snakedwi

Ali Khan

Apr 25, 2022

CONTENTS:

BIDS app and snakemake workflow for dwi pre-processing

CHAPTER

ONE

INSTALLATION

Install from github:

```
pip install -e git+https://github.com/akhanf/snakedwi#egg=snakedwi
```

Note: you can re-run this command to re-install with the latest version

CHAPTER

TWO

RUNNING THE APP

Do a dry-run first (`-n`) and simply print (`-p`) what would be run:

```
snakedwi /path/to/bids/dir /path/to/output/dir participant -np
```

Run the app, using all cores:

```
snakedwi /path/to/bids/dir /path/to/output/dir participant --cores all
```

If any workflow rules require containers, then run with the `--use-singularity` option.

2.1 Generating a report

After your processing is complete, you can use snakemake's `--report` feature to generate an HTML report. This report will include a graph of all the jobs run, with clickable nodes to inspect the shell command or python code used in each job, along with the config files and run times for each job. Workflows may also contain append images for quality assurance or to summarize outputs, by using the `report(...)` function on any snakemake output.

To generate a report, run:

```
snakedwi /path/to/bids/dir /path/to/output/dir participant --report
```

CHAPTER

THREE

COMPUTE CANADA INSTRUCTIONS

3.1 Setting up a dev environment

Here are some instructions to get your python environment set-up on graham to run snakedwi:

1. Create a virtualenv and activate it:

```
cd $SCRATCH
module load python/3
virtualenv venv_snakedwi
source venv_snakedwi/bin/activate
```

2. Follow the steps above to install from github repository

3.2 Install job submission helpers

Snakemake can submit jobs with SLURM, but you need to set-up a Snakemake profile to enable this. The Khan lab has a snakemake profile that is configured for graham but is customizable upon install, please see cc-slurm for more info.

If you don't need Snakemake to parallelize jobs across different nodes, you can make use of the simple job submission wrappers in neuroglia-helpers, e.g. `regularSubmit` or `regularInteractive` wrappers.

These are used in the instructions below.

3.3 Running jobs on Compute Canada

In an interactive job (for testing):

```
regularInteractive -n 8
snakedwi bids_dir out_dir participant --participant_label 001 -j 8
```

Submitting a job (for larger cores, more subjects), still single job, but snakemake will parallelize over the 32 cores:

```
regularSubmit -j Fat snakedwi bids_dir out_dir participant  -j 32
```

Scaling up to ~hundred subjects (needs cc-slurm snakemake profile installed), submits 1 16core job per subject:

```
snakedwi bids_dir out_dir participant  --profile cc-slurm
```

Scaling up to even more subjects (uses group-components to bundle multiple subjects in each job), 1 32core job for N subjects (e.g. 10):

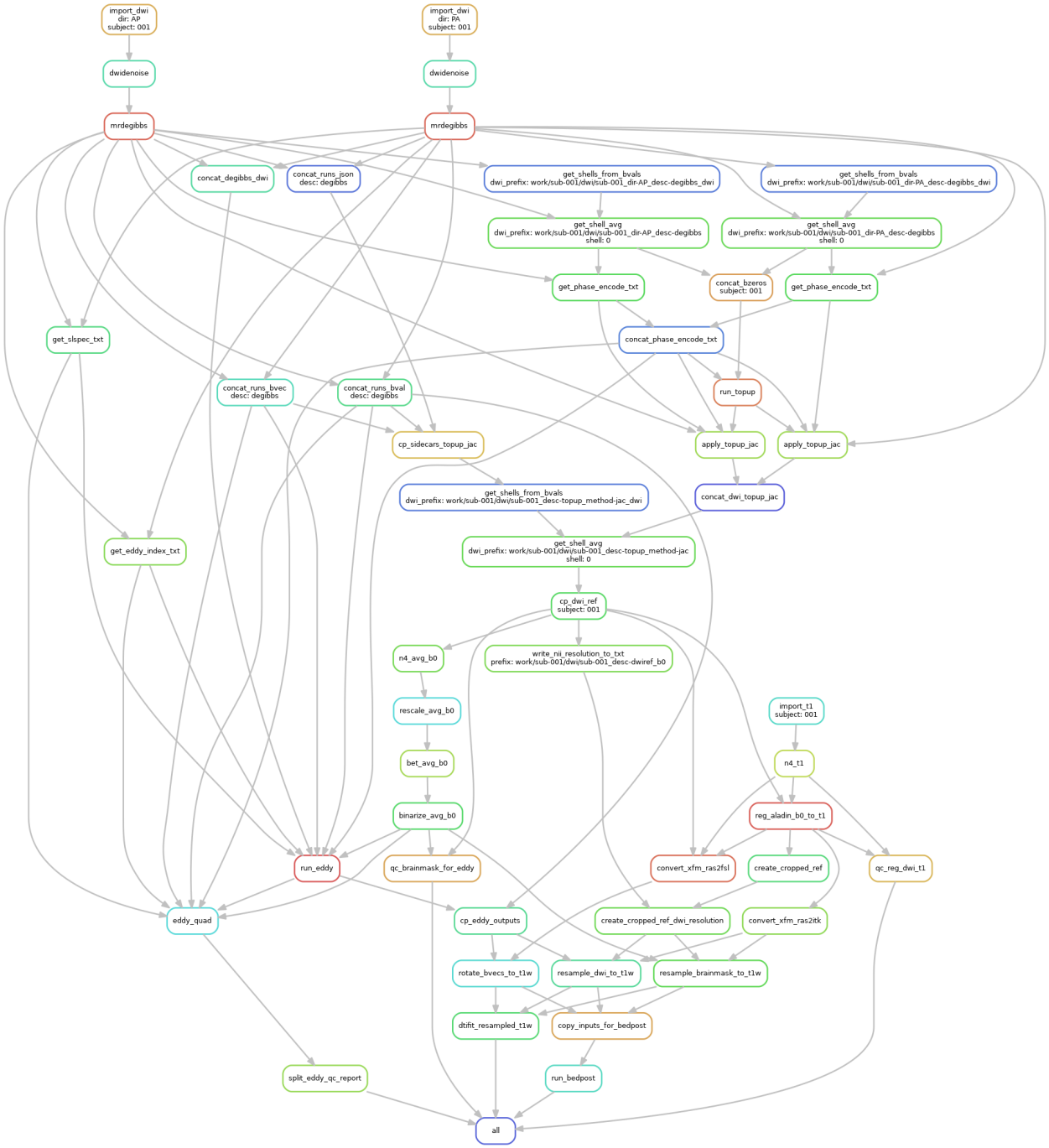
```
snakedwi bids_dir out_dir participant  --profile cc-slurm --group-components subj=10
```

