autosubmit Documentation

Release 3.0

Domingo Manubens - Javier Vegas

May 04, 2015

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 What is Autosubmit ?

Autosubmit is a python-based tool to create, manage and monitor experiments by using Computing Clusters, HPC's and Supercomputers remotely via ssh. It has support for experiments running in more than one HPC and for different workflow configurations.

Autosubmit is currently used at IC3 to run EC-Earth and NEMO models and at Barcelona Supercomputing Centre (BSC) to run NMMB air quality model.

Autosubmit has been used to manage models running at supercomputers in IC3, BSC, ECMWF, EPCC, PDC and OLCF.

Autosubmit 3.0 version is now available via *PyPi* package under the terms of *GNU General Public License*.

1.2 Why is Autosubmit needed ?

Autosubmit is the only existing tool that satisfies the following requirements from the weather and climate community:

- *Automatisation*: Job submission to machines and dependencies between jobs are managed by Autosubmit. No user intervention is needed.
- *Data provenance*: Assigns unique identifiers for each experiment and stores information about model version, experiment configuration and computing facilities used in the whole process.
- *Failure tolerance*: Automatic retrials and ability to rerun chunks in case of corrupted or missing data.
- *Resource management*: Autosubmit manages supercomputer particularities, allowing users to run their experiments in the available machine without having to adapt the code. Autosubmit also allows to submit tasks from the same experiment to different platforms.

1.3 How does Autosubmit work ?

You can find help about how to use autosubmit and a list of available commands, just executing:

```
autosubmit -h
```

Execute autosubmit <command> -h for detailed help for each command:

```
autosubmit expid -h
```

1.3.1 Experiment creation

To create a new experiment, run the command:

```
autosubmit expid -H HPCname -d Description
```

HPCname is the name of the main HPC platform for the experiment: it will be the default platform for the tasks. *Description* is a brief experiment description.

This command assigns a unique four character identifier (`xxxx`, names starting from a letter, the other three characters) to the experiment and creates a new folder in experiments repository with structure shown in Figure 1.1.

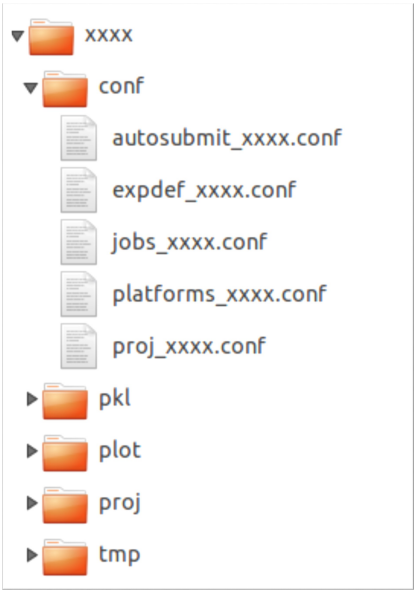

Figure 1.1: Example of an experiment directory tree.

1.3.2 Experiment configuration

To configure the experiment, edit `expdef_xxxx.conf`, `jobs_xxxx.conf` and `platforms_xxxx.conf` in the `conf` folder of the experiment (see contents in Figure 1.2).

After that, you are expected to run the command:

```
autosubmit create xxxx
```

This command creates the experiment project in the `proj` folder. The experiment project contains the scripts specified in `jobs_xxxx.conf` and a copy of model source code and data specified in `expdef_xxxx.conf`.

1.3.3 Experiment run

To run the experiment, just execute the command:

```
autosubmit run xxxx
```

Autosubmit will start submitting jobs to the relevant platforms (both HPC and supporting computers) by using the scripts specified in `jobs_xxxx.conf`. Autosubmit will substitute variables present on scripts where handlers ap-

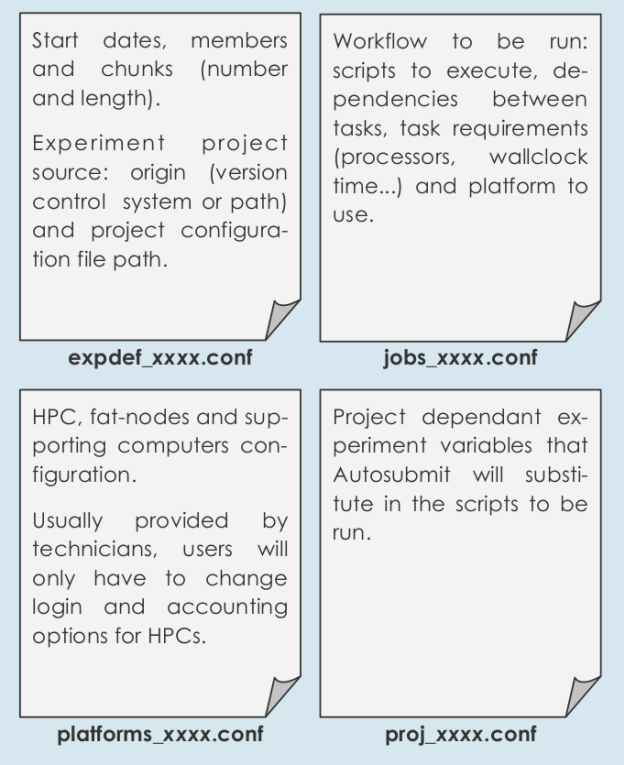

Figure 1.2: Configuration files content

pear in *%variable_name%* format. Autosubmit provides variables for *current chunk*, *start date*, *member*, *computer configuration* and more, and also will replace variables form `proj_xxxx.conf`.

To monitor the status of the experiment, the command:

```
autosubmit monitor xxxx
```

is available. This will plot the workflow of the experiment and the current status.

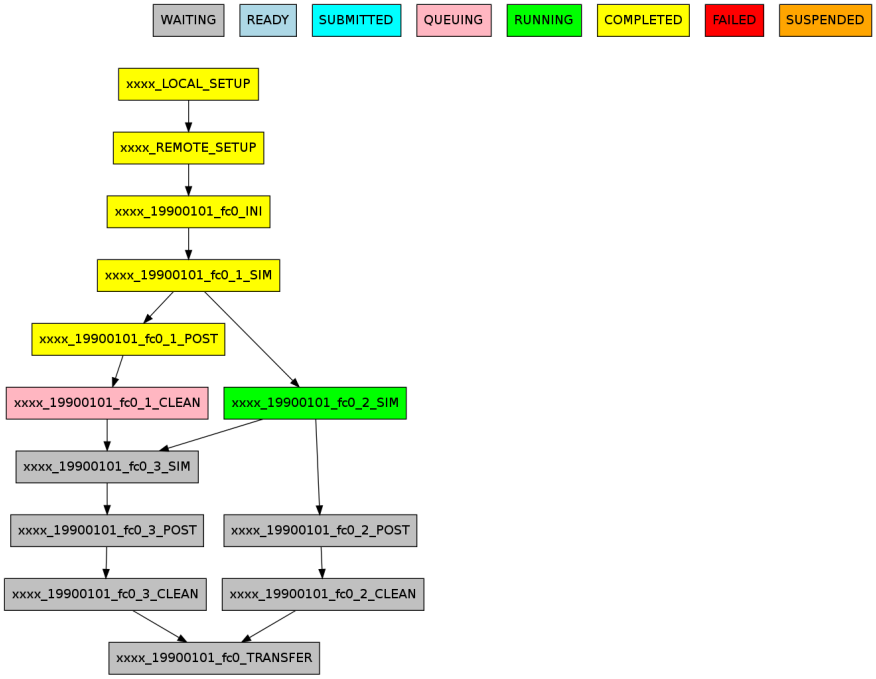

Figure 1.3: Example of monitoring plot for EC-Earth run with Autosubmit for 1 start date, 1 member and 3 chunks.

CHAPTER

TWO

INSTALLATION

2.1 How to install

The Autosubmit code is maintained in *PyPi*, the main source for python packages.

- Pre-requisties: These packages (bash, python2, sqlite3, git-scm > 1.8.2, subversion) must be available at local host machine. These packages (argparse, dateutil, pyparsing, numpy, pydotplus, matplotlib, paramiko) must be available for python runtime.

Important: The host machine has to be able to access HPC's/Clusters via password-less ssh.

To install autosubmit just execute:

```
pip install autosubmit
```

or download, unpack and:

```
python setup.py install
```

2.2 How to configure

After installation, you have to configure database and path for Autosubmit. It can be done at host, user or local level (by default at host level). If it does not exist, create a repository for experiments: Say for example `/cfu/autosubmit`

Then follow the confiugre instructions after executing:

```
autosubmit configure -u
```

and introduce path to experiment storage and database. Folders must exit.

For installing the database for Autosubmit on the configured folder, when no database is created on the given path, execute:

```
autosubmit install
```

Danger: Be careful ! autosubmit install will create a blank database.

Now you are ready to use Autosubmit !

CHAPTER

THREE

USAGE

3.1 Command list

-expid	Create a new experiment
-create	Create specified experiment workflow
-check	Check configuration for specified experiment
-run	Run specified experiment
-test	Test experiment
-monitor	Plot specified experiment
-stats	Plot statistics for specified experiment
-setstatus	Sets job status for an experiment
-recovery	Recover specified experiment
-clean	Clean specified experiment
-refresh	Refresh project directory for an experiment
-delete	Delete specified experiment
-configure	Configure database and path for autosubmit
-install	Install database for Autosubmit on the configured folder

3.2 How to create an experiment

To create a new experiment, just run the command:

```
autosubmit expid -H HPCname -d Description
```

HPCname is the name of the main HPC platform for the experiment: it will be the default platform for the tasks. *Description* is a brief experiment description.

Options:

```
usage: autosubmit expid [-h] [-y COPY | -dm] -H HPC -d DESCRIPTION

  -h, --help            show this help message and exit
  -y COPY, --copy COPY  makes a copy of the specified experiment
  -dm, --dummy          creates a new experiment with default values, usually for testing
  -H HPC, --HPC HPC     specifies the HPC to use for the experiment
```

```
  -d DESCRIPTION, --description DESCRIPTION
                        sets a description for the experiment to store in the database.
```

Example:

```
autosubmit expid --HPC ithaca --description "experiment is about..."
```

3.3 How to create a copy of an experiment

This option makes a copy of an existing experiment. It registrates a new unique identifier and copies all configuration files in the new experiment folder:

```
autosubmit expid -H HPCname -y COPY -d Description
```

HPCname is the name of the main HPC platform for the experiment: it will be the default platform for the tasks. *COPY* is the experiment identifier to copy from. *Description* is a brief experiment description.

Example:

```
autosubmit expid -H ithaca -y cxxx -d "experiment is about..."
```

Warning: You can only copy experiments created with Autosubmit 3.0 or above.

3.4 How to create a dummy experiment

This command creates a new experiment with default values, useful for testing:

```
autosubmit expid -H HPCname -dm -d Description
```

HPCname is the name of the main HPC platform for the experiment: it will be the default platform for the tasks. *Description* is a brief experiment description.

Example:

```
autosubmit expid -H ithaca -dm "experiment is about..."
```

3.5 How to configure the experiment

Edit `expdef_cxxx.conf`, `jobs_cxxx.conf` and `platforms_cxxx.conf` in the `conf` folder of the experiment.

***expdef_cxxx.conf* contains:**

- Start dates, members and chunks (number and length).
- Experiment project source: origin (version control system or path)
- Project configuration file path.

