains:

```
hello world
```

Usually, your drmr scripts will contain commands explicitly redirecting their standard output to files, and you'll only refer to these default output files when jobs fail.

The rest of the files are drmr housekeeping: there's a script to cancel all the jobs (*hello.cancel*), completion jobs (*hello.finish.slurm* and *hello.success.slurm*) and their output files, and finally a couple of marker files: *hello.finished* and *hello.success*. That last one is what you want to see: if the *.success* file exists, all of your drmr script's jobs completed successfully. If you see the *.finished* file, but not *.success*, something went wrong.

A more complete example is included in the output of `drmr --help`, which you can read under *drmr* below. See also the real-world scripts under *Examples*.

Command reference

drmrc

Creates the drmr configuration file, *.drmrc*.

Help is available by running `drmrc --help`:

```
usage: drmrc [-h] [-a ACCOUNT] [-d DESTINATION] [-o]
                      [-r RESOURCE_MANAGER]

Generate a drmr configuration file for your local environment.

optional arguments:
  -h, --help            show this help message and exit
  -a ACCOUNT, --account ACCOUNT
                        The account to which jobs will be charged by default.
  -d DESTINATION, --destination DESTINATION
                        The default queue/partition in which to run jobs.
  -o, --overwrite       Overwrite any existing configuration file.
  -r RESOURCE_MANAGER, --resource-manager RESOURCE_MANAGER
                        If you have more than one resource manager available,
                        you can specify it.
```

drmr

Submits a pipeline script to a distributed resource manager. By default all the pipeline's commands are run concurrently, but you can indicate dependencies by adding `# drmr:wait` directives between jobs. Whenever a wait directive is encountered, the pipeline will wait for all prior jobs to complete before continuing.

You may also specify job parameters, like CPU or memory requirements, time limits, etc. in `# drmr:job` directives.

You can get help, including a full example, by running `drmr --help`:

```
usage: drmr [-h] [-a ACCOUNT] [-d DESTINATION] [--debug] [-j JOB_NAME]
            [-f FROM_LABEL] [--mail-at-finish] [--mail-on-error]
            [--start-held] [-t TO_LABEL] [-w WAIT_LIST]
            input

Submit a drmr script to a distributed resource manager.

positional arguments:
  input                 The file containing commands to submit. Use "-" for
                        stdin.

optional arguments:
  -h, --help            show this help message and exit
  -a ACCOUNT, --account ACCOUNT
                        The account to be billed for the jobs.
  -d DESTINATION, --destination DESTINATION
                        The queue/partition in which to run the jobs.
  --debug               Turn on debug-level logging.
  -j JOB_NAME, --job-name JOB_NAME
                        The job name.
  -f FROM_LABEL, --from-label FROM_LABEL
                        Ignore script lines before the given label.
```

```
  --mail-at-finish      Send mail when all jobs are finished.
  --mail-on-error       Send mail if any job fails.
  --start-held          Submit a held job at the start of the pipeline, which
                        must be released to start execution.
  -t TO_LABEL, --to-label TO_LABEL
                        Ignore script lines after the given label.
  -w WAIT_LIST, --wait-list WAIT_LIST
                        A colon-separated list of job IDs that must complete
                        before any of this script's jobs are started.

Supported resource managers are:

  Slurm
  PBS

drmr will read configuration from your ~/.drmrc, which must be
valid JSON. You can specify your resource manager and default
values for any job parameters listed below.

Directives
==========

Your script can specify job control directives in special
comments starting with "drmr:".

# drmr:wait

  Drmr by default runs all the script's commands
  concurrently. The wait directive tells drmr to wait for
  any jobs started since the last wait directive, or the
  beginning of the script, to complete successfully.

# drmr:label

  Labels let you selectively run sections of your script: you can
  restart from a label with --from-label, running everything after
  it, or just the commands before the label given with --to-label.

# drmr:job

  You can customize the following job parameters:

  time_limit: The maximum amount of time the DRM should allow the job: "12:30:00" or
↪"12h30m".
  working_directory: The directory where the job should be run.
  processor_memory: The amount of memory required per processor.
  node_properties: A comma-separated list of properties each node must have.
  account: The account to which the job will be billed.
  processors: The number of cores required on each node.
  default: Use the resource manager's default job parameters.
  destination: The execution environment (queue, partition, etc.) for the job.
  job_name: A name for the job.
  memory: The amount of memory required on any one node.
  nodes: The number of nodes required for the job.
  email: The submitter's email address, for notifications.

  Whatever you specify will apply to all jobs after the directive.
```

```
  To revert to default parameters, use:

  # drmr:job default

  To request 4 CPUs, 8GB of memory per processor, and a
  limit of 12 hours of execution time on one node:

  # drmr:job nodes=1 processors=4 processor_memory=8000 time_limit=12:00:00

Example
=======

A complete example script follows:

#!/bin/bash

#
# Example drmr script. It can be run as a normal shell script, or
# submitted to a resource manager with the drmr command.
#

#
# You can just write commands as you would in any script. Their output
# will be captured in files by the resource manager.
#
echo thing1

#
# You can only use flow control within a command; drmr's parser is not
# smart enough to deal with conditionals, or create jobs for each
# iteration of a for loop, or anything like that.
#
# You can do this, but it will just all happen in a single job:
#
for i in $(seq 1 4); do echo thing${i}; done

#
# Comments are OK.
#
echo thing2  # even trailing comments

#
# Line continuations are OK.
#
echo thing1 \
     thing2 \
     thing3

#
# Pipes are OK.
#
echo funicular berry harvester | wc -w

#
# The drmr wait directive makes subsequent tasks depend on the
# successful completion of all jobs since the last wait directive or
# the start of the script.
#
```

```
# drmr:wait
echo "And proud we are of all of them."

#
# You can specify job parameters:
#

# drmr:job nodes=1 processors=4 processor_memory=8000 time_limit=12:00:00
echo "I got mine but I want more."

#
# And revert to the defaults defined by drmr or the resource manager.
#

# drmr:job default
echo "This job feels so normal."

# drmr:wait
# drmr:job time_limit=00:15:00
echo "All done!"

# And finally, a job is automatically submitted to wait on all the
# other jobs and report success or failure of the entire script.
# Its job ID will be printed.
```