CHAPTER

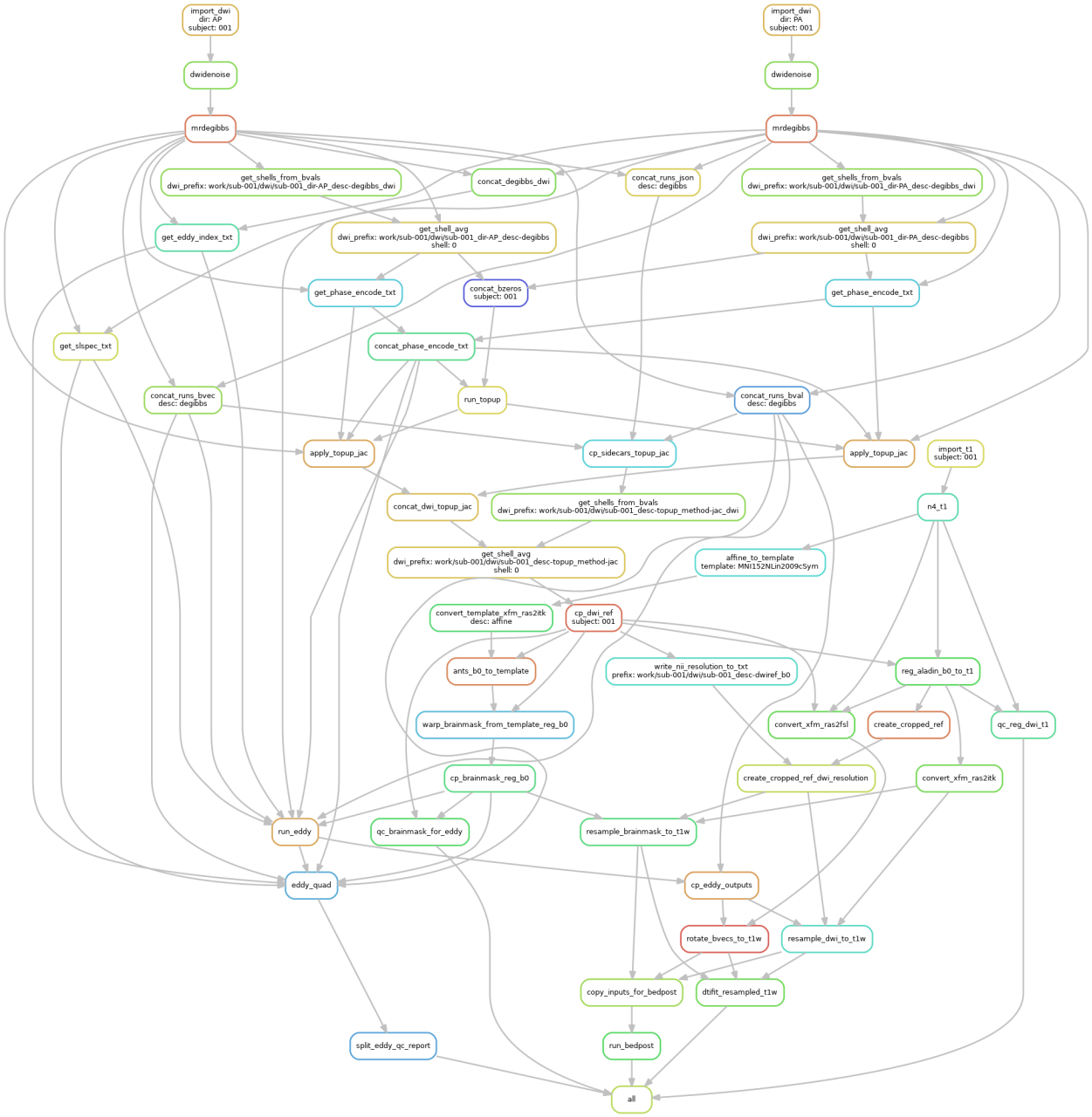
FOUR

WORKFLOWS

Workflow for AP-PA dwi with BET masking



Workflow for AP-PA dwi with SyN registration-based masking


import_dwi
dir: AP
subject: 001
import_dwi
dir: PA
subject: 001
dwidenoise
dwidenoise
mrdegibbs
mrdegibbs
get_shells_from_bvals
dwi_prefix: work/sub-001/dwi/sub-001_dir-AP_desc-degibbs_dwi
concat_degibbs_dwi
concat_runs_json
desc: degibbs
get_shells_from_bvals
dwi_prefix: work/sub-001/dwi/sub-001_dir-PA_desc-degibbs_dwi
get_eddy_index_txt
get_shell_avg
dwi_prefix: work/sub-001/dwi/sub-001_dir-AP_desc-degibbs
shell: 0
get_shell_avg
dwi_prefix: work/sub-001/dwi/sub-001_dir-PA_desc-degibbs
shell: 0
get_phase_encode_txt
concat_bzeros
subject: 001
get_phase_encode_txt
get_slspec_txt
concat_phase_encode_txt
concat_runs_bvec
desc: degibbs
run_topup
concat_runs_bval
desc: degibbs
apply_topup_jac
cp_sidecars_topup_jac
apply_topup_jac
import_t1
subject: 001
concat_dwi_topup_jac
get_shells_from_bvals
dwi_prefix: work/sub-001/dwi/sub-001_desc-topup_method-jac_dwi
n4_t1
get_shell_avg
dwi_prefix: work/sub-001/dwi/sub-001_desc-topup_method-jac
shell: 0
affine_to_template
template: MNI152NLin2009cSym
convert_template_xfm_ras2itk
desc: affine
cp_dwi_ref
subject: 001
ants_b0_to_template
write_nii_resolution_to_txt
prefix: work/sub-001/dwi/sub-001_desc-dwiref_b0
reg_aladin_b0_to_t1
warp_brainmask_from_template_reg_b0
convert_xfm_ras2fsl
create_cropped_ref
qc_reg_dwi_t1
cp_brainmask_reg_b0
create_cropped_ref_dwi_resolution
convert_xfm_ras2itk
run_eddy
qc_brainmask_for_eddy
resample_brainmask_to_t1w
eddy_quad
cp_eddy_outputs
rotate_bvecs_to_t1w
resample_dwi_to_t1w
copy_inputs_for_bedpost
dtifit_resampled_t1w
split_eddy_qc_report
run_bedpost
all