***jobs_cxxx.conf* contains the workflow to be run:**

- Scripts to execute.
- Dependencies between tasks.

- Task requirements (processors, wallclock time...).
- Platform to use.

***platforms_cxxx.conf* contains:**

- HPC, fat-nodes and supporting computers configuration.

Note: *platforms_cxxx.conf* is usually provided by technicians, users will only have to change login and accounting options for HPCs.

You may want to configure Autosubmit parameters for the experiment. Just edit `autosubmit_cxxx.conf`.

***autosubmit_cxxx.conf* contains:**

- Maximum number of jobs to be running at the same time at the HPC.
- Time (seconds) between connections to the HPC queue scheduler to poll already submitted jobs status.
- Number of retrials if a job fails.

Then, Autosubmit *create* command uses the `expdef_cxxx.conf` and generates the experiment: After editing the files you can proceed to the experiment workflow creation. Experiment workflow, which contains all the jobs and its dependencies, will be saved as a *pkl* file:

```
autosubmit create EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit create [-h] [-np] expid

  expid          experiment identifier

  -h, --help     show this help message and exit
  -np, --noplot  omit plot
```

Example:

```
autosubmit create cxxx
```

More info on pickle can be found at http://docs.python.org/library/pickle.html

3.6 How to check the experiment configuration

To check the configuration of the experiment, use the command:

```
autosubmit check EXPID
```

EXPID is the experiment identifier.

It checks experiment configuration and warns about any detected error or inconsistency.

Options:

```
usage: autosubmit check [-h] expid

  expid                 experiment identifier

  -h, --help            show this help message and exit
```

Example:

```
autosubmit check cxxx
```

3.7 How to run the experiment

Launch Autosubmit with the command:

```
autosubmit run EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit run [-h] expid

  expid       experiment identifier

  -h, --help  show this help message and exit
```

Example:

```
autosubmit run cxxx
```

Hint: It is recommended to launch it in background and with `nohup` (continue running although the user who launched the process logs out).

Example:

```
nohup autosubmit run cxxx &
```

Important: Before launching Autosubmit check password-less ssh is feasible (*HPCName* is the hostname):

```
ssh HPCName
```

More info on password-less ssh can be found at: http://www.linuxproblem.org/art_9.html

Caution: After launching Autosubmit, one must be aware of login expiry limit and policy (if applicable for any HPC) and renew the login access accordingly (by using token/key etc) before expiry.

3.8 How to test the experiment

This method is to conduct a test for a given experiment. It creates a new experiment for a given experiment with a given number of chunks with a random start date and a random member to be run on a random HPC.

To test the experiment, use the command:

```
autosubmit test CHUNKS EXPID
```

EXPID is the experiment identifier. *CHUNKS* is the number of chunks to run in the test.

Options:

```
usage: autosubmit test [-h] -c CHUNKS [-m MEMBER] [-s STARDATE] [-H HPC] [-b BRANCH] expid

expid                 experiment identifier

 -h, --help            show this help message and exit
 -c CHUNKS, --chunks CHUNKS
                       chunks to run
 -m MEMBER, --member MEMBER
                       member to run
 -s STARDATE, --stardate STARDATE
                       stardate to run
 -H HPC, --HPC HPC     HPC to run experiment on it
 -b BRANCH, --branch BRANCH
                       branch from git to run (or revision from subversion)
```

Example:

```
autosubmit test -c 1 -s 19801101 -m fc0 -H ithaca -b develop cxxx
```

3.9 How to monitor the experiment

To monitor the status of the experiment, use the command:

```
autosubmit monitor EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit monitor [-h] [-o {pdf,png,ps,svg}] expid

  expid                 experiment identifier

  -h, --help            show this help message and exit
  -o {pdf,png,ps,svg}, --output {pdf,png,ps,svg}
                        type of output for generated plot
```

Example:

```
autosubmit monitor cxxx
```

The location where user can find the generated plots with date and timestamp can be found below:

```
<experiments_directory>/cxxx/plot/cxxx_<date>_<time>.pdf
```

3.10 How to monitor job statistics

The following command could be adopted to generate the plots for visualizing the jobs statistics of the experiment at any instance:

```
autosubmit stats EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit stats [-h] [-o {pdf,png,ps,svg}] expid

  expid                 experiment identifier

  -h, --help            show this help message and exit
  -o {pdf,png,ps,svg}, --output {pdf,png,ps,svg}
                        type of output for generated plot
```

Example:

```
autosubmit stats cxxx
```

The location where user can find the generated plots with date and timestamp can be found below:

```
<experiments_directory>/cxxx/plot/cxxx_statistics_<date>_<time>.pdf
```

3.11 How to stop the experiment

You can stop Autosubmit by sending a signal to the process. To get the process identifier (PID) you can use the ps command on a shell interpreter/terminal.

```
ps -ef | grep autosubmit
dmanubens  22835     1  1 May04 ?        00:45:35 autosubmit run cxxy
dmanubens  25783     1  1 May04 ?        00:42:25 autosubmit run cxxx
```

To send a signal to a process you can use kill also on a terminal.

To stop immediately experiment cxxx:

```
kill -9 22835
```

Important: In case you want to restart the experiment, you must follow the *How to restart the experiment* procedure, explained below, in order to properly resynchronize all completed jobs.

3.12 How to restart the experiment

This procedure allows you to restart an experiment.

You must execute:

```
autosubmit recovery EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit recovery [-h] [-all] [-s] expid

    expid       experiment identifier

    -h, --help  show this help message and exit
    -all        Get all completed files to synchronize pkl
    -s, --save  Save changes to disk
```

Example:

```
autosubmit recovery cxxx -s
```

Hint: When we are satisfied with the results we can use the parameter -s, which will save the change to the pkl file and rename the update file.

The -all flag is used to synchronize all jobs of our experiment locally with the information available on the remote platform (i.e.: download the COMPLETED files we may not have). In case new files are found, the `pkl` will be updated.

Example:

```
autosubmit recovery cxxx -all -s
```

3.13 How to rerun a part of the experiment

This procedure allows you to create automatically a new pickle with a list of jobs of the experiment to rerun.

Using the `expdef_<expid>.conf` the `create` command will generate the rerun if the variable RERUN is set to TRUE and a CHUNKLIST is provided.

```
autosubmit create cxxx
```

It will read the list of chunks specified in the CHUNKLIST and will generate a new plot.

Note: The results are saved in the new pkl `rerun_job_list.pkl`.

Example:

```
vi <experiments_directory>/cxxx/conf/expdef_cxxx.conf
```

```
    [rerun]
    # Is a rerun or not? [Default: Do set FALSE]. BOOLEAN = TRUE, FALSE
    RERUN = TRUE
    # If RERUN = TRUE then supply the list of chunks to rerun
    # LIST = "[ 19601101 [ fc0 [1 2 3 4] fc1 [1] ] 19651101 [ fc0 [16-30] ] ]"
    CHUNKLIST = [ 19601101 [ fc1 [1] ]
```

Then we are able to start again Autosubmit for the rerun of cxxx 19601101, chunk 1, member 1:

```
nohup autosubmit run cxxx &
```

3.14 How to clean the experiment

This procedure allows you to save space after finalising an experiment. You must execute:

```
autosubmit clean EXPID
```

Options:

```
usage: autosubmit clean [-h] [-pr] [-p] [-s] expid

  expid           experiment identifier

  -h, --help      show this help message and exit
```

```
  -pr, --project  clean project
  -p, --plot      clean plot, only 2 last will remain
  -s, --stats     clean stats, only last will remain
```

- The -p and -s flag are used to clean our experiment `plot` folder to save disk space. Only the two latest plots will be kept. Older plots will be removed.

Example:

```
autosubmit clean cxxx -p
```

- The -pr flag is used to clean our experiment `proj` locally in order to save space (it could be particullary big).

Caution: Bear in mind that if you have not synchronized your experiment project folder with the information available on the remote repository (i.e.: commit and push any changes we may have), or in case new files are found, the clean procedure will be failing although you specify the -pr option.

Example:

```
autosubmit clean cxxx -pr
```

A bare copy (which occupies less space on disk) will be automatically made.

Hint: That bare clone can be always reconverted in a working clone if we want to run again the experiment by using `git clone bare_clone original_clone`.

Note: In addition, every time you run this command with -pr option, it will check the commit unique identifier for local working tree existing on the `proj` directory. In case that commit identifier exists, clean will register it to the `expdef_cxxx.conf` file.

3.15 How to refresh the experiment project

To refresh the project directory of the experiment, use the command:

```
autosubmit refresh EXPID
```

EXPID is the experiment identifier.

It checks experiment configuration and copy code from original repository to project directory.

Warning: DO NOT USE THIS COMMAND IF YOU ARE NOT SURE ! Project directory will be overwritten and you may loose local changes.

Options:

```
usage: autosubmit refresh [-h] expid

  expid                 experiment identifier

  -h, --help            show this help message and exit
```

Example:

```
autosubmit refresh cxxx
```

3.16 How to delete the experiment

To delete the experiment, use the command:

```
autosubmit delete EXPID
```

EXPID is the experiment identifier.

> **Warning:** DO NOT USE THIS COMMAND IF YOU ARE NOT SURE ! It deletes the experiment from database and experiment's folder.

Options:

```
usage: autosubmit delete [-h] [-f] expid

  expid                 experiment identifier

  -h, --help            show this help message and exit
  -f, --force  deletes experiment without confirmation
```

Example:

```
autosubmit delete cxxx
```

> **Warning:** Be careful ! force option does not ask for your confirmation.

CHAPTER

FOUR

TROUBLESHOOTING

4.1 How to change the job status stopping autosubmit

This procedure allows you to modify the status of your jobs.

Warning: Beware that Autosubmit must be stopped to use `setstatus`. Otherwise a running instance of Autosubmit, at some point, will overwritte any change you may have done.

You must execute:

```
autosubmit setstatus EXPID -f fs STATUS_ORIGINAL -t STATUS_FINAL -s
```

EXPID is the experiment identifier. *STATUS_ORIGINAL* is the original status to filter by the list of jobs. *STATUS_FINAL* the desired target status.

Options:

```
usage: autosubmit setstatus [-h] [-s] -t
    {READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN,QUEUING,RUNNING}
    (-l LIST
    | -fc FILTER_CHUNKS
    | -fs {Any,READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN}
    | -ft FILTER_TYPE)
    expid

expid                 experiment identifier
-h, --help            show this help message and exit
-s, --save            Save changes to disk
-t {READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN},
            --status_final {READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN}
                        Supply the target status
    -l LIST, --list LIST  Supply the list of job names to be changed. Default =
                        "Any". LIST = "cxxx_20101101_fc3_21_sim
                        cxxx_20111101_fc4_26_sim"
    -fc FILTER_CHUNKS, --filter_chunks FILTER_CHUNKS
                        Supply the list of chunks to change the status.
                        Default = "Any". LIST = "[ 19601101 [ fc0 [1 2 3 4]
                        fc1 [1] ] 19651101 [ fc0 [16-30] ] ]"
    -fs {Any,READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN},
            --filter_status {Any,READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN}
                        Select the original status to filter the list of jobs
    -ft FILTER_TYPE, --filter_type FILTER_TYPE
                        Select the job type to filter the list of jobs
```

Examples:

```
autosubmit setstatus cxxx -l "cxxx_20101101_fc3_21_sim cxxx_20111101_fc4_26_sim" -t READY -s
autosubmit setstatus cxxx -f -fc [ 19601101 [ fc1 [1] ] ] -t READY -s
autosubmit setstatus cxxx -f -fs FAILED -t READY -s
autosubmit setstatus cxxx -f -ft TRANSFER -t SUSPENDED -s
```

This script has three mandatory arguments.