drmrarray

If you have hundreds or thousands of tasks that don't depend on each other, you can make life easier for yourself and your DRM by submitting them in a job array with *drmrarray*. Both Slurm and PBS cope better with lots of jobs if they're part of an array, and it's definitely easier to make sense of the DRM's status output when it doesn't contain hundreds or thousands of lines.

With *drmrarray*, job parameters can only be specified once, at the top of the script, and will apply to all jobs in the array. And of course you cannot define dependencies. You can, however, run whatever program you like on each line of the script you feed to drmrarray.

You can get help, including a full example, by running `drmrarray --help`:

```
usage: drmrarray [-h] [-a ACCOUNT] [-d DESTINATION] [--debug] [-f]
                 [-j JOB_NAME] [--mail-at-finish] [--mail-on-error]
                 [-s SLOT_LIMIT] [-w WAIT_LIST]
                 input

Submit a drmr script to a distributed resource manager as a job array.

positional arguments:
  input                 The file containing commands to submit. Use "-" for
                        stdin.

optional arguments:
  -h, --help            show this help message and exit
  -a ACCOUNT, --account ACCOUNT
                        The account to be billed for the jobs.
  -d DESTINATION, --destination DESTINATION
                        The queue/partition in which to run the jobs.
  --debug               Turn on debug-level logging.
```

```
  -f, --finish-jobs     If specified, two extra jobs will be queued after the
                        main array, to indicate success and completion.
  -j JOB_NAME, --job-name JOB_NAME
                        The job name.
  --mail-at-finish      Send mail when all jobs are finished.
  --mail-on-error       Send mail if any job fails.
  -s SLOT_LIMIT, --slot-limit SLOT_LIMIT
                        The number of jobs that will be run concurrently when
                        the job is started, or 'all' (the default).
  -w WAIT_LIST, --wait-list WAIT_LIST
                        A colon-separated list of job IDs that must complete
                        before any of this script's jobs are started.

Supported resource managers are:

  Slurm
  PBS

drmrarray will read configuration from your ~/.drmrc, which must be valid
JSON. You can specify your resource manager and default values for any job
parameters listed below.

Directives
==========

Your script can specify job parameters in special comments starting
with "drmr:job".

# drmr:job

  You can customize the following job parameters:

  time_limit: The maximum amount of time the DRM should allow the job: "12:30:00" or
↪"12h30m".
  working_directory: The directory where the job should be run.
  processor_memory: The amount of memory required per processor.
  node_properties: A comma-separated list of properties each node must have.
  account: The account to which the job will be billed.
  processors: The number of cores required on each node.
  default: Use the resource manager's default job parameters.
  destination: The execution environment (queue, partition, etc.) for the job.
  job_name: A name for the job.
  memory: The amount of memory required on any one node.
  nodes: The number of nodes required for the job.
  email: The submitter's email address, for notifications.

  Whatever you specify will apply to all jobs after the directive.

  To revert to default parameters, use:

  # drmr:job default

  To request 4 CPUs, 8GB of memory per processor, and a
  limit of 12 hours of execution time on one node:

  # drmr:job nodes=1 processors=4 processor_memory=8000 time_limit=12:00:00
```