CHAPTER

FIVE

COMMAND LINE INTERFACE

snakebids-app

```
usage: snakedwi [-h]
                [--participant_label PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]]
                [--exclude_participant_label EXCLUDE_PARTICIPANT_LABEL [EXCLUDE_
↪PARTICIPANT_LABEL ...]]
                [--no_topup] [--no_bedpost] [--eddy_no_s2v] [--eddy_no_quad]
                [--masking_method [{b0_BET,b0_SyN}]]
                [--b0_bet_frac B0_BET_FRAC] [--use_gpu_eddy_container]
                [--use_gpu_bedpost_container]
                [--filter_dwi FILTER_DWI [FILTER_DWI ...]]
                [--filter_T1w FILTER_T1W [FILTER_T1W ...]]
                bids_dir output_dir {participant}
```

5.1 Positional Arguments

bids_dir The directory with the input dataset formatted according to the BIDS standard.

output_dir The directory where the output files should be stored. If you are running group level analysis this folder should be prepopulated with the results of the participant level analysis.

analysis_level Possible choices: participant

Level of the analysis that will be performed.

5.2 Named Arguments

--participant_label The label(s) of the participant(s) that should be analyzed. The label corresponds to sub-<participant_label> from the BIDS spec (so it does not include "sub-"). If this parameter is not provided all subjects should be analyzed. Multiple participants can be specified with a space separated list.

--exclude_participant_label The label(s) of the participant(s) that should be excluded. The label corresponds to sub-<participant_label> from the BIDS spec (so it does not include "sub-"). If this parameter is not provided all subjects should be analyzed. Multiple participants can be specified with a space separated list.

--no_topup Disable topup (default: False)

Default: False

--no_bedpost Disable bedpost (default: False)

Default: False

--eddy_no_s2v Disables slice-to-volume in eddy. This option must be used if your dwi json does not include SliceTiming (default: False)

Default: False

--eddy_no_quad Disables eddy_quad QC report (default: False)

Default: False

--masking_method Possible choices: b0_BET, b0_SyN

Brain masking method to use (default: "b0_BET")

Default: "b0_BET"

--b0_bet_frac BET fractional intensity threshold for b0 masking (default: 0.5)

Default: 0.5

--use_gpu_eddy_container Use GPU-based eddy container

Default: False

--use_gpu_bedpost_container Use GPU-based bedpost container

Default: False

--filter_dwi Filters (PyBIDS) for dwi, where FILTER_DWI is key=value pair(s) (default: suffix=dwi extension=.nii.gz scope=raw invalid_filters=allow datatype=dwi)

--filter_T1w Filters (PyBIDS) for T1w, where FILTER_T1W is key=value pair(s) (default: suffix=T1w extension=.nii.gz scope=raw datatype=anat invalid_filters=allow space=None)

CHAPTER

SIX

SNAKEMAKE COMMAND LINE INTERFACE

Snakemake is a Python based language and execution environment for GNU Make-like workflows.

```
usage: snakemake [-h] [--dry-run] [--profile PROFILE]
                 [--cache [RULE [RULE ...]]] [--snakefile FILE] [--cores [N]]
                 [--jobs [N]] [--local-cores N]
                 [--resources [NAME=INT [NAME=INT ...]]]
                 [--set-threads RULE=THREADS [RULE=THREADS ...]]
                 [--max-threads MAX_THREADS]
                 [--set-resources RULE:RESOURCE=VALUE [RULE:RESOURCE=VALUE ...]]
                 [--set-scatter NAME=SCATTERITEMS [NAME=SCATTERITEMS ...]]
                 [--default-resources [NAME=INT [NAME=INT ...]]]
                 [--preemption-default PREEMPTION_DEFAULT]
                 [--preemptible-rules PREEMPTIBLE_RULES [PREEMPTIBLE_RULES ...]]
                 [--config [KEY=VALUE [KEY=VALUE ...]]]
                 [--configfile FILE [FILE ...]]
                 [--envvars VARNAME [VARNAME ...]] [--directory DIR] [--touch]
                 [--keep-going] [--force] [--forceall]
                 [--forcerun [TARGET [TARGET ...]]]
                 [--prioritize TARGET [TARGET ...]]
                 [--batch RULE=BATCH/BATCHES] [--until TARGET [TARGET ...]]
                 [--omit-from TARGET [TARGET ...]] [--rerun-incomplete]
                 [--shadow-prefix DIR] [--scheduler [{ilp,greedy}]]
                 [--wms-monitor [WMS_MONITOR]]
                 [--wms-monitor-arg [NAME=VALUE [NAME=VALUE ...]]]
                 [--scheduler-ilp-solver {PULP_CBC_CMD}]
                 [--scheduler-solver-path SCHEDULER_SOLVER_PATH]
                 [--conda-base-path CONDA_BASE_PATH] [--no-subworkflows]
                 [--groups GROUPS [GROUPS ...]]
                 [--group-components GROUP_COMPONENTS [GROUP_COMPONENTS ...]]
                 [--report [FILE]] [--report-stylesheet CSSFILE]
                 [--draft-notebook TARGET] [--edit-notebook TARGET]
                 [--notebook-listen IP:PORT] [--lint [{text,json}]]
                 [--generate-unit-tests [TESTPATH]] [--containerize]
                 [--export-cwl FILE] [--list] [--list-target-rules] [--dag]
                 [--rulegraph] [--filegraph] [--d3dag] [--summary]
                 [--detailed-summary] [--archive FILE]
                 [--cleanup-metadata FILE [FILE ...]] [--cleanup-shadow]
                 [--skip-script-cleanup] [--unlock] [--list-version-changes]
                 [--list-code-changes] [--list-input-changes]
```

(continues on next page)