The first where you must specify the experiment id, the -t where you must specify the target status of the jobs you want to change to:

```
{READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN}
```

The third argument has two alternatives, the -l and -f; with those we can apply a filter for the jobs we want to change:

The -l flag recieves a list of jobnames separated by blank spaces: e.g.:

```
"cxxx_20101101_fc3_21_sim cxxx_20111101_fc4_26_sim"
```

If we supply the key word "Any", all jobs will be changed to the target status.

The -f flag can be used in three modes: the chunk filter, the status filter or the type filter.

- **The variable -fc should be a list of individual chunks or ranges of chunks in the following format:**

  ```
  [ 19601101 [ fc0 [1 2 3 4] fc1 [1] ] 19651101 [ fc0 [16-30] ] ]
  ```

- **The variable -fs can be one of the following status for job:**

  ```
  {Any,READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN}
  ```

- The variable -ft can be one of the defined types of job.

Hint: When we are satisfied with the results we can use the parameter -s, which will save the change to the pkl file.

4.2 How to change the job status without stopping autosubmit

This procedure allows you to modify the status of your jobs without having to stop Autosubmit.

You must create a file in `<experiments_directory>/<expid>/pkl/` named:

```
updated_list_<expid>.txt
```

Format:

This file should have two columns: the first one has to be the job_name and the second one the status.

Options:

```
READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN
```

Example:

```
vi updated_list_cxxx.txt
```

```
    cxxx_20101101_fc3_21_sim    READY
    cxxx_20111101_fc4_26_sim    READY
```

If Autosubmit finds the above file, it will process it. You can check that the processing was OK at a given date and time, if you see that the file name has changed to:

```
update_list_<expid>_<date>_<time>.txt
```

Note: A running instance of Autosubmit will check the existance of avobe file after checking already submitted jobs. It may take some time, depending on the setting `SAFETYSLEEPTIME`.

Warning: Keep in mind that autosubmit reads the file automatically so it is suggested to create the file in another location like `/tmp` or `/var/tmp` and then copy/move it to the `pkl` folder. Alternativelly you can create the file with a different name an rename it when you have finished.

CHAPTER

FIVE

TUTORIAL

5.1 Quick start guide

5.1.1 First Step: Experiment creation

To create a new experiment, run the command:

```
autosubmit expid -H HPCname -d Description
```

HPCname is the name of the main HPC platform for the experiment: it will be the default platform for the tasks. *Description* is a brief experiment description.

This command assigns a unique four character identifier (`xxxx`, names starting from a letter, the other three characters) to the experiment and creates a new folder in experiments repository.

Examples:

```
autosubmit expid --HPC ithaca --description "experiment is about..."
```

Caution: The *HPCname*, e.g. ithaca, must be defined in the platforms configuration. See next section *Second Step: Experiment configuration*.

```
autosubmit expid --copy a000 --HPC ithaca -d "experiment is about..."
```

Warning: You can only copy experiments created with Autosubmit 3.0 or above.

5.1.2 Second Step: Experiment configuration

To configure the experiment, edit `expdef_cxxx.conf`, `jobs_cxxx.conf` and `platforms_cxxx.conf` in the `conf` folder of the experiment.

***expdef_cxxx.conf* contains:**

- Start dates, members and chunks (number and length).
- Experiment project source: origin (version control system or path)
- Project configuration file path.

***jobs_cxxx.conf* contains the workflow to be run:**

- Scripts to execute.
- Dependencies between tasks.

- Task requirements (processors, wallclock time...).
- Platform to use.

***platforms_cxxx.conf* contains:**

- HPC, fat-nodes and supporting computers configuration.

Note: *platforms_cxxx.conf* is usually provided by technicians, users will only have to change login and accounting options for HPCs.

Examples:

```
vi <experiments_directory>/cxxx/conf/expdef_cxxx.conf

        # Supply the list of members. LIST = fc0 fc1 fc2 fc3 fc4
        MEMBERS = fc0 fc1 fc2

vi <experiments_directory>/cxxx/conf/jobs_cxxx.conf

        [JOBNAME]
        # Script to execute. If not specified, job will be omited from workflow.
        # Path relative to the project directory
        FILE = scripts/run.sh
```

You may want to configure Autosubmit parameters for the experiment. Just edit `autosubmit_cxxx.conf`.

***expdef_cxxx.conf* contains:**

- Maximum number of jobs to be waiting in the HPC queue.
- Maximum number of jobs to be running at the same time at the HPC.
- Time (seconds) between connections to the HPC queue scheduler to poll already submitted jobs status.
- Number of retrials if a job fails.

Example:

```
vi <experiments_directory>/cxxx/conf/autosubmit_cxxx.conf

    # Maximum number of jobs to be running at the same time at the HPC
    # Default = 6
    TOTALJOBS = 10
```

Then, Autosubmit *create* command uses the `expdef_cxxx.conf` and generates the experiment:

```
autosubmit create cxxx
```

cxxx is the name of the experiment.

In the process of creating the new experiment a plot has been created.

It can be found in `<experiments_directory>/cxxx/plot/`

5.1.3 Third Step: Experiment run

After filling the experiment configuration and create, user can go into `proj` which has a copy of the model.

A short reference on how to prepare the experiment project is detailed in the following section of this documentation:

Developing a project

The experiment project contains the scripts specified in `jobs_xxxx.conf` and a copy of model source code and data specified in `expdef_xxxx.conf`.

To configure experiment project parameters for the experiment, edit `proj_cxxx.conf`.

***proj_cxxx.conf* contains:**

- The project dependant experiment variables that Autosubmit will substitute in the scripts to be run.

Example:

```
vi <experiments_directory>/cxxx/conf/prof_cxxx.conf

        # Number of scales for SPPT [Default: set 3]. NUMERIC = 1, 2, 3
        NS_SPPT = 2
```

Launch Autosubmit *run* in background and with `nohup` (continue running although the user who launched the process logs out).

```
nohup autosubmit run cxxx &
```

5.1.4 Fourth Step: Experiment monitor

The following procedure could be adopted to generate the plots for visualizing the status of the experiment at any instance. With this command we can generate new plots to check which is the status of the experiment. Different job status are represented with different colors.

```
autosubmit monitor  cxxx
```

The location where user can find the generated plots with date and timestamp can be found below:

```
<experiments_directory>/cxxx/plot/cxxx_<date>_<time>.pdf
```

CHAPTER

SIX

DEVELOPING A PROJECT

Autosubmit is used at IC3 to run EC-Earth. To do that, a git repository has been created that contains the model source code and the scripts used to run the tasks.

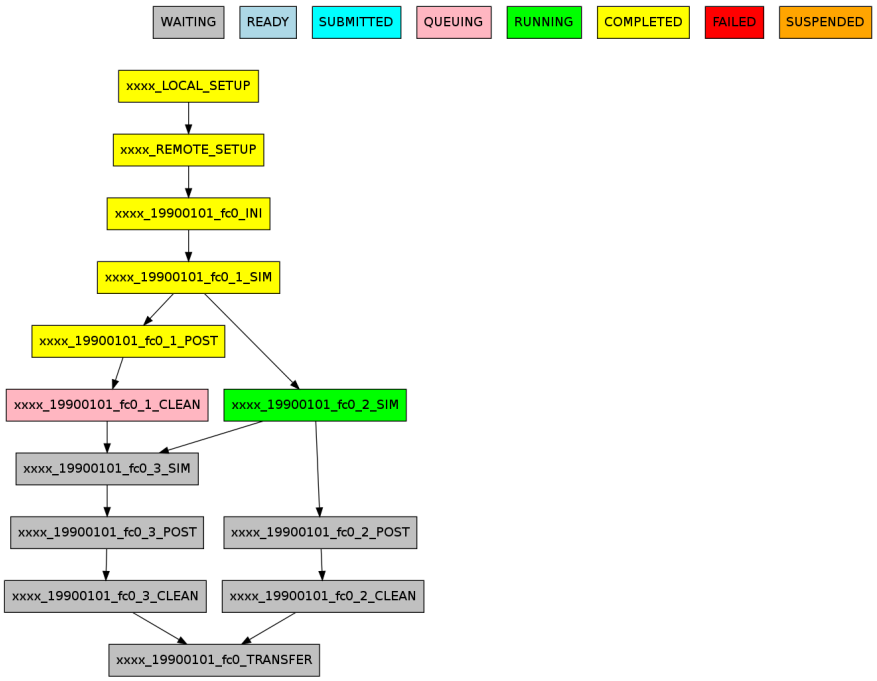

Figure 6.1: Example of monitoring plot for EC-Earth run with Autosubmit for 1 start date, 1 member and 3 chunks.

The workflow is defined using seven job types, as shown in the figure above. These job types are:

- Local_setup: prepares a patch for model changes and copies it to HPC.
- Remote_setup: creates a model copy and applies the patch to it.
- Ini: prepares model to start the simulation of one member.
- Sim: runs a simulation chunk (usually 1 to 3 months).
- Post: post-process outputs for one simulation chunk.
- Clean: removes unnecessary outputs from the simulated chunk.
- Transfer: transfers post-processed outputs to definitive storage.

Since Autosubmit 2.2 the user can select the desired source repository for the experiment project and using a given concrete branch is possible. This introduce a better version control system for project and more options to create new experiments based on different developments by the user. The different projects contain the shell script to run, for each job type (local setup, remote setup, ini, sim, post, clean and transfer) that are platform independent. Additionally the user can modify the sources under proj folder. The executable scripts are created at runtime so the modifications on the sources can be done on the fly.

Important: For a complete reference on how to develop an EC-Earth project, please have a look in the following wiki page: http://ic3.cat/wikicfu/index.php/Models

---

CHAPTER

SEVEN

MODULE DOCUMENTATION

7.1 autosubmit

Main module for autosubmit. Only contains an interface class to all functionality implemented on autosubmit

class `autosubmit.autosubmit.`**`Autosubmit`**

Interface class for autosubmit.

static `check`(*expid*)

Checks experiment configuration and warns about any detected error or inconsistency.

Parameters **expid** (*str*) – experiment identifier:

static `clean`(*expid*, *project*, *plot*, *stats*)

Clean experiment's directory to save storage space. It removes project directory and outdated plots or stats.

Parameters

- **expid** (*str*) – identifier of experiment to clean
- **project** (*bool*) – set True to delete project directory
- **plot** (*bool*) – set True to delete outdated plots
- **stats** (*bool*) – set True to delete outdated stats

static `configure`(*database_path*, *database_filename*, *local_root_path*, *platforms_conf_path*, *jobs_conf_path*, *machine*, *local*)

Configure several paths for autosubmit: database, local root and others. Can be configured at system, user or local levels. Local level configuration precedes user level and user level precedes system configuration.

Parameters

- **database_path** (*str*) – path to autosubmit database
- **database_path** – path to autosubmit database
- **local_root_path** (*str*) – path to autosubmit's experiments' directory
- **platforms_conf_path** (*str*) – path to platforms conf file to be used as model for new experiments
- **jobs_conf_path** (*str*) – path to jobs conf file to be used as model for new experiments
- **machine** (*bool*) – True if this configuration has to be stored for all the machine users
- **local** (*bool*) – True if this configuration has to be stored in the local path

static `create`(*expid*, *noplot*)

Creates job list for given experiment. Configuration files must be valid before realizaing this process.