drmrm

A drmr script can generate a lot of jobs. Deleting them with the DRM tools (e.g. qdel, scancel) can be cumbersome, so drmrm tries to make it easier. Help is available by running `drmrm --help`

```
usage: drmrm [-h] [--debug] [-n] [-j JOB_NAME] [-u USER] [job_id [job_id ...]]

Remove jobs from a distributed resource manager.

positional arguments:
  job_id                A job ID to remove.

optional arguments:
  -h, --help            show this help message and exit
  --debug               Turn on debug-level logging.
  -n, --dry-run         Just print jobs that would be removed, without
                        actually removing them.
  -j JOB_NAME, --job-name JOB_NAME
                        Remove only jobs whose names contain this string.
  -u USER, --user USER  Remove only jobs belonging to this user.
```

CHAPTER 4

Examples

Bioinformatics

ATAC-seq pipeline

This sample script does some basic processing of the data from the original ATAC-seq paper (https://dx.doi.org/10.1038/nmeth.2688). First, the sequence data is cleaned up by removing adapter sequence, then it's aligned to the reference genome with bwa, then a subset of the alignments are selected for peak calling, which is done with macs2.

Note the *drmr:label* and *drmr:job* directives. Often when you're developing a pipeline, you need to correct it and run it again. Maybe you didn't specify enough memory for a job and it was killed by the resource manager, stopping the entire pipeline, or you want to tweak some program arguments and rerun just that step. You can do that with drmr's `--from-label` and `--to-label` options.

```
#!/bin/bash
# -*- mode: sh; coding: utf-8 -*-

#
# trim adapter sequence from reads
#

# drmr:label trim-adapters
# drmr:job time_limit=4h working_directory=/drmr/example/atac-seq

/usr/bin/time -v ionice -c3 cta SRR891268_1.fq.gz SRR891268_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891269_1.fq.gz SRR891269_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891270_1.fq.gz SRR891270_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891271_1.fq.gz SRR891271_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891272_1.fq.gz SRR891272_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891273_1.fq.gz SRR891273_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891274_1.fq.gz SRR891274_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891275_1.fq.gz SRR891275_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891276_1.fq.gz SRR891276_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891277_1.fq.gz SRR891277_2.fq.gz
```

```
/usr/bin/time -v ionice -c3 cta SRR891278_1.fq.gz SRR891278_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891279_1.fq.gz SRR891279_2.fq.gz
/usr/bin/time -v ionice -c3 cta SRR891280_1.fq.gz SRR891280_2.fq.gz

# drmr:wait

#
# align the reads to the hg19 reference genome
#

# drmr:label bwa
# drmr:job nodes=1 processors=4 working_directory=/drmr/example/atac-seq

/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891268_1.trimmed.fq.gz SRR891268_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891268.sort -o SRR891268.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891269_1.trimmed.fq.gz SRR891269_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891269.sort -o SRR891269.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891270_1.trimmed.fq.gz SRR891270_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891270.sort -o SRR891270.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891271_1.trimmed.fq.gz SRR891271_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891271.sort -o SRR891271.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891272_1.trimmed.fq.gz SRR891272_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891272.sort -o SRR891272.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891273_1.trimmed.fq.gz SRR891273_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891273.sort -o SRR891273.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891274_1.trimmed.fq.gz SRR891274_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891274.sort -o SRR891274.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891275_1.trimmed.fq.gz SRR891275_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891275.sort -o SRR891275.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891276_1.trimmed.fq.gz SRR891276_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891276.sort -o SRR891276.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891277_1.trimmed.fq.gz SRR891277_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891277.sort -o SRR891277.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891278_1.trimmed.fq.gz SRR891278_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891278.sort -o SRR891278.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891279_1.trimmed.fq.gz SRR891279_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891279.sort -o SRR891279.bam -
/usr/bin/time -v ionice -c3 bwa mem -t 4 /lab/data/reference/human/hg19/index/bwa/0.7.
↪12/hg19 SRR891280_1.trimmed.fq.gz SRR891280_2.trimmed.fq.gz | samtools sort -m 1g -
↪@ 4 -O bam -T SRR891280.sort -o SRR891280.bam -

# drmr:wait

# drmr:label mark-duplicates
# drmr:job nodes=1 processors=2 memory=9g working_directory=/drmr/example/atac-seq
```

```
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891268.
↪bam O=SRR891268.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891268.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891269.
↪bam O=SRR891269.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891269.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891270.
↪bam O=SRR891270.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891270.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891271.
↪bam O=SRR891271.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891271.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891272.