(continued from previous page)

```
                [--list-params-changes] [--list-untracked]
                [--delete-all-output] [--delete-temp-output]
                [--bash-completion] [--keep-incomplete] [--drop-metadata]
                [--version] [--reason] [--gui [PORT]] [--printshellcmds]
                [--debug-dag] [--stats FILE] [--nocolor] [--quiet]
                [--print-compilation] [--verbose] [--force-use-threads]
                [--allow-ambiguity] [--nolock] [--ignore-incomplete]
                [--max-inventory-time SECONDS] [--latency-wait SECONDS]
                [--wait-for-files [FILE [FILE ...]]]
                [--wait-for-files-file FILE] [--notemp] [--all-temp]
                [--keep-remote] [--keep-target-files]
                [--allowed-rules ALLOWED_RULES [ALLOWED_RULES ...]]
                [--local-groupid LOCAL_GROUPID]
                [--max-jobs-per-second MAX_JOBS_PER_SECOND]
                [--max-status-checks-per-second MAX_STATUS_CHECKS_PER_SECOND]
                [-T RESTART_TIMES] [--attempt ATTEMPT]
                [--wrapper-prefix WRAPPER_PREFIX]
                [--default-remote-provider {S3,GS,FTP,SFTP,S3Mocked,gfal,gridftp,iRODS,
↪AzBlob,XRootD}]
                [--default-remote-prefix DEFAULT_REMOTE_PREFIX]
                [--no-shared-fs] [--greediness GREEDINESS] [--no-hooks]
                [--overwrite-shellcmd OVERWRITE_SHELLCMD] [--debug]
                [--runtime-profile FILE] [--mode {0,1,2}]
                [--show-failed-logs] [--log-handler-script FILE]
                [--log-service {none,slack,wms}]
                [--cluster CMD | --cluster-sync CMD | --drmaa [ARGS]]
                [--cluster-config FILE] [--immediate-submit]
                [--jobscript SCRIPT] [--jobname NAME]
                [--cluster-status CLUSTER_STATUS]
                [--cluster-cancel CLUSTER_CANCEL]
                [--cluster-cancel-nargs CLUSTER_CANCEL_NARGS]
                [--cluster-sidecar CLUSTER_SIDECAR] [--drmaa-log-dir DIR]
                [--kubernetes [NAMESPACE]] [--container-image IMAGE]
                [--tibanna] [--tibanna-sfn TIBANNA_SFN]
                [--precommand PRECOMMAND]
                [--tibanna-config TIBANNA_CONFIG [TIBANNA_CONFIG ...]]
                [--google-lifesciences]
                [--google-lifesciences-regions GOOGLE_LIFESCIENCES_REGIONS [GOOGLE_
↪LIFESCIENCES_REGIONS ...]]
                [--google-lifesciences-location GOOGLE_LIFESCIENCES_LOCATION]
                [--google-lifesciences-keep-cache] [--tes URL] [--use-conda]
                [--conda-not-block-search-path-envvars] [--list-conda-envs]
                [--conda-prefix DIR] [--conda-cleanup-envs]
                [--conda-cleanup-pkgs [{tarballs,cache}]]
                [--conda-create-envs-only] [--conda-frontend {conda,mamba}]
                [--use-singularity] [--singularity-prefix DIR]
                [--singularity-args ARGS] [--use-envmodules]
                [target [target ...]]
```

6.1 EXECUTION

target Targets to build. May be rules or files.

--dry-run, --dryrun, -n Do not execute anything, and display what would be done. If you have a very large workflow, use –dry-run –quiet to just print a summary of the DAG of jobs.

Default: False

--profile Name of profile to use for configuring Snakemake. Snakemake will search for a corresponding folder in /etc/xdg/snakemake and /home/docs/.config/snakemake. Alternatively, this can be an absolute or relative path. The profile folder has to contain a file 'config.yaml'. This file can be used to set default values for command line options in YAML format. For example, '–cluster qsub' becomes 'cluster: qsub' in the YAML file. Profiles can be obtained from https://github.com/snakemake-profiles. The profile can also be set via the environment variable $SNAKEMAKE_PROFILE.

--cache Store output files of given rules in a central cache given by the environment variable $SNAKEMAKE_OUTPUT_CACHE. Likewise, retrieve output files of the given rules from this cache if they have been created before (by anybody writing to the same cache), instead of actually executing the rules. Output files are identified by hashing all steps, parameters and software stack (conda envs or containers) needed to create them.

--snakefile, -s The workflow definition in form of a snakefile.Usually, you should not need to specify this. By default, Snakemake will search for 'Snakefile', 'snakefile', 'workflow/Snakefile', 'workflow/snakefile' beneath the current working directory, in this order. Only if you definitely want a different layout, you need to use this parameter.

--cores, -c Use at most N CPU cores/jobs in parallel. If N is omitted or 'all', the limit is set to the number of available CPU cores. In case of cluster/cloud execution, this argument sets the number of total cores used over all jobs (made available to rules via workflow.cores).

--jobs, -j Use at most N CPU cluster/cloud jobs in parallel. For local execution this is an alias for –cores.

--local-cores In cluster/cloud mode, use at most N cores of the host machine in parallel (default: number of CPU cores of the host). The cores are used to execute local rules. This option is ignored when not in cluster/cloud mode.

Default: 2

--resources, --res Define additional resources that shall constrain the scheduling analogously to threads (see above). A resource is defined as a name and an integer value. E.g. –resources mem_mb=1000. Rules can use resources by defining the resource keyword, e.g. resources: mem_mb=600. If now two rules require 600 of the resource 'mem_mb' they won't be run in parallel by the scheduler.

--set-threads Overwrite thread usage of rules. This allows to fine-tune workflow parallelization. In particular, this is helpful to target certain cluster nodes by e.g. shifting a rule to use more, or less threads than defined in the workflow. Thereby, THREADS has to be a positive integer, and RULE has to be the name of the rule.

--max-threads Define a global maximum number of threads for any job. This can be helpful in a cluster/cloud setting, when you want to restrict the maximum number of requested

threads without modifying the workflow definition or overwriting them invidiually with –set-threads.

--set-resources Overwrite resource usage of rules. This allows to fine-tune workflow resources. In particular, this is helpful to target certain cluster nodes by e.g. defining a certain partition for a rule, or overriding a temporary directory. Thereby, VALUE has to be a positive integer or a string, RULE has to be the name of the rule, and RESOURCE has to be the name of the resource.

--set-scatter Overwrite number of scatter items of scattergather processes. This allows to fine-tune workflow parallelization. Thereby, SCATTERITEMS has to be a positive integer, and NAME has to be the name of the scattergather process defined via a scattergather directive in the workflow.