Parameters

- **expid** (*str*) – experiment identifier
- **noplot** (*bool*) – if True, method omits final ploting of joblist. Only needed on large experiments when plotting time can be much larger than creation time.

Returns True if succesful, False if not

Return type bool

static `delete`(*expid*, *force*)
Deletes and experiment from database and experiment's folder

Parameters

- **expid** (*str*) – identifier of the experiment to delete
- **force** (*bool*) – if True, does not ask for confrmation

Returns True if succesful, False if not

Return type bool

static `expid`(*hpc*, *description*, *copy_id=''*, *dummy=False*)
Creates a new experiment for given HPC

Parameters

- **hpc** (*str*) – name of the main HPC for the experiment
- **description** (*str*) – short experiment's description.
- **copy_id** (*str*) – experiment identifier of experiment to copy
- **dummy** (*bool*) – if true, writes a default dummy configuration for testing

Returns experiment identifier. If method fails, returns ''.

Return type str

static `install`()
Creates a new database instance for autosubmit at the configured path

static `monitor`(*expid*, *file_format*)
Plots workflow graph for a given experiment with status of each job coded by node color. Plot is created in experiment's plot folder with name <expid>_<date>_<time>.<file_format>

Parameters

- **expid** (*str*) – identifier of the experiment to plot
- **file_format** (*str*) – plot's file format. It can be pdf, png or ps

static `parse_args`()
Parse arguments given to an executable and start execution of command given

static `recovery`(*expid*, *save*, *all_jobs*)
TODO

Parameters

- **expid** (*str*) – identifier of the experiment to recover
- **save** (*bool*) – If true, recovery saves changes to joblist
- **all_jobs** (*bool*) – if True, it tries to get completed files for all jobs, not only active.

static refresh (*expid*)
Refresh project folder for given experiment

Parameters **expid** (*str*) – experiment identifier

static run_experiment (*expid*)
Runs and experiment (submitting all the jobs properly and repeating its execution in case of failure).

Parameters **expid** (*str*) – identifier of experiment to be run

Returns True if run to the end, False otherwise

Return type bool

static set_status (*expid*, *save*, *final*, *lst*, *filter_chunks*, *filter_status*, *filter_section*)
TODO

Parameters

- **expid** (*str*) – experiment identifier
- **save** (*bool*) –
- **final** (*str*) –
- **lst** (*str*) –
- **filter_chunks** (*str*) –
- **filter_status** (*str*) –
- **filter_section** (*str*) –

static statistics (*expid*, *file_format*)
Plots statistics graph for a given experiment. Plot is created in experiment's plot folder with name <expid>_<date>_<time>.<file_format>

Parameters

- **expid** (*str*) – identifier of the experiment to plot
- **file_format** (*str*) – plot's file format. It can be pdf, png or ps

static test (*expid*, *chunks*, *member=None*, *stardate=None*, *hpc=None*, *branch=None*)
Method to conduct a test for a given experiment. It creates a new experiment for a given experiment qith a given number of chunks with a random start date and a random member to be run on a random HPC.

Parameters

- **expid** (*str*) – experiment identifier
- **chunks** (*int*) – number of chunks to be run by the experiment
- **member** (*str*) – member to be used by the test. If None, it uses a random one from which are defined on the experiment.
- **stardate** (*str*) – start date to be used by the test. If None, it uses a random one from which are defined on the experiment.
- **hpc** (*str*) – HPC to be used by the test. If None, it uses a random one from which are defined on the experiment.
- **branch** (*str*) – branch or revision to be used by the test. If None, it uses configured branch.

Returns True if test was succesful, False otherwise

Return type bool

7.2 autosubmit.config

7.2.1 autosubmit.config.basicConfig

class autosubmit.config.basicConfig.**BasicConfig**
Class to manage configuration for autosubmit path, database and default values for new experiments

static read()
Reads configuration from .autosubmitrc files, first from /etc, then for user directory and last for current path.

7.2.2 autosubmit.config.config_common

class autosubmit.config.config_common.**AutosubmitConfig**(*expid*)
Class to handle experiment configuration coming from file or database

check_conf_files()
Checks configuration files (autosubmit, experiment jobs and queues), looking for invalid values, missing required options. Prints results in log

Returns True if everithing is correct, False if it founds any error

Return type bool

static check_exists(*parser*, *section*, *option*)
Checks if an option exists in given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check

Returns True if option exists, False otherwise

Return type bool

static check_is_boolean(*parser*, *section*, *option*, *must_exist*)
Checks if an option is a boolean value in given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist

Returns True if option value is boolean, False otherwise

Return type bool

static check_is_choice(*parser*, *section*, *option*, *must_exist*, *choices*)
Checks if an option is a valid choice in given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option

- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist
- **choices** (*list*) – valid choices

Returns True if option value is a valid choice, False otherwise

Return type bool

static check_is_int(*parser*, *section*, *option*, *must_exist*)
Checks if an option is an integer value in given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist

Returns True if option value is integer, False otherwise

Return type bool

static check_json(*key*, *value*)
Checks if value is a valid json

Parameters

- **key** (*str*) – key to check
- **value** (*str*) – value

Returns True if value is a valid json, False otherwise

Return type bool

check_parameters()
Function to check configuration of Autosubmit.

Returns True if all variables are set. If some parameter do not exist, the function returns False.

Return type bool

check_proj()
Checks project config file

Returns True if everything is correct, False if it founds any error

Return type bool

static check_regex(*parser*, *section*, *option*, *must_exist*, *regex*)
Checks if an option complies with a regular expression in given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist
- **regex** (*str*) – regular expression to check

Returns True if option complies with regex, False otherwise

Return type bool

experiment_file
Returns experiment's config file name

static get_bool_option(*parser*, *section*, *option*, *default*)
Gets a boolean option from given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to get
- **default** (*bool*) – value to be returned if option is not present

Returns option value

Return type bool

get_chunk_list()
Returns chunk list from experiment's config file

Returns experiment's chunks

Return type list

get_chunk_size_unit()
Unit for the chunk length

Returns Unit for the chunk length Options: {hour, day, month, year}

Return type str

get_date_list()
Returns startdates list from experiment's config file

Returns experiment's startdates

Return type list

get_expid()
Returns experiment identifier read from experiment's config file

Returns experiment identifier

Return type str

get_file_project_conf()
Returns path to project config file from experiment config file

Returns path to project config file

Return type str

get_git_project_branch()
Returns git branch from experiment's config file

Returns git branch

Return type str

get_git_project_commit()
Returns git commit from experiment's config file

Returns git commit

Return type str

get_git_project_origin()
Returns git origin from experiment config file

Returns git origin

Return type str

get_local_project_path()
Gets path to origin for local project

Returns path to local project

Return type str

get_max_waiting_jobs()
Returns max number of waitng jobs from autosubmit's config file

Returns main platforms

Return type int

get_member_list()
Returns members list from experiment's config file

Returns experiment's members

Return type list

get_num_chunks()
Returns number of chunks to run for each member

Returns number of chunks

Return type int

static get_option(*parser*, *section*, *option*, *default*)
Gets an option from given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to get
- **default** (*object*) – value to be returned if option is not present

Returns option value

Return type str

static get_parser(*file_path*)
Gets parser for given file

Parameters **file_path** (*str*) – path to file to be parsed

Returns parser

Return type SafeConfigParser

get_platform()
Returns main platforms from experiment's config file

Returns main platforms

Return type str

get_project_destination()
Returns git commit from experiment's config file

Returns git commit

Return type str

get_project_dir()
Returns experiment's project directory

Returns experiment's project directory

Return type str

get_project_type()
Returns project type from experiment config file

Returns project type

Return type str

get_rerun()
Returns startdates list from experiment's config file

Returns rerurn value

Return type list

get_retrials()
Returns max number of retrials for job from autosubmit's config file

Returns safety sleep time

Return type int

get_safetysleeptime()
Returns safety sleep time from autosubmit's config file

Returns safety sleep time

Return type int

get_svn_project_revision()
Get revision for subversion project

Returns revision for subversion project

Return type str

get_svn_project_url()
Gets subversion project url

Returns subversion project url

Return type str

get_total_jobs()
Returns max number of running jobs from autosubmit's config file

Returns max number of running jobs

Return type int

load_parameters()
Load parameters from experiment and autosubmit config files. If experiment's type is not none, also load parameters from model's config file

Returns a dictionary containing tuples [parameter_name, parameter_value]

Return type dict

load_project_parameters()
Loads parameters from model config file

Returns dictionary containing tuples [parameter_name, parameter_value]

Return type dict

platforms_file
Returns experiment's queues config file name

Returns queues config file's name

Return type str

static **print_parameters**(*title*, *parameters*)
Prints the parameters table in a tabular mode

Parameters

- **title** (*str*) – table's title
- **parameters** – parameters to print

Type list

project_file
Returns model's config file name

read_platforms_conf()
Read platforms configuration file and create defined platforms. Also adds the local remote_platform to the list

Returns platforms defined on file and local remote_platform. None if configuration is invalid

Return type list

reload()
Creates parser objects for configuration files

set_expid(*exp_id*)
Set experiment identifier in autosubmit and experiment config files

Parameters **exp_id** (*str*) – experiment identifier to store

set_git_project_commit()
Function to register in the configuration the commit SHA of the git project version.

set_platform(*hpc*)
Sets main platforms in experiment's config file

Parameters **hpc** – main platforms

Type str

set_safetysleeptime(*sleep_time*)
Sets autosubmit's version in autosubmit's config file

Parameters **sleep_time** (*int*) – value to set

set_version(*autosubmit_version*)
Sets autosubmit's version in autosubmit's config file

Parameters **autosubmit_version** (*str*) – autosubmit's version

7.2.3 autosubmit.config.log

class autosubmit.config.log.**Log**

Static class to manage the log for the application. Messages will be sent to console and to file if it is configured. Levels can be set for each output independently. These levels are (from lower to higher priority):

- •EVERYTHING : this level is just defined to show every output
- •DEBUG
- •INFO
- •RESULT
- •USER_WARNING
- •WARNING
- •ERROR
- •CRITICAL
- •NO_LOG : this level is just defined to remove every output

static critical (*msg*, **args*)

Sends critical errors to the log. It will be shown in red in the console.

Parameters

- **msg** – message to show
- **args** – arguments for message formating (it will be done using format() method on str)

static debug (*msg*, **args*)

Sends debug information to the log

Parameters

- **msg** – message to show
- **args** – arguments for message formating (it will be done using format() method on str)

static error (*msg*, **args*)

Sends errors to the log. It will be shown in red in the console.

Parameters

- **msg** – message to show
- **args** – arguments for message formating (it will be done using format() method on str)

static info (*msg*, **args*)

Sends information to the log

Parameters

- **msg** – message to show
- **args** – arguments for message formating (it will be done using format() method on str)

static result (*msg*, **args*)

Sends results information to the log. It will be shown in green in the console.

Parameters

- **msg** – message to show
- **args** – arguments for message formating (it will be done using format() method on str)

static set_console_level (*level*)
Sets log level for logging to console. Every output of level equal or higher to parameter level will be printed on console

Parameters **level** – new level for console

Returns None

static set_file (*file_path*)
Configure the file to store the log. If another file was specified earlier, new messages will only go to the new file.