↪bam O=SRR891272.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891272.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891273.
↪bam O=SRR891273.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891273.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891274.
↪bam O=SRR891274.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891274.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891275.
↪bam O=SRR891275.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891275.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891276.
↪bam O=SRR891276.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891276.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891277.
↪bam O=SRR891277.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891277.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891278.
↪bam O=SRR891278.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891278.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891279.
↪bam O=SRR891279.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891279.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq
/usr/bin/time -v java -Xmx8g -jar $PICARD_HOME/picard.jar MarkDuplicates I=SRR891280.
↪bam O=SRR891280.md.bam ASSUME_SORTED=true METRICS_FILE=SRR891280.markdup.metrics␣
↪VALIDATION_STRINGENCY=LENIENT TMP_DIR=/drmr/example/atac-seq

# drmr:wait

#
# index the merged BAM files with marked duplicates, so we can prune them
#

# drmr:label index-sample-libraries

/usr/bin/time -v samtools index SRR891268.md.bam
/usr/bin/time -v samtools index SRR891269.md.bam
/usr/bin/time -v samtools index SRR891270.md.bam
/usr/bin/time -v samtools index SRR891271.md.bam
/usr/bin/time -v samtools index SRR891272.md.bam
/usr/bin/time -v samtools index SRR891273.md.bam
/usr/bin/time -v samtools index SRR891274.md.bam
/usr/bin/time -v samtools index SRR891275.md.bam
/usr/bin/time -v samtools index SRR891276.md.bam
/usr/bin/time -v samtools index SRR891277.md.bam
```

```
/usr/bin/time -v samtools index SRR891278.md.bam
/usr/bin/time -v samtools index SRR891279.md.bam
/usr/bin/time -v samtools index SRR891280.md.bam

# drmr:wait

#
# prune the BAM files with marked duplicates down to properly paired
# and mapped primary autosomal alignments of good quality, for peak calling
#

# drmr:label prune
# drmr:job nodes=1 processors=1 memory=4g time_limit=4h working_directory=/drmr/
↪example/atac-seq

/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891268.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891268.md.bam $CHROMOSOMES > SRR891268.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891269.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891269.md.bam $CHROMOSOMES > SRR891269.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891270.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891270.md.bam $CHROMOSOMES > SRR891270.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891271.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891271.md.bam $CHROMOSOMES > SRR891271.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891272.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891272.md.bam $CHROMOSOMES > SRR891272.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891273.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891273.md.bam $CHROMOSOMES > SRR891273.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891274.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891274.md.bam $CHROMOSOMES > SRR891274.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891275.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891275.md.bam $CHROMOSOMES > SRR891275.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891276.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891276.md.bam $CHROMOSOMES > SRR891276.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891277.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891277.md.bam $CHROMOSOMES > SRR891277.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891278.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891278.md.bam $CHROMOSOMES > SRR891278.pruned.bam)"
```

```
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891279.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891279.md.bam $CHROMOSOMES > SRR891279.pruned.bam)"
/usr/bin/time -v bash -c "(export CHROMOSOMES=$(samtools view -H SRR891280.md.bam |␣
↪grep '^@SQ' | cut -f 2 | grep -v -e _ -e chrM -e chrX -e chrY -e 'VN:' | sed 's/SN:/
↪/' | xargs echo); samtools view -b -h -f 3 -F 4 -F 8 -F 256 -F 1024 -F 2048 -q 30␣
↪SRR891280.md.bam $CHROMOSOMES > SRR891280.pruned.bam)"

# drmr:wait

#
# peak calling
#

# drmr:label macs2
# drmr:job nodes=1 processors=1 memory=8g working_directory=/drmr/example/atac-seq

/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891268.pruned.bam -f BAM -n␣
↪SRR891268.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891269.pruned.bam -f BAM -n␣
↪SRR891269.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891270.pruned.bam -f BAM -n␣
↪SRR891270.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891271.pruned.bam -f BAM -n␣
↪SRR891271.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891272.pruned.bam -f BAM -n␣
↪SRR891272.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891273.pruned.bam -f BAM -n␣
↪SRR891273.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891274.pruned.bam -f BAM -n␣
↪SRR891274.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891275.pruned.bam -f BAM -n␣
↪SRR891275.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891276.pruned.bam -f BAM -n␣
↪SRR891276.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891277.pruned.bam -f BAM -n␣
↪SRR891277.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891278.pruned.bam -f BAM -n␣
↪SRR891278.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891279.pruned.bam -f BAM -n␣
↪SRR891279.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
/usr/bin/time -v ionice -c3 macs2 callpeak -t SRR891280.pruned.bam -f BAM -n␣
↪SRR891280.broad -g hs --nomodel --shift -100 --extsize 200 -B --broad --keep-dup all
```