--default-resources, --default-res Define default values of resources for rules that do not define their own values. In addition to plain integers, python expressions over inputsize are allowed (e.g. '2*input.size_mb').When specifying this without any arguments (–default-resources), it defines 'mem_mb=max(2*input.size_mb, 1000)' 'disk_mb=max(2*input.size_mb, 1000)' i.e., default disk and mem usage is twice the input file size but at least 1GB.In addition, the system temporary directory (as given by $TMPDIR, $TEMP, or $TMP) is used for the tmpdir resource. The tmpdir resource is automatically used by shell commands, scripts and wrappers to store temporary data (as it is mirrored into $TMPDIR, $TEMP, and $TMP for the executed subprocesses). If this argument is not specified at all, Snakemake just uses the tmpdir resource as outlined above.

--preemption-default A preemptible instance can be requested when using the Google Life Sciences API. If you set a –preemption-default,all rules will be subject to the default. Specifically, this integer is the number of restart attempts that will be made given that the instance is killed unexpectedly. Note that preemptible instances have a maximum running time of 24 hours. If you want to set preemptible instances for only a subset of rules, use –preemptible-rules instead.

--preemptible-rules A preemptible instance can be requested when using the Google Life Sciences API. If you want to use these instances for a subset of your rules, you can use –preemptible-rules and then specify a list of rule and integer pairs, where each integer indicates the number of restarts to use for the rule's instance in the case that the instance is terminated unexpectedly. –preemptible-rules can be used in combination with –preemption-default, and will take priority. Note that preemptible instances have a maximum running time of 24. If you want to apply a consistent number of retries across all your rules, use –premption-default instead. Example: snakemake –preemption-default 10 –preemptible-rules map_reads=3 call_variants=0

--config, -C Set or overwrite values in the workflow config object. The workflow config object is accessible as variable config inside the workflow. Default values can be set by providing a JSON file (see Documentation).

--configfile, --configfiles Specify or overwrite the config file of the workflow (see the docs). Values specified in JSON or YAML format are available in the global config dictionary inside the workflow. Multiple files overwrite each other in the given order. Thereby missing keys in previous config files are extended by following configfiles. Note that this order also includes a config file defined in the workflow definition itself (which will come first).

--envvars Environment variables to pass to cloud jobs.

--directory, -d Specify working directory (relative paths in the snakefile will use this as their

origin).

--touch, -t Touch output files (mark them up to date without really changing them) instead of running their commands. This is used to pretend that the rules were executed, in order to fool future invocations of snakemake. Fails if a file does not yet exist. Note that this will only touch files that would otherwise be recreated by Snakemake (e.g. because their input files are newer). For enforcing a touch, combine this with –force, –forceall, or –forcerun. Note however that you loose the provenance information when the files have been created in realitiy. Hence, this should be used only as a last resort.

Default: False

--keep-going, -k Go on with independent jobs if a job fails.

Default: False

--force, -f Force the execution of the selected target or the first rule regardless of already created output.

Default: False

--forceall, -F Force the execution of the selected (or the first) rule and all rules it is dependent on regardless of already created output.

Default: False

--forcerun, -R Force the re-execution or creation of the given rules or files. Use this option if you changed a rule and want to have all its output in your workflow updated.

--prioritize, -P Tell the scheduler to assign creation of given targets (and all their dependencies) highest priority. (EXPERIMENTAL)

--batch Only create the given BATCH of the input files of the given RULE. This can be used to iteratively run parts of very large workflows. Only the execution plan of the relevant part of the workflow has to be calculated, thereby speeding up DAG computation. It is recommended to provide the most suitable rule for batching when documenting a workflow. It should be some aggregating rule that would be executed only once, and has a large number of input files. For example, it can be a rule that aggregates over samples.

--until, -U Runs the pipeline until it reaches the specified rules or files. Only runs jobs that are dependencies of the specified rule or files, does not run sibling DAGs.

--omit-from, -O Prevent the execution or creation of the given rules or files as well as any rules or files that are downstream of these targets in the DAG. Also runs jobs in sibling DAGs that are independent of the rules or files specified here.

--rerun-incomplete, --ri Re-run all jobs the output of which is recognized as incomplete.

Default: False

--shadow-prefix Specify a directory in which the 'shadow' directory is created. If not supplied, the value is set to the '.snakemake' directory relative to the working directory.

--scheduler Possible choices: ilp, greedy

Specifies if jobs are selected by a greedy algorithm or by solving an ilp. The ilp scheduler aims to reduce runtime and hdd usage by best possible use of resources.

Default: "greedy"

--wms-monitor IP and port of workflow management system to monitor the execution of snakemake (e.g. http://127.0.0.1:5000) Note that if your service requires an authorization token, you must export WMS_MONITOR_TOKEN in the environment.

--wms-monitor-arg If the workflow management service accepts extra arguments, provide. them in key value pairs with –wms-monitor-arg. For example, to run an existing workflow using a wms monitor, you can provide the pair id=12345 and the arguments will be provided to the endpoint to first interact with the workflow

--scheduler-ilp-solver Possible choices: PULP_CBC_CMD

Specifies solver to be utilized when selecting ilp-scheduler.

Default: "COIN_CMD"

--scheduler-solver-path Set the PATH to search for scheduler solver binaries (internal use only).

--conda-base-path Path of conda base installation (home of conda, mamba, activate) (internal use only).

--no-subworkflows, --nosw Do not evaluate or execute subworkflows.

Default: False

6.2 GROUPING

--groups Assign rules to groups (this overwrites any group definitions from the workflow).

--group-components Set the number of connected components a group is allowed to span. By default, this is 1, but this flag allows to extend this. This can be used to run e.g. 3 jobs of the same rule in the same group, although they are not connected. It can be helpful for putting together many small jobs or benefitting of shared memory setups.

6.3 REPORTS

--report Create an HTML report with results and statistics. This can be either a .html file or a .zip file. In the former case, all results are embedded into the .html (this only works for small data). In the latter case, results are stored along with a file report.html in the zip archive. If no filename is given, an embedded report.html is the default.

--report-stylesheet Custom stylesheet to use for report. In particular, this can be used for branding the report with e.g. a custom logo, see docs.

6.4 NOTEBOOKS

--draft-notebook Draft a skeleton notebook for the rule used to generate the given target file. This notebook can then be opened in a jupyter server, exeucted and implemented until ready. After saving, it will automatically be reused in non-interactive mode by Snakemake for subsequent jobs.

--edit-notebook Interactively edit the notebook associated with the rule used to generate the given target file. This will start a local jupyter notebook server. Any changes to the notebook should be saved, and the server has to be stopped by closing the notebook

and hitting the 'Quit' button on the jupyter dashboard. Afterwards, the updated notebook will be automatically stored in the path defined in the rule. If the notebook is not yet present, this will create an empty draft.