Parameters **file_path** (*str*) – file to store the log

static set_file_level (*level*)
Sets log level for logging to file. Every output of level equal or higher to parameter level will be added to log file

Parameters **level** – new level for log file

static user_warning (*msg*, **args*)
Sends warnings for the user to the log. It will be shown in yellow in the console.

Parameters

- **msg** – message to show
- **args** – arguments for message formating (it will be done using format() method on str)

static warning (*msg*, **args*)
Sends program warnings to the log. It will be shown in yellow in the console.

Parameters

- **msg** – message to show
- **args** – arguments for message formating (it will be done using format() method on str)

class autosubmit.config.log.**LogFormatter** (*to_file=False*)
Class to format log output.

Parameters **to_file** (*bool*) – If True, creates a LogFormatter for files; if False, for console

format (*record*)
Format log output, adding labels if needed for log level. If logging to console, also manages font color. If logging to file adds timestamp

Parameters **record** (*LogRecord*) – log record to format

Returns formatted record

Return type str

7.3 autosubmit.database

Module containing functions to manage autosubmit's database.

exception autosubmit.database.db_common.**DbException** (*message*)
Exception class for database errors

autosubmit.database.db_common.**base36decode** (*number*)
Converts a base36 string to a positive integer

Parameters **number** (*str*) – base36 string to convert

Returns number's integer value

Return type int

autosubmit.database.db_common.**base36encode**(*number*, *alphabet='0123456789abcdefghijklmnopqrstuvwxyz'*)

Convert positive integer to a base36 string.

Parameters

- **number** (*int*) – number to convert
- **alphabet** (*str*) – set of characters to use

Returns number's base36 string value

Return type str

autosubmit.database.db_common.**check_db**()

Checks if database file exist

Returns None if exists, terminates program if not

autosubmit.database.db_common.**check_experiment_exists**(*name*, *error_on_inexistence=True*)

Checks if exist an experiment with the given name.

Parameters **name** (*str*) – Experiment name

Returns If experiment exists returns true, if not returns false

Return type bool

autosubmit.database.db_common.**check_name**(*name*)

Checks if it is a valid experiment identifier

Parameters **name** (*str*) – experiment identifier to check

Returns name if is valid, terminates program otherwise

Return type str

autosubmit.database.db_common.**close_conn**(*conn*, *cursor*)

Commits changes and close connection to database

Parameters

- **conn** (*sqlite3.Connection*) – connection to close
- **cursor** (*sqlite3.Cursor*) – cursor to close

autosubmit.database.db_common.**copy_experiment**(*name*, *hpc*, *description*, *version*)

Creates a new experiment by copying an existing experiment

Parameters

- **name** (*str*) – identifier of experiment to copy
- **hpc** (*str*) – name of the main HPC to be used by the experiment
- **description** (*str*) – experiment's description

Returns experiment id for the new experiment

Return type str

autosubmit.database.db_common.**create_db**(*qry*)

Creates a new database for autosubmit

Parameters **qry** (*str*) – query to create the new database

autosubmit.database.db_common.**delete_experiment**(*name*)
Removes experiment from database

Parameters **name** (*str*) – experiment identifier

Returns True if delete is succesful

Return type bool

autosubmit.database.db_common.**last_name_used**()
Gets last experiment identifier used for HPC

Returns last experiment identifier used for HPC, 'empty' if there is none

Return type str

autosubmit.database.db_common.**new_experiment**(*hpc*, *description*, *version*)
Stores a new experiment on the database and generates its identifier

Parameters

- **hpc** (*str*) – name of the main HPC to be used by the experiment
- **description** (*str*) – experiment's description

Returns experiment id for the new experiment

Return type str

autosubmit.database.db_common.**open_conn**(*check_version=True*)
Opens a connection to database

Returns connection object, cursor object

Return type sqlite3.Connection, sqlite3.Cursor

7.4 autosubmit.date

In this python script there are tools to manipulate the dates and make mathematical operations between them.

autosubmit.date.chunk_date_lib.**add_days**(*date*, *number_of_days*, *cal*)
Adds days to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **number_of_days** (*int*) – number of days to add
- **cal** (*str*) – calendar to use

Returns base date plus added days

Return type date

autosubmit.date.chunk_date_lib.**add_hours**(*date*, *number_of_hours*, *cal*)
Adds hours to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **number_of_hours** (*int*) – number of hours to add

- **cal** (*str*) – calendar to use

Returns base date plus added hours

Return type date

autosubmit.date.chunk_date_lib.**add_months**(*date*, *number_of_months*, *cal*)
Adds months to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **number_of_months** (*int*) – number of months to add
- **cal** (*str*) – calendar to use

Returns base date plus added months

Return type date

autosubmit.date.chunk_date_lib.**add_time**(*date*, *total_size*, *chunk_unit*, *cal*)
Adds given time to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **total_size** (*int*) – time to add
- **chunk_unit** (*str*) – unit of time to add
- **cal** (*str*) – calendar to use

Returns result of adding time to base date

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**add_years**(*date*, *number_of_years*)
Adds years to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **number_of_years** (*int*) – number of years to add

Returns base date plus added years

Return type date

autosubmit.date.chunk_date_lib.**chunk_end_date**(*start_date*, *chunk_length*, *chunk_unit*, *cal*)
Gets chunk interval end date

Parameters

- **start_date** (*datetime.datetime*) – chunk's start date
- **chunk_length** (*int*) – length of the chunks
- **chunk_unit** (*str*) – chunk length unit
- **cal** (*str*) – calendar to use

Returns chunk's end date

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**chunk_start_date**(*date*, *chunk*, *chunk_length*, *chunk_unit*, *cal*)

Gets chunk's interval start date

Parameters

- **date** (*datetime.datetime*) – start date for member
- **chunk** (*int*) – number of chunk
- **chunk_length** (*int*) – length of chunks
- **chunk_unit** (*str*) – chunk length unit
- **cal** (*str*) – calendar to use

Returns chunk's start date

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**date2str**(*date*, *date_format=''*)

Converts a datetime object to a str

Parameters **date** (*datetime.datetime*) – date to convert

Return type str

autosubmit.date.chunk_date_lib.**parse_date**(*string_date*)

Parses a string into a datetime object

Parameters **string_date** (*str*) – string to parse

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**previous_day**(*date*, *cal*)

Gets previous day

Parameters

- **date** (*datetime.datetime*) – base date
- **cal** (*str*) – calendar to use

Returns base date minus one day

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**sub_days**(*date*, *number_of_days*, *cal*)

Substract days to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **number_of_days** (*int*) – number of days to substract
- **cal** (*str*) – calendar to use

Returns base date minus substracted days

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**subs_dates**(*start_date*, *end_date*, *cal*)

Gets days between start_date and end_date

Parameters

- **start_date** (*datetime.datetime*) – interval's start date
- **end_date** (*datetime.datetime*) – interval's end date

- **cal** (*str*) – calendar to use

Returns interval length in days

Return type int

7.5 autosubmit.git

class autosubmit.git.git_common.**AutosubmitGit**(*expid*)
Class to handle experiment git repository

Parameters **expid** (*str*) – experiment identifier

clean_git()
Function to clean space on BasicConfig.LOCAL_ROOT_DIR/git directory.

7.6 autosubmit.job

Main module for autosubmit. Only contains an interface class to all functionality implemented on autosubmit

class autosubmit.job.job.**Job**(*name*, *jobid*, *status*, *priority*)
Class to handle all the tasks with Jobs at HPC. A job is created by default with a name, a jobid, a status and a type. It can have children and parents. The inheritance reflects the dependency between jobs. If Job2 must wait until Job1 is completed then Job2 is a child of Job1. Inversely Job1 is a parent of Job2

Parameters

- **name** (*str*) – job's name
- **jobid** (*int*) – job's identifier
- **status** (*Status*) – job inicial status
- **priority** (*int*) – job's priority

add_parent(**new_parent*)
Add parents for the job. It also adds current job as a child for all the new parents

Parameters ***new_parent** (*Job*) – job parent

ancestors
Returns all job's ancestors

Returns job ancestors

Return type set

check_completion(*default_status=-1*)
Check the presence of *COMPLETED* file and touch a Checked or failed file. Change statis to COMPLETED if *COMPLETED* file exists and to FAILED otherwise.

check_end_time()
Returns end time from completed file

Returns completed date and time

Return type str

check_fail_queued_time()
Returns total time spent waiting for failed jobs

Returns total time waiting in HPC platforms for failed jobs

Return type str

check_fail_run_time()
Returns total time running for failed jobs

Returns total time running in HPC for failed jobs

Return type str

check_failed_times()
Returns number of failed attempts before completing the job

Returns failed attempts to run

Return type str

check_queued_time()
Returns job's waiting time in HPC

Returns total time waiting in HPC platforms

Return type str

check_run_time()
Returns job's running time

Returns total time running

Return type str

check_script(*as_conf*)
Checks if script is well formed

Parameters **as_conf** (*AutosubmitConfig*) – configuration file

Returns true if not problem has been detected, false otherwise

Return type bool

children
Returns a list containing all children of the job

Returns child jobs

Return type set

compare_by_id(*other*)
Compare jobs by ID

Parameters **other** (*Job*) – job to compare

Returns comparison result

Return type bool

compare_by_name(*other*)
Compare jobs by name

Parameters **other** (*Job*) – job to compare

Returns comparison result

Return type bool

compare_by_status(*other*)
Compare jobs by status value

Parameters **other** (*Job*) – job to compare

Returns comparison result

Return type bool

create_script(*as_conf*)
Creates script file to be run for the job

Parameters **as_conf** (*AutosubmitConfig*) – configuration object

Returns script's filename

Return type str

delete_child(*child*)
Removes a child from the job

Parameters **child** (*Job*) – child to remove

delete_parent(*parent*)
Remove a parent from the job

Parameters **parent** (*Job*) – parent to remove

get_platform()
Returns the platforms to be used by the job. Chooses between serial and parallel platforms

:return HPCPlatform object for the job to use :rtype: HPCPlatform

get_queue()
Returns the queue to be used by the job. Chooses between serial and parallel platforms

:return HPCPlatform object for the job to use :rtype: HPCPlatform

has_children()
Returns true if job has any children, else return false

Returns true if job has any children, otherwise return false

Return type bool

has_parents()
Returns true if job has any parents, else return false

Returns true if job has any parent, otherwise return false

Return type bool

inc_fail_count()
Increments fail count

log_job()
Prints job information in log

long_name
Job's long name. If not setted, returns name

Returns long name

Return type str

parents
Return parent jobs list

Returns parent jobs

Return type set

print_job()
Prints debug information about the job

print_parameters()
Print sjob parameters in log

remove_dependencies()
Checks if job is completed and then remove dependencies for childs

set_platform(*value*)
Sets the HPC platforms to be used by the job.

Parameters **value** (*HPCPlatform*) – platforms to set

set_queue(*value*)
Sets the queue to be used by the job.