CHAPTER 5

Getting help

If you have questions about installing or using drmr, mail us:

parkerlab-software@umich.edu

If you've found a bug, please file a report at GitHub:

https://github.com/ParkerLab/drmr/issues/

CHAPTER 6

Contributing

Contributing knowledge

If you've found a bug or have a suggestion, please let us know by creating a GitHub issue at:

https://github.com/ParkerLab/drmr/issues

Contributing code

We welcome contributions of code, too. Here's how to get started.

1. Fork the repo on GitHub: https://github.com/ParkerLab/drmr/
2. Set up a Python virtualenv. With Python 2 and virtualenv:

   ```
   $ virtualenv drmr
   $ cd drmr; . ./bin/activate
   ```

   With Python 3 and pyvenv:

   ```
   $ pyvenv drmr
   $ cd drmr; . ./bin/activate
   ```

3. Check out your drmr repository:

   ```
   $ git clone git@github.com:your_name_here/drmr.git
   ```

4. Install drmr in the virtualenv, configured so that changes in your working copy are effective immediately:

   ```
   $ cd drmr/
   $ python setup.py develop
   ```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

6. *(Optional, but much appreciated.)* When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 drmr tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just *pip install* them into your virtualenv.

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. If the pull request adds functionality, please update the documentation, especially docstrings and script help text.
2. If you have time to write tests too, great, but we understand you're volunteering your time to help our project, and we will take care of making sure changes are tested.

CHAPTER 7

Credits

The drmr package was built by the Parker Lab at the University of Michigan.

CHAPTER 8

History

CHAPTER 9

1.0.0 (2016-06-15)

- First public release.

CHAPTER 10

Indices and tables

- genindex
- modindex