--notebook-listen The IP address and PORT the notebook server used for editing the notebook (–edit-notebook) will listen on.

Default: "localhost:8888"

6.5 UTILITIES

--lint Possible choices: text, json

Perform linting on the given workflow. This will print snakemake specific suggestions to improve code quality (work in progress, more lints to be added in the future). If no argument is provided, plain text output is used.

--generate-unit-tests Automatically generate unit tests for each workflow rule. This assumes that all input files of each job are already present. Rules without a job with present input files will be skipped (a warning will be issued). For each rule, one test case will be created in the specified test folder (.tests/unit by default). After successfull execution, tests can be run with 'pytest TESTPATH'.

--containerize Print a Dockerfile that provides an execution environment for the workflow, including all conda environments.

Default: False

--export-cwl Compile workflow to CWL and store it in given FILE.

--list, -l Show available rules in given Snakefile.

Default: False

--list-target-rules, --lt Show available target rules in given Snakefile.

Default: False

--dag Do not execute anything and print the directed acyclic graph of jobs in the dot language. Recommended use on Unix systems: snakemake –dag | dot | displayNote print statements in your Snakefile may interfere with visualization.

Default: False

--rulegraph Do not execute anything and print the dependency graph of rules in the dot language. This will be less crowded than above DAG of jobs, but also show less information. Note that each rule is displayed once, hence the displayed graph will be cyclic if a rule appears in several steps of the workflow. Use this if above option leads to a DAG that is too large. Recommended use on Unix systems: snakemake –rulegraph | dot | displayNote print statements in your Snakefile may interfere with visualization.

Default: False

--filegraph Do not execute anything and print the dependency graph of rules with their input and output files in the dot language. This is an intermediate solution between above DAG of jobs and the rule graph. Note that each rule is displayed once, hence the displayed graph will be cyclic if a rule appears in several steps of the workflow. Use this if above option leads to a DAG that is too large. Recommended

use on Unix systems: snakemake –filegraph | dot | displayNote print statements in your Snakefile may interfere with visualization.

Default: False

--d3dag Print the DAG in D3.js compatible JSON format.

Default: False

--summary, -S Print a summary of all files created by the workflow. The has the following columns: filename, modification time, rule version, status, plan. Thereby rule version contains the versionthe file was created with (see the version keyword of rules), and status denotes whether the file is missing, its input files are newer or if version or implementation of the rule changed since file creation. Finally the last column denotes whether the file will be updated or created during the next workflow execution.

Default: False

--detailed-summary, -D Print a summary of all files created by the workflow. The has the following columns: filename, modification time, rule version, input file(s), shell command, status, plan. Thereby rule version contains the version the file was created with (see the version keyword of rules), and status denotes whether the file is missing, its input files are newer or if version or implementation of the rule changed since file creation. The input file and shell command columns are self explanatory. Finally the last column denotes whether the file will be updated or created during the next workflow execution.

Default: False

--archive Archive the workflow into the given tar archive FILE. The archive will be created such that the workflow can be re-executed on a vanilla system. The function needs conda and git to be installed. It will archive every file that is under git version control. Note that it is best practice to have the Snakefile, config files, and scripts under version control. Hence, they will be included in the archive. Further, it will add input files that are not generated by by the workflow itself and conda environments. Note that symlinks are dereferenced. Supported formats are .tar, .tar.gz, .tar.bz2 and .tar.xz.

--cleanup-metadata, --cm Cleanup the metadata of given files. That means that snakemake removes any tracked version info, and any marks that files are incomplete.

--cleanup-shadow Cleanup old shadow directories which have not been deleted due to failures or power loss.

Default: False

--skip-script-cleanup Don't delete wrapper scripts used for execution

Default: False

--unlock Remove a lock on the working directory.

Default: False

--list-version-changes, --lv List all output files that have been created with a different version (as determined by the version keyword).

Default: False

--list-code-changes, --lc List all output files for which the rule body (run or shell) have changed in the Snakefile.

Default: False

--list-input-changes, --li List all output files for which the defined input files have changed in the Snakefile (e.g. new input files were added in the rule definition or files were renamed). For listing input file modification in the filesystem, use –summary.

Default: False

--list-params-changes, --lp List all output files for which the defined params have changed in the Snakefile.

Default: False

--list-untracked, --lu List all files in the working directory that are not used in the workflow. This can be used e.g. for identifying leftover files. Hidden files and directories are ignored.

Default: False

--delete-all-output Remove all files generated by the workflow. Use together with –dry-run to list files without actually deleting anything. Note that this will not recurse into subworkflows. Write-protected files are not removed. Nevertheless, use with care!

Default: False

--delete-temp-output Remove all temporary files generated by the workflow. Use together with –dry-run to list files without actually deleting anything. Note that this will not recurse into subworkflows.

Default: False

--bash-completion Output code to register bash completion for snakemake. Put the following in your .bashrc (including the accents): *snakemake –bash-completion* or issue it in an open terminal session.

Default: False

--keep-incomplete Do not remove incomplete output files by failed jobs.

Default: False

--drop-metadata Drop metadata file tracking information after job finishes. Provenance-information based reports (e.g. –report and the –list_x_changes functions) will be empty or incomplete.

Default: False

--version, -v show program's version number and exit

6.6 OUTPUT

--reason, -r Print the reason for each executed rule.

Default: False

--gui Serve an HTML based user interface to the given network and port e.g. 168.129.10.15:8000. By default Snakemake is only available in the local network (default port: 8000). To make Snakemake listen to all ip addresses add the special host address 0.0.0.0 to the url (0.0.0.0:8000). This is important if Snakemake is used in a virtualised environment like Docker. If possible, a browser window is opened.

--printshellcmds, -p Print out the shell commands that will be executed.

Default: False

--debug-dag Print candidate and selected jobs (including their wildcards) while inferring DAG. This can help to debug unexpected DAG topology or errors.

Default: False

--stats Write stats about Snakefile execution in JSON format to the given file.

--nocolor Do not use a colored output.

Default: False

--quiet, -q Do not output any progress or rule information.

Default: False

--print-compilation Print the python representation of the workflow.

Default: False

--verbose Print debugging output.

Default: False

6.7 BEHAVIOR

--force-use-threads Force threads rather than processes. Helpful if shared memory (/dev/shm) is full or unavailable.