Parameters **value** (*HPCPlatform*) – queue to set

short_name
Job short name

Returns short name

Return type str

update_content(*project_dir*)
Create the script content to be run for the job

Parameters **project_dir** (*str*) – project directory

Returns script code

Return type str

update_parameters(*as_conf*)
Refresh parameters value

Parameters **as_conf** (*AutosubmitConfig*) –

class autosubmit.job.job_common.**StatisticsSnippet**
Class to handle the statistics snippet of a job. It contains header and tailer for local and remote jobs

class autosubmit.job.job_common.**Status**
Class to handle the status of a job

class autosubmit.job.job_list.**DicJobs**(*joblist*, *parser*, *date_list*, *member_list*, *chunk_list*, *date_format*)
Class to create jobs from conf file and to find jobs by stardate, member and chunk

Parameters

- **joblist** (*JobList*) – joblist to use
- **parser** (*SafeConfigParser*) – jobs conf file parser
- **date_list** (*list*) – startdates
- **member_list** (*list*) – member
- **chunk_list** (*list*) – chunks
- **date_format** (*str*) – option to formate dates

get_jobs(*section*, *date=None*, *member=None*, *chunk=None*)
Return all the jobs matching section, date, member and chunk provided. If any parameter is none, returns

all the jobs without checking that parameter value. If a job has one parameter to None, is returned if all the others match parameters passed

Parameters

- **section** (*str*) – section to return
- **date** (*str*) – stardate to return
- **member** (*str*) – member to return
- **chunk** (*int*) – chunk to return

Returns jobs matching parameters passed

Return type list

get_option(*section*, *option*, *default*)
Returns value for a given option

Parameters

- **section** (*str*) – section name
- **option** (*str*) – option to return
- **default** (*object*) – value to return if not defined in configuration file

read_section(*section*, *priority*)
Read a section from jobs conf and creates all jobs for it

Parameters

- **section** (*str*) – section to read
- **priority** (*int*) – priority for the jobs

class autosubmit.job.job_list.**JobList**(*expid*)
Class to manage the list of jobs to be run by autosubmit

Parameters **expid** (*str*) – experiment's indentifier

check_scripts(*as_conf*)
When we have created the scripts, all parameters should have been substituted. %PARAMETER% handlers not allowed

Parameters **as_conf** (*AutosubmitConfig*) – experiment configuration

create(*date_list*, *member_list*, *num_chunks*, *parameters*, *date_format*)
Creates all jobs needed for the current workflow

Parameters

- **date_list** (*list*) – start dates
- **member_list** (*list*) – members
- **num_chunks** (*int*) – number of chunks to run
- **parameters** (*dict*) – parameters for the jobs
- **date_format** (*str*) – option to formate dates

expid
Returns experiment identifier

Returns experiment's identifier

Return type str

get_active()
Returns a list of active jobs (In platforms, Ready)

Returns active jobs

Return type list

get_completed()
Returns a list of completed jobs

Returns completed jobs

Return type list

get_failed()
Returns a list of failed jobs

Returns failed jobs

Return type list

get_finished()
Returns a list of jobs finished (Completed, Failed)

Returns finsihed jobs

Return type list

get_in_queue()
Returns a list of jobs in the platforms (Submitted, Running, Queuing)

Returns jobs in platforms

Return type list

get_job_by_name(*name*)
Returns the job that its name matches parameter name

Parameters **name** (*str*) – name to look for

Returns found job

Return type job

get_job_list()
Get inner job list

Returns job list

Return type list

get_not_in_queue()
Returns a list of jobs NOT in the platforms (Ready, Waiting)

Returns jobs not in platforms

Return type list

get_queuing()
Returns a list of jobs queuing

Returns queuedjobs

Return type list

get_ready()
Returns a list of ready jobs

Returns ready jobs

Return type list

get_running()
Returns a list of jobs running

Returns running jobs

Return type list

get_submitted()
Returns a list of submitted jobs

Returns submitted jobs

Return type list

get_unknown()
Returns a list of jobs on unknown state

Returns unknown state jobs

Return type list

get_waiting()
Returns a list of jobs waiting

Returns waiting jobs

Return type list

load()
Recreates an stored joblist from the pickle file

Returns loaded joblist object

Return type JobList

static load_file(*filename*)
Recreates an stored joblist from the pickle file

Parameters **filename** (*str*) – pickle file to load

Returns loaded joblist object

Return type JobList

remove_rerun_only_jobs()
Removes all jobs to be runned only in reruns

rerun(*chunk_list*)
Updates joblist to rerun the jobs specified by chunk_list

Parameters **chunk_list** (*str*) – list of chunks to rerun

Returns

save()
Stores joblist as a pickle file

Returns loaded joblist object

Return type JobList

sort_by_id()
Returns a list of jobs sorted by id

Returns jobs sorted by ID

Return type list

sort_by_name()
Returns a list of jobs sorted by name

Returns jobs sorted by name

Return type list

sort_by_status()
Returns a list of jobs sorted by status

Returns job sorted by status

Return type list

sort_by_type()
Returns a list of jobs sorted by type

Returns job sorted by type

Return type list

update_genealogy()
When we have created the joblist, every type of job is created. Update genealogy remove jobs that have no templates

update_shortened_names()
In some cases the scheduler only can operate with names shorter than 15 characters. Update the job list replacing job names by the corresponding shortened job name

7.7 autosubmit.monitor

class autosubmit.monitor.monitor.**Monitor**
Class to handle monitoring of Jobs at HPC.

static clean_plot(*expid*)
Function to clean space on BasicConfig.LOCAL_ROOT_DIR/plot directory. Removes all plots except last two.

Parameters **expid** (*str*) – experiment's identifier

static clean_stats(*expid*)
Function to clean space on BasicConfig.LOCAL_ROOT_DIR/plot directory. Removes all stats' plots except last two.

Parameters **expid** (*str*) – experiment's identifier

static color_status(*status*)
Return color associated to given status

Parameters **status** (*Status*) – status

Returns color

Return type str

static create_bar_diagram(*expid*, *joblist*, *output_file*)
Function to plot statistics

Parameters

- **expid** (*str*) – experiment's identifier
- **joblist** (*JobList*) – joblist to plot
- **output_file** (*str*) – path to create file

create_tree_list (*expid*, *joblist*)
Create graph from joblist

Parameters

- **expid** (*str*) – experiment's identifier
- **joblist** (*JobList*) – joblist to plot

Returns created graph

Return type pydotplus.Dot

generate_output (*expid*, *joblist*, *output_format='pdf'*)
Plots graph for joblist and stores it in a file

Parameters

- **expid** (*str*) – experiment's identifier
- **joblist** (*JobList*) – joblist to plot
- **output_format** (*str (png, pdf, ps)*) – file format for plot

generate_output_stats (*expid*, *joblist*, *output_format='pdf'*)
Plots stats for joblist and stores it in a file

Parameters

- **expid** (*str*) – experiment's identifier
- **joblist** (*JobList*) – joblist to plot
- **output_format** (*str (png, pdf, ps)*) – file format for plot

7.8 autosubmit.queue

class autosubmit.queue.hpcqueue.**HPCQueue**
Base class to manage schedulers

cancel_job (*job_id*)
Cancels job

Parameters **job_id** (*int*) – job to cancel

check_host ()
Checks host availability

Returns True if host is available, False otherwise

Return type bool

check_job (*job_id*)
Checks job statuts

Parameters **job_id** (*int*) – job to check

Returns current job status

Return type Status

check_remote_log_dir()
Creates log dir on remote host

close_connection()
Closes ssh connection to host

connect()
Creates ssh connection to host

Returns True if connection is created, False otherwise

Return type bool

get_checkhost_cmd()
Gets command to check queue availability

Returns command to check queue availability

Return type str

get_checkjob_cmd(*job_id*)
Returns command to check job status on remote queue

Parameters

- **job_id** – id of job to check
- **job_id** – int

Returns command to check job status

Return type str

get_completed_files(*jobname*)
Copies *COMPLETED* files from remote to local

Parameters **jobname** (*str*) – name of job to check

Returns True if succesful, False otherwise

Return type bool

get_file(*remote_path*, *local_path*)
Copies file in remote_path to local_path

Parameters

- **remote_path** (*str*) – path to the remote file to copy
- **local_path** (*str*) – path to the local file to create

Returns True if succesful, False if failed

Return type bool

get_header(*job*)
Gets header to be used by the job

Parameters **job** (*Job*) – job

Returns header to use

Return type str

get_mkdir_cmd()
Gets command to create directories on HPC

Returns command to create directories on HPC

Return type str

static get_pscall(*job_id*)
Gets command to check if a job is running given process identifier

Parameters **job_id** (*int*) – process indentifier

Returns command to check job status script

Return type str

static get_qstatjob(*job_id*)
Gets qstat command for given job id

Parameters **job_id** (*int*) – job to check

Returns qstat command for job

Return type str

get_remote_log_dir()
Gets remote directory for logs

Returns log directory path

Return type str

get_serial_queue()
Returns serial queue for current host. If not configured, returns self

Returns serial queue for host

Return type HPCQueue

get_shcall(*job_script*)
Gets execution command for given job

Parameters **job_script** (*str*) – script to run

Returns command to execute script

Return type str

get_ssh_output()
Gets output from last command executed

Returns output from last command

Return type str

get_submit_cmd(*job_script*)
Get command to add job to scheduler

Parameters

- **job_script** – path to job script
- **job_script** – str

Returns command to submit job to queue

Return type str

get_submitted_job_id(*output*)
Parses submit command output to extract job id :param output: output to parse :type output: str :return: job id :rtype: str

header
Header to add to jobs for scheduler cofiguration

Returns header

Return type object

jobs_in_queue()
Get jobs in queue in this host

Returns jobs in queue

Return type list

parse_job_output(*output*)
Parses check job command output so it can be interpreted by autosubmit

Parameters **output** (*str*) – output to parse

Returns job status

Return type str

send_command(*command*)
Sends given command to HPC

Parameters **command** (*str*) – command to send

Returns True if executed, False if failed

Return type bool

send_file(*local_path*, *root_path*)
Copies file in local_path to remote_path

Parameters

- **local_path** (*str*) – path to the local file to copy
- **root_path** (*str*) – path to the remote file to create

Returns True if succesful, False if failed

Return type bool

send_script(*job_script*)
Send a script to remote host

Parameters **job_script** (*str*) – name of script to send

set_budget(*new_budget*)
Sets budget :param new_budget: project :type new_budget: str

set_host(*new_host*)
Sets host name :param new_host: host :type new_host: str

set_project(*new_project*)
Sets project name :param new_project: project :type new_project: str

set_remote_log_dir(*new_remote_log_dir*)
Sets remote directory for logs :param new_remote_log_dir: path to log directory :type new_remote_log_dir: str

set_scratch(*new_scratch*)
Sets scracth directory name :param new_scratch: scratch path :type new_scratch: str

set_serial_queue(*value*)
Configures serial queue for current host.

Parameters **value** (*HPCQueue*) – serial queue for host

set_user(*new_user*)
Sets user name :param new_user: user :type new_user: str

submit_job(*job_script*)
Submits job to scheduler and returns job id

Parameters **job_script** (*str*) – script path

Returns job id

Return type int

update_cmds()
Updates commands for queue

exception autosubmit.queue.hpcqueue.**HPCQueueException**(*msg*)
Bases: exceptions.Exception

Exception raised from HPC queues

class autosubmit.queue.ecqueue.**EcCcaHeader**
Class to handle the ECMWF headers of a job

class autosubmit.queue.ecqueue.**EcHeader**
Class to handle the ECMWF headers of a job

class autosubmit.queue.ecqueue.**EcQueue**(*expid*, *scheduler*)
Bases: autosubmit.queue.hpcqueue.HPCQueue

Class to manage queues with eceacces

Parameters

- **expid** (*str*) – experiment's identifier
- **scheduler** (*str (pbs, loadleveler)*) – scheduler to use

cancel_job(*job_id*)
Cancels job

Parameters **job_id** (*int*) – job to cancel

check_host()
Checks host availability

Returns True if host is available, False otherwise

Return type bool

check_job(*job_id*)
Checks job statuts

Parameters **job_id** (*int*) – job to check

Returns current job status

Return type Status

check_remote_log_dir()
Creates log dir on remote host

close_connection()
Closes ssh connection to host

connect()
In this case, it does nothing because connection is established foe each command

Returns True

Return type bool

get_completed_files(*jobname*)
Copies *COMPLETED* files from remote to local

Parameters **jobname** (*str*) – name of job to check

Returns True if succesful, False otherwise

Return type bool

get_header(*job*)
Gets header to be used by the job

Parameters **job** (*Job*) – job

Returns header to use

Return type str

static get_pscall(*job_id*)
Gets command to check if a job is running given process identifier

Parameters **job_id** (*int*) – process indentifier

Returns command to check job status script

Return type str

static get_qstatjob(*job_id*)
Gets qstat command for given job id

Parameters **job_id** (*int*) – job to check

Returns qstat command for job

Return type str

get_serial_queue()
Returns serial queue for current host. If not configured, returns self

Returns serial queue for host

Return type HPCQueue

get_shcall(*job_script*)
Gets execution command for given job

Parameters **job_script** (*str*) – script to run

Returns command to execute script

Return type str

header
Header to add to jobs for scheduler cofiguration

Returns header

Return type object

jobs_in_queue()
Returns empty list because ecacces does not support this command

Returns empty list

Return type list

send_script(*job_script*)
Send a script to remote host

Parameters **job_script** (*str*) – name of script to send

set_budget(*new_budget*)
Sets budget :param new_budget: project :type new_budget: str

set_host(*new_host*)
Sets host name :param new_host: host :type new_host: str

set_project(*new_project*)
Sets project name :param new_project: project :type new_project: str

set_remote_log_dir(*new_remote_log_dir*)
Sets remote directory for logs :param new_remote_log_dir: path to log directory :type new_remote_log_dir: str

set_scratch(*new_scratch*)
Sets scracth directory name :param new_scratch: scratch path :type new_scratch: str

set_serial_queue(*value*)
Configures serial queue for current host.

Parameters **value** (*HPCQueue*) – serial queue for host

set_user(*new_user*)
Sets user name :param new_user: user :type new_user: str

submit_job(*job_script*)
Submits job to scheduler and returns job id

Parameters **job_script** (*str*) – script path

Returns job id

Return type int

update_cmds()
Updates commands for queue

class autosubmit.queue.lsfqueue.**LsfHeader**
Class to handle the MareNostrum3 headers of a job

class autosubmit.queue.lsfqueue.**LsfQueue**(*expid*)
Bases: autosubmit.queue.hpcqueue.HPCQueue

Class to manage jobs to host using LSF scheduler

Parameters **expid** (*str*) – experiment's identifier

cancel_job(*job_id*)
Cancels job

Parameters **job_id** (*int*) – job to cancel

check_host()
Checks host availability

Returns True if host is available, False otherwise

Return type bool

check_job(*job_id*)
Checks job statuts

Parameters **job_id** (*int*) – job to check

Returns current job status

Return type Status

check_remote_log_dir()
Creates log dir on remote host

close_connection()
Closes ssh connection to host

connect()
Creates ssh connection to host

Returns True if connection is created, False otherwise

Return type bool

get_completed_files(*jobname*)
Copies *COMPLETED* files from remote to local

Parameters **jobname** (*str*) – name of job to check

Returns True if succesful, False otherwise

Return type bool

get_file(*remote_path*, *local_path*)
Copies file in remote_path to local_path

Parameters

- **remote_path** (*str*) – path to the remote file to copy
- **local_path** (*str*) – path to the local file to create

Returns True if succesful, False if failed

Return type bool

get_header(*job*)
Gets header to be used by the job

Parameters **job** (*Job*) – job

Returns header to use

Return type str

static get_pscall(*job_id*)
Gets command to check if a job is running given process identifier

Parameters **job_id** (*int*) – process indentifier

Returns command to check job status script

Return type str

static get_qstatjob(*job_id*)
Gets qstat command for given job id

Parameters **job_id** (*int*) – job to check

Returns qstat command for job

Return type str

get_serial_queue()
Returns serial queue for current host. If not configured, returns self

Returns serial queue for host

Return type HPCQueue

get_shcall(*job_script*)
Gets execution command for given job

Parameters **job_script** (*str*) – script to run

Returns command to execute script

Return type str

get_ssh_output()
Gets output from last command executed

Returns output from last command

Return type str

header
Header to add to jobs for scheduler cofiguration

Returns header

Return type object

send_command(*command*)
Sends given command to HPC

Parameters **command** (*str*) – command to send

Returns True if executed, False if failed

Return type bool

send_file(*local_path*, *root_path*)
Copies file in local_path to remote_path

Parameters

- **local_path** (*str*) – path to the local file to copy
- **root_path** (*str*) – path to the remote file to create

Returns True if succesful, False if failed

Return type bool

send_script(*job_script*)
Send a script to remote host

Parameters **job_script** (*str*) – name of script to send

set_budget(*new_budget*)
Sets budget :param new_budget: project :type new_budget: str

set_host(*new_host*)
Sets host name :param new_host: host :type new_host: str

set_project(*new_project*)
Sets project name :param new_project: project :type new_project: str

set_remote_log_dir(*new_remote_log_dir*)
Sets remote directory for logs :param new_remote_log_dir: path to log directory :type new_remote_log_dir: str

set_scratch(*new_scratch*)
Sets scracth directory name :param new_scratch: scratch path :type new_scratch: str

set_serial_queue(*value*)
Configures serial queue for current host.

Parameters **value** (*HPCQueue*) – serial queue for host

set_user(*new_user*)
Sets user name :param new_user: user :type new_user: str

submit_job(*job_script*)
Submits job to scheduler and returns job id

Parameters **job_script** (*str*) – script path

Returns job id

Return type int

class autosubmit.queue.pbsqueue.**PBSQueue**(*expid*, *version*)
Bases: autosubmit.queue.hpcqueue.HPCQueue

Class to manage jobs to host using PBS scheduler

Parameters

- **expid** (*str*) – experiment's identifier
- **version** (*str*) – scheduler version

cancel_job(*job_id*)
Cancels job

Parameters **job_id** (*int*) – job to cancel

check_host()
Checks host availability

Returns True if host is available, False otherwise

Return type bool

check_job(*job_id*)
Checks job statuts

Parameters **job_id** (*int*) – job to check

Returns current job status

Return type Status

check_remote_log_dir()
Creates log dir on remote host

close_connection()
Closes ssh connection to host

connect()
Creates ssh connection to host

Returns True if connection is created, False otherwise

Return type bool

get_completed_files(*jobname*)
Copies *COMPLETED* files from remote to local

Parameters **jobname** (*str*) – name of job to check

Returns True if succesful, False otherwise

Return type bool

get_file(*remote_path*, *local_path*)
Copies file in remote_path to local_path

Parameters

- **remote_path** (*str*) – path to the remote file to copy
- **local_path** (*str*) – path to the local file to create

Returns True if succesful, False if failed

Return type bool

get_header(*job*)
Gets header to be used by the job

Parameters **job** (*Job*) – job

Returns header to use

Return type str

static get_pscall(*job_id*)
Gets command to check if a job is running given process identifier

Parameters **job_id** (*int*) – process indentifier

Returns command to check job status script

Return type str

static get_qstatjob(*job_id*)
Gets qstat command for given job id

Parameters **job_id** (*int*) – job to check

Returns qstat command for job

Return type str

get_serial_queue()
Returns serial queue for current host. If not configured, returns self

Returns serial queue for host

Return type HPCQueue

get_shcall(*job_script*)
Gets execution command for given job

Parameters **job_script** (*str*) – script to run

Returns command to execute script

Return type str

get_ssh_output()
Gets output from last command executed

Returns output from last command

Return type str

header
Header to add to jobs for scheduler cofiguration

Returns header

Return type object

send_command(*command*)
Sends given command to HPC

Parameters **command** (*str*) – command to send

Returns True if executed, False if failed

Return type bool

send_file(*local_path*, *root_path*)
Copies file in local_path to remote_path

Parameters

- **local_path** (*str*) – path to the local file to copy
- **root_path** (*str*) – path to the remote file to create

Returns True if succesful, False if failed

Return type bool

send_script(*job_script*)
Send a script to remote host

Parameters **job_script** (*str*) – name of script to send

set_budget(*new_budget*)
Sets budget :param new_budget: project :type new_budget: str

set_host(*new_host*)
Sets host name :param new_host: host :type new_host: str

set_project(*new_project*)
Sets project name :param new_project: project :type new_project: str

set_remote_log_dir(*new_remote_log_dir*)
Sets remote directory for logs :param new_remote_log_dir: path to log directory :type new_remote_log_dir: str

set_scratch(*new_scratch*)
Sets scracth directory name :param new_scratch: scratch path :type new_scratch: str

set_serial_queue(*value*)
Configures serial queue for current host.

Parameters **value** (*HPCQueue*) – serial queue for host

set_user(*new_user*)
Sets user name :param new_user: user :type new_user: str

submit_job(*job_script*)
Submits job to scheduler and returns job id

Parameters **job_script** (*str*) – script path

Returns job id

Return type int

class autosubmit.queue.pbsqueue.**Pbs10Header**
Class to handle the Hector headers of a job

class autosubmit.queue.pbsqueue.**Pbs11Header**
Class to handle the Lindgren headers of a job

class autosubmit.queue.pbsqueue.**Pbs12Header**
Class to handle the Archer headers of a job

class autosubmit.queue.psqueue.**PsHeader**
Class to handle the Ps headers of a job

class autosubmit.queue.psqueue.**PsQueue**(*expid*)
Bases: autosubmit.queue.hpcqueue.HPCQueue

Class to manage jobs to host not using any scheduler

Parameters **expid** (*str*) – experiment's identifier

cancel_job(*job_id*)
Cancels job

Parameters **job_id** (*int*) – job to cancel

check_host()
Checks host availability

Returns True if host is available, False otherwise

Return type bool

check_job(*job_id*)
Checks job statuts

Parameters **job_id** (*int*) – job to check

Returns current job status

Return type Status

check_remote_log_dir()
Creates log dir on remote host

close_connection()
Closes ssh connection to host

connect()
Creates ssh connection to host

Returns True if connection is created, False otherwise

Return type bool

get_completed_files(*jobname*)
Copies *COMPLETED* files from remote to local

Parameters **jobname** (*str*) – name of job to check

Returns True if succesful, False otherwise

Return type bool

get_file(*remote_path*, *local_path*)
Copies file in remote_path to local_path

Parameters

- **remote_path** (*str*) – path to the remote file to copy
- **local_path** (*str*) – path to the local file to create

Returns True if succesful, False if failed

Return type bool

get_header(*job*)
Gets header to be used by the job

Parameters **job** (*Job*) – job

Returns header to use

Return type str

static get_pscall(*job_id*)
Gets command to check if a job is running given process identifier

Parameters **job_id** (*int*) – process indentifier

Returns command to check job status script

Return type str

static get_qstatjob(*job_id*)
Gets qstat command for given job id

Parameters **job_id** (*int*) – job to check

Returns qstat command for job

Return type str

get_serial_queue()
Returns serial queue for current host. If not configured, returns self

Returns serial queue for host

Return type HPCQueue

get_shcall(*job_script*)
Gets execution command for given job

Parameters **job_script** (*str*) – script to run

Returns command to execute script

Return type str

get_ssh_output()
Gets output from last command executed

Returns output from last command

Return type str

header
Header to add to jobs for scheduler cofiguration

Returns header

Return type object

send_command(*command*)
Sends given command to HPC

Parameters **command** (*str*) – command to send

Returns True if executed, False if failed

Return type bool

send_file(*local_path*, *root_path*)
Copies file in local_path to remote_path

Parameters

- **local_path** (*str*) – path to the local file to copy
- **root_path** (*str*) – path to the remote file to create

Returns True if succesful, False if failed

Return type bool

send_script(*job_script*)
Send a script to remote host

Parameters **job_script** (*str*) – name of script to send

set_budget(*new_budget*)
Sets budget :param new_budget: project :type new_budget: str

set_host(*new_host*)
Sets host name :param new_host: host :type new_host: str

set_project(*new_project*)
Sets project name :param new_project: project :type new_project: str

set_remote_log_dir(*new_remote_log_dir*)
Sets remote directory for logs :param new_remote_log_dir: path to log directory :type new_remote_log_dir: str

set_scratch(*new_scratch*)
Sets scracth directory name :param new_scratch: scratch path :type new_scratch: str

set_serial_queue(*value*)
Configures serial queue for current host.