Default: False

--allow-ambiguity, -a Don't check for ambiguous rules and simply use the first if several can produce the same file. This allows the user to prioritize rules by their order in the snakefile.

Default: False

--nolock Do not lock the working directory

Default: False

--ignore-incomplete, --ii Do not check for incomplete output files.

Default: False

--max-inventory-time Spend at most SECONDS seconds to create a file inventory for the working directory. The inventory vastly speeds up file modification and existence checks when computing which jobs need to be executed. However, creating the inventory itself can be slow, e.g. on network file systems. Hence, we do not spend more than a given amount of time and fall back to individual checks for the rest.

Default: 20

--latency-wait, --output-wait, -w Wait given seconds if an output file of a job is not present after the job finished. This helps if your filesystem suffers from latency (default 5).

Default: 5

--wait-for-files Wait –latency-wait seconds for these files to be present before executing the workflow. This option is used internally to handle filesystem latency in cluster environments.

--wait-for-files-file Same behaviour as –wait-for-files, but file list is stored in file instead of being passed on the commandline. This is useful when the list of files is too long to be passed on the commandline.

--notemp, --nt Ignore temp() declarations. This is useful when running only a part of the workflow, since temp() would lead to deletion of probably needed files by other parts of the workflow.

Default: False

--all-temp Mark all output files as temp files. This can be useful for CI testing, in order to save space.

Default: False

--keep-remote Keep local copies of remote input files.

Default: False

--keep-target-files Do not adjust the paths of given target files relative to the working directory.

Default: False

--allowed-rules Only consider given rules. If omitted, all rules in Snakefile are used. Note that this is intended primarily for internal use and may lead to unexpected results otherwise.

--local-groupid Name for local groupid, meant for internal use only.

Default: "local"

--max-jobs-per-second Maximal number of cluster/drmaa jobs per second, default is 10, fractions allowed.

Default: 10

--max-status-checks-per-second Maximal number of job status checks per second, default is 10, fractions allowed.

Default: 10

-T, --restart-times Number of times to restart failing jobs (defaults to 0).

Default: 0

--attempt Internal use only: define the initial value of the attempt parameter (default: 1).

Default: 1

--wrapper-prefix Prefix for URL created from wrapper directive (default: https://github.com/snakemake/snakemake-wrappers/raw/). Set this to a different URL to use your fork or a local clone of the repository, e.g., use a git URL like 'git+file://path/to/your/local/clone@'.

Default: "https://github.com/snakemake/snakemake-wrappers/raw/"

--default-remote-provider Possible choices: S3, GS, FTP, SFTP, S3Mocked, gfal, gridftp, iRODS, AzBlob, XRootD

Specify default remote provider to be used for all input and output files that don't yet specify one.

--default-remote-prefix Specify prefix for default remote provider. E.g. a bucket name.

Default: ""

--no-shared-fs Do not assume that jobs share a common file system. When this flag is activated, Snakemake will assume that the filesystem on a cluster node is not shared with other nodes. For example, this will lead to downloading remote files on each cluster node separately. Further, it won't take special measures to deal with filesystem latency issues. This option will in most cases only make sense in combination with –default-remote-provider. Further, when using –cluster you will have to also provide –cluster-status. Only activate this if you know what you are doing.

Default: False

--greediness Set the greediness of scheduling. This value between 0 and 1 determines how careful jobs are selected for execution. The default value (1.0) provides the best speed and still acceptable scheduling quality.

--no-hooks Do not invoke onstart, onsuccess or onerror hooks after execution.

Default: False

--overwrite-shellcmd Provide a shell command that shall be executed instead of those given in the workflow. This is for debugging purposes only.

--debug Allow to debug rules with e.g. PDB. This flag allows to set breakpoints in run blocks.

Default: False

--runtime-profile Profile Snakemake and write the output to FILE. This requires yappi to be installed.

--mode Possible choices: 0, 1, 2

Set execution mode of Snakemake (internal use only).

Default: 0

--show-failed-logs Automatically display logs of failed jobs.

Default: False

--log-handler-script Provide a custom script containing a function 'def log_handler(msg):'. Snakemake will call this function for every logging output (given as a dictionary msg)allowing to e.g. send notifications in the form of e.g. slack messages or emails.

--log-service Possible choices: none, slack, wms

Set a specific messaging service for logging output.Snakemake will notify the service on errors and completed execution.Currently slack and workflow management system (wms) are supported.

6.8 CLUSTER

--cluster Execute snakemake rules with the given submit command, e.g. qsub. Snakemake compiles jobs into scripts that are submitted to the cluster with the given command, once all input files for a particular job are present. The submit command can be decorated to make it aware of certain job properties (name, rulename, input, output, params, wildcards, log, threads and dependencies (see the argument below)), e.g.: $ snakemake –cluster 'qsub -pe threaded {threads}'.

--cluster-sync cluster submission command will block, returning the remote exitstatus upon remote termination (for example, this should be usedif the cluster command is 'qsub -sync y' (SGE)

--drmaa Execute snakemake on a cluster accessed via DRMAA, Snakemake compiles jobs into scripts that are submitted to the cluster with the given command, once all input files for a particular job are present. ARGS can be used to specify options of the underlying cluster system, thereby using the job properties name, rulename, input, output, params, wildcards, log, threads and dependencies, e.g.: –drmaa ' -pe threaded {threads}'. Note that ARGS must be given in quotes and with a leading whitespace.

--cluster-config, -u A JSON or YAML file that defines the wildcards used in 'cluster'for specific rules, instead of having them specified in the Snakefile. For example, for rule 'job' you may define: { 'job' : { 'time' : '24:00:00' } } to specify the time for rule 'job'. You can specify more than one file. The configuration files are merged with later values overriding earlier ones. This option is deprecated in favor of using –profile, see docs.

Default: []

--immediate-submit, --is Immediately submit all jobs to the cluster instead of waiting for present input files. This will fail, unless you make the cluster aware of job dependencies, e.g. via: $ snakemake –cluster 'sbatch –dependency {dependencies}. Assuming that your submit script (here sbatch) outputs the generated job id to the first stdout line, {dependencies} will be filled with space separated job ids this job depends on. Does not work for workflows that contain checkpoint rules.

Default: False

--jobscript, --js Provide a custom job script for submission to the cluster. The default script resides as 'jobscript.sh' in the installation directory.

--jobname, --jn Provide a custom name for the jobscript that is submitted to the cluster (see –cluster). NAME is "snakejob.{name}.{jobid}.sh" per default. The wildcard {jobid} has to be present in the name.

Default: "snakejob.{name}.{jobid}.sh"

--cluster-status Status command for cluster execution. This is only considered in combination with the –cluster flag. If provided, Snakemake will use the status command to determine if a job has finished successfully or failed. For this it is necessary that the submit command provided to –cluster returns the cluster job id. Then, the status command will be invoked with the job id. Snakemake expects it to return 'success' if the job was successfull, 'failed' if the job failed and 'running' if the job still runs.

--cluster-cancel Specify a command that allows to stop currently running jobs. The command will be passed a single argument, the job id.

--cluster-cancel-nargs Specify maximal number of job ids to pass to –cluster-cancel command, defaults to 1000.

Default: 1000

--cluster-sidecar Optional command to start a sidecar process during cluster execution. Only active when –cluster is given as well.

--drmaa-log-dir Specify a directory in which stdout and stderr files of DRMAA jobs will be written. The value may be given as a relative path, in which case Snakemake will use the current invocation directory as the origin. If given, this will override any given

'-o' and/or '-e' native specification. If not given, all DRMAA stdout and stderr files are written to the current working directory.

6.9 KUBERNETES

--kubernetes Execute workflow in a kubernetes cluster (in the cloud). NAMESPACE is the namespace you want to use for your job (if nothing specified: 'default'). Usually, this requires –default-remote-provider and –default-remote-prefix to be set to a S3 or GS bucket where your . data shall be stored. It is further advisable to activate conda integration via –use-conda.

--container-image Docker image to use, e.g., when submitting jobs to kubernetes Defaults to 'https://hub.docker.com/r/snakemake/snakemake', tagged with the same version as the currently running Snakemake instance. Note that overwriting this value is up to your responsibility. Any used image has to contain a working snakemake installation that is compatible with (or ideally the same as) the currently running version.

6.10 TIBANNA

--tibanna Execute workflow on AWS cloud using Tibanna. This requires –default-remote-prefix to be set to S3 bucket name and prefix (e.g. 'bucketname/subdirectory') where input is already stored and output will be sent to. Using –tibanna implies –default-resources is set as default. Optionally, use –precommand to specify any preparation command to run before snakemake command on the cloud (inside snakemake container on Tibanna VM). Also, –use-conda, –use-singularity, –config, –configfile are supported and will be carried over.

Default: False

--tibanna-sfn Name of Tibanna Unicorn step function (e.g. tibanna_unicorn_monty).This works as serverless scheduler/resource allocator and must be deployed first using tibanna cli. (e.g. tibanna deploy_unicorn –usergroup=monty –buckets=bucketname)

--precommand Any command to execute before snakemake command on AWS cloud such as wget, git clone, unzip, etc. This is used with –tibanna.Do not include input/output download/upload commands - file transfer between S3 bucket and the run environment (container) is automatically handled by Tibanna.

--tibanna-config Additional tibanna config e.g. –tibanna-config spot_instance=true subnet=<subnet_id> security group=<security_group_id>

6.11 GOOGLE_LIFE_SCIENCE

--google-lifesciences Execute workflow on Google Cloud cloud using the Google Life. Science API. This requires default application credentials (json) to be created and export to the environment to use Google Cloud Storage, Compute Engine, and Life Sciences. The credential file should be exported as GOOGLE_APPLICATION_CREDENTIALS for snakemake to discover. Also, –use-conda, –use-singularity, –config, –configfile are supported and will be carried over.

Default: False

--google-lifesciences-regions Specify one or more valid instance regions (defaults to US)

Default: ['us-east1', 'us-west1', 'us-central1']

--google-lifesciences-location The Life Sciences API service used to schedule the jobs. E.g., us-centra1 (Iowa) and europe-west2 (London) Watch the terminal output to see all options found to be available. If not specified, defaults to the first found with a matching prefix from regions specified with –google-lifesciences-regions.

--google-lifesciences-keep-cache Cache workflows in your Google Cloud Storage Bucket specified by –default-remote-prefix/{source}/{cache}. Each workflow working directory is compressed to a .tar.gz, named by the hash of the contents, and kept in Google Cloud Storage. By default, the caches are deleted at the shutdown step of the workflow.

Default: False

6.12 TES

--tes Send workflow tasks to GA4GH TES server specified by url.

6.13 CONDA

--use-conda If defined in the rule, run job in a conda environment. If this flag is not set, the conda directive is ignored.

Default: False

--conda-not-block-search-path-envvars Do not block environment variables that modify the search path (R_LIBS, PYTHONPATH, PERL5LIB, PERLLIB) when using conda environments.

Default: False

--list-conda-envs List all conda environments and their location on disk.

Default: False

--conda-prefix Specify a directory in which the 'conda' and 'conda-archive' directories are created. These are used to store conda environments and their archives, respectively. If not supplied, the value is set to the '.snakemake' directory relative to the invocation directory. If supplied, the *–use-conda* flag must also be set. The value may be given as a relative path, which will be extrapolated to the invocation directory, or as an absolute path. The value can also be provided via the environment variable $SNAKEMAKE_CONDA_PREFIX.

--conda-cleanup-envs Cleanup unused conda environments.

Default: False

--conda-cleanup-pkgs Possible choices: tarballs, cache

Cleanup conda packages after creating environments. In case of 'tarballs' mode, will clean up all downloaded package tarballs. In case of 'cache' mode, will additionally clean up unused package caches. If mode is omitted, will default to only cleaning up the tarballs.

--conda-create-envs-only If specified, only creates the job-specific conda environments then exits. The *–use-conda* flag must also be set.

Default: False

--conda-frontend Possible choices: conda, mamba

Choose the conda frontend for installing environments. Mamba is much faster and highly recommended.

Default: “mamba”

6.14 SINGULARITY

--use-singularity If defined in the rule, run job within a singularity container. If this flag is not set, the singularity directive is ignored.

Default: False

--singularity-prefix Specify a directory in which singularity images will be stored.If not supplied, the value is set to the ‘.snakemake’ directory relative to the invocation directory. If supplied, the *–use-singularity* flag must also be set. The value may be given as a relative path, which will be extrapolated to the invocation directory, or as an absolute path.

--singularity-args Pass additional args to singularity.

Default: “”

6.15 ENVIRONMENT MODULES

--use-envmodules If defined in the rule, run job within the given environment modules, loaded in the given order. This can be combined with –use-conda and –use-singularity, which will then be only used as a fallback for rules which don’t define environment modules.

Default: False