Parameters **value** (*HPCQueue*) – serial queue for host

set_user(*new_user*)
Sets user name :param new_user: user :type new_user: str

submit_job(*job_script*)
Submits job to scheduler and returns job id

Parameters **job_script** (*str*) – script path

Returns job id

Return type int

class autosubmit.queue.sgequeue.**SgeHeader**
Class to handle the Ithaca headers of a job

class autosubmit.queue.sgequeue.**SgeQueue**(*expid*)
Bases: autosubmit.queue.hpcqueue.HPCQueue

Class to manage jobs to host using SGE scheduler

Parameters **expid** (*str*) – experiment's identifier

cancel_job(*job_id*)
Cancels job

Parameters **job_id** (*int*) – job to cancel

check_host()
Checks host availability

Returns True if host is available, False otherwise

Return type bool

check_job(*job_id*)
Checks job statuts

Parameters **job_id** (*int*) – job to check

Returns current job status

Return type Status

check_remote_log_dir()
Creates log dir on remote host

close_connection()
Closes ssh connection to host

connect()
Creates ssh connection to host

Returns True if connection is created, False otherwise

Return type bool

get_completed_files(*jobname*)
Copies *COMPLETED* files from remote to local

Parameters **jobname** (*str*) – name of job to check

Returns True if succesful, False otherwise

Return type bool

get_file(*remote_path*, *local_path*)
Copies file in remote_path to local_path

Parameters

- **remote_path** (*str*) – path to the remote file to copy
- **local_path** (*str*) – path to the local file to create

Returns True if succesful, False if failed

Return type bool

get_header(*job*)
Gets header to be used by the job

Parameters **job** (*Job*) – job

Returns header to use

Return type str

static **get_pscall**(*job_id*)
Gets command to check if a job is running given process identifier

Parameters **job_id** (*int*) – process indentifier

Returns command to check job status script

Return type str

static **get_qstatjob**(*job_id*)
Gets qstat command for given job id

Parameters **job_id** (*int*) – job to check

Returns qstat command for job

Return type str

get_serial_queue()
Returns serial queue for current host. If not configured, returns self

Returns serial queue for host

Return type HPCQueue

get_shcall(*job_script*)
Gets execution command for given job

Parameters **job_script** (*str*) – script to run

Returns command to execute script

Return type str

get_ssh_output()
Gets output from last command executed

Returns output from last command

Return type str

header
Header to add to jobs for scheduler cofiguration

Returns header

Return type object

send_command(*command*)
Sends given command to HPC

Parameters **command** (*str*) – command to send

Returns True if executed, False if failed

Return type bool

send_file(*local_path*, *root_path*)
Copies file in local_path to remote_path

Parameters

- **local_path** (*str*) – path to the local file to copy
- **root_path** (*str*) – path to the remote file to create

Returns True if succesful, False if failed

Return type bool

send_script (*job_script*)
Send a script to remote host

Parameters **job_script** (*str*) – name of script to send

set_budget (*new_budget*)
Sets budget :param new_budget: project :type new_budget: str

set_host (*new_host*)
Sets host name :param new_host: host :type new_host: str

set_project (*new_project*)
Sets project name :param new_project: project :type new_project: str

set_remote_log_dir (*new_remote_log_dir*)
Sets remote directory for logs :param new_remote_log_dir: path to log directory :type new_remote_log_dir: str

set_scratch (*new_scratch*)
Sets scracth directory name :param new_scratch: scratch path :type new_scratch: str

set_serial_queue (*value*)
Configures serial queue for current host.

Parameters **value** (*HPCQueue*) – serial queue for host

set_user (*new_user*)
Sets user name :param new_user: user :type new_user: str

submit_job (*job_script*)
Submits job to scheduler and returns job id

Parameters **job_script** (*str*) – script path

Returns job id

Return type int

class autosubmit.queue.slurmqueue.**SlurmHeader**
Class to handle the SLURM headers of a job

class autosubmit.queue.slurmqueue.**SlurmQueue** (*expid*)
Bases: autosubmit.queue.hpcqueue.HPCQueue

Class to manage jobs to host using SLURM scheduler

Parameters **expid** (*str*) – experiment's identifier

cancel_job (*job_id*)
Cancels job

Parameters **job_id** (*int*) – job to cancel

check_host ()
Checks host availability

Returns True if host is available, False otherwise

Return type bool

check_job (*job_id*)
Checks job statuts

Parameters **job_id** (*int*) – job to check

Returns current job status

Return type Status

check_remote_log_dir()
Creates log dir on remote host

close_connection()
Closes ssh connection to host

connect()
Creates ssh connection to host

Returns True if connection is created, False otherwise

Return type bool

get_completed_files(*jobname*)
Copies *COMPLETED* files from remote to local

Parameters **jobname** (*str*) – name of job to check

Returns True if succesful, False otherwise

Return type bool

get_file(*remote_path*, *local_path*)
Copies file in remote_path to local_path

Parameters

- **remote_path** (*str*) – path to the remote file to copy
- **local_path** (*str*) – path to the local file to create

Returns True if succesful, False if failed

Return type bool

get_header(*job*)
Gets header to be used by the job

Parameters **job** (*Job*) – job

Returns header to use

Return type str

static get_pscall(*job_id*)
Gets command to check if a job is running given process identifier

Parameters **job_id** (*int*) – process indentifier

Returns command to check job status script

Return type str

static get_qstatjob(*job_id*)
Gets qstat command for given job id

Parameters **job_id** (*int*) – job to check

Returns qstat command for job

Return type str

get_serial_queue()
Returns serial queue for current host. If not configured, returns self

Returns serial queue for host

Return type HPCQueue

get_shcall(*job_script*)
Gets execution command for given job

Parameters **job_script** (*str*) – script to run

Returns command to execute script

Return type str

get_ssh_output()
Gets output from last command executed

Returns output from last command

Return type str

header
Header to add to jobs for scheduler cofiguration

Returns header

Return type object

send_command(*command*)
Sends given command to HPC

Parameters **command** (*str*) – command to send

Returns True if executed, False if failed

Return type bool

send_file(*local_path*, *root_path*)
Copies file in local_path to remote_path

Parameters

- **local_path** (*str*) – path to the local file to copy
- **root_path** (*str*) – path to the remote file to create

Returns True if succesful, False if failed

Return type bool

send_script(*job_script*)
Send a script to remote host

Parameters **job_script** (*str*) – name of script to send

set_budget(*new_budget*)
Sets budget :param new_budget: project :type new_budget: str

set_host(*new_host*)
Sets host name :param new_host: host :type new_host: str

set_project(*new_project*)
Sets project name :param new_project: project :type new_project: str

set_remote_log_dir(*new_remote_log_dir*)
Sets remote directory for logs :param new_remote_log_dir: path to log directory :type new_remote_log_dir: str

set_scratch(*new_scratch*)
Sets scracth directory name :param new_scratch: scratch path :type new_scratch: str

set_serial_queue(*value*)
Configures serial queue for current host.

Parameters **value** (*HPCQueue*) – serial queue for host

set_user(*new_user*)
Sets user name :param new_user: user :type new_user: str

submit_job(*job_script*)
Submits job to scheduler and returns job id

Parameters **job_script** (*str*) – script path

Returns job id

Return type int

class autosubmit.queue.localqueue.**LocalHeader**
Class to handle the Ps headers of a job

class autosubmit.queue.localqueue.**LocalQueue**(*expid*)
Bases: autosubmit.queue.hpcqueue.HPCQueue

Class to manage jobs to localhost

Parameters **expid** (*str*) – experiment's identifier

cancel_job(*job_id*)
Cancels job

Parameters **job_id** (*int*) – job to cancel

check_host()
Checks host availability

Returns True if host is available, False otherwise

Return type bool

check_job(*job_id*)
Checks job statuts

Parameters **job_id** (*int*) – job to check

Returns current job status

Return type Status

check_remote_log_dir()
Creates log dir on remote host

close_connection()
Closes ssh connection to host

get_completed_files(*jobname*)
Copies *COMPLETED* files from remote to local

Parameters **jobname** (*str*) – name of job to check

Returns True if succesful, False otherwise

Return type bool

get_header(*job*)
Gets header to be used by the job

Parameters **job** (*Job*) – job

Returns header to use

Return type str

static get_pscall(*job_id*)
Gets command to check if a job is running given process identifier

Parameters **job_id** (*int*) – process indentifier

Returns command to check job status script

Return type str

static get_qstatjob(*job_id*)
Gets qstat command for given job id

Parameters **job_id** (*int*) – job to check

Returns qstat command for job

Return type str

get_serial_queue()
Returns serial queue for current host. If not configured, returns self

Returns serial queue for host

Return type HPCQueue

get_shcall(*job_script*)
Gets execution command for given job

Parameters **job_script** (*str*) – script to run

Returns command to execute script

Return type str

header
Header to add to jobs for scheduler cofiguration

Returns header

Return type object

send_script(*job_script*)
Send a script to remote host

Parameters **job_script** (*str*) – name of script to send

set_budget(*new_budget*)
Sets budget :param new_budget: project :type new_budget: str

set_host(*new_host*)
Sets host name :param new_host: host :type new_host: str

set_project(*new_project*)
Sets project name :param new_project: project :type new_project: str

set_remote_log_dir(*new_remote_log_dir*)
Sets remote directory for logs :param new_remote_log_dir: path to log directory :type new_remote_log_dir: str

set_scratch(*new_scratch*)
Sets scracth directory name :param new_scratch: scratch path :type new_scratch: str

set_serial_queue (*value*)
Configures serial queue for current host.

Parameters **value** (*HPCQueue*) – serial queue for host

set_user (*new_user*)
Sets user name :param new_user: user :type new_user: str

submit_job (*job_script*)
Submits job to scheduler and returns job id

Parameters **job_script** (*str*) – script path

Returns job id

Return type int

a

A

B

C

D

E

F

G

S

T

U

W