autosubmit Documentation

Release 3.4.0

Domingo Manubens - Javier Vegas

May 12, 2016

Contents

CHAPTER 1

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 Barcelona Supercomputing Centre (BSC) to run EC-Earth, NEMO and NMMB air quality model.

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

Autosubmit 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
```

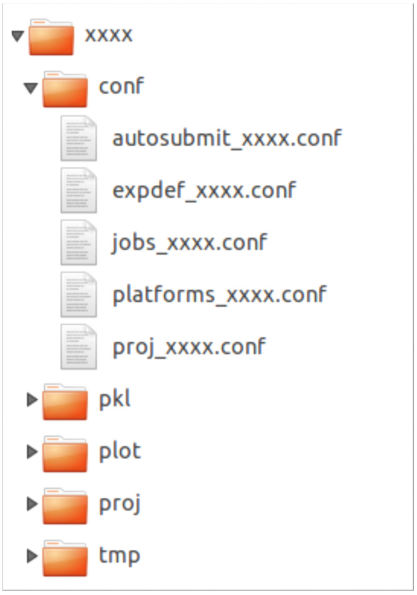
1.3.1 Experiment creation

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.



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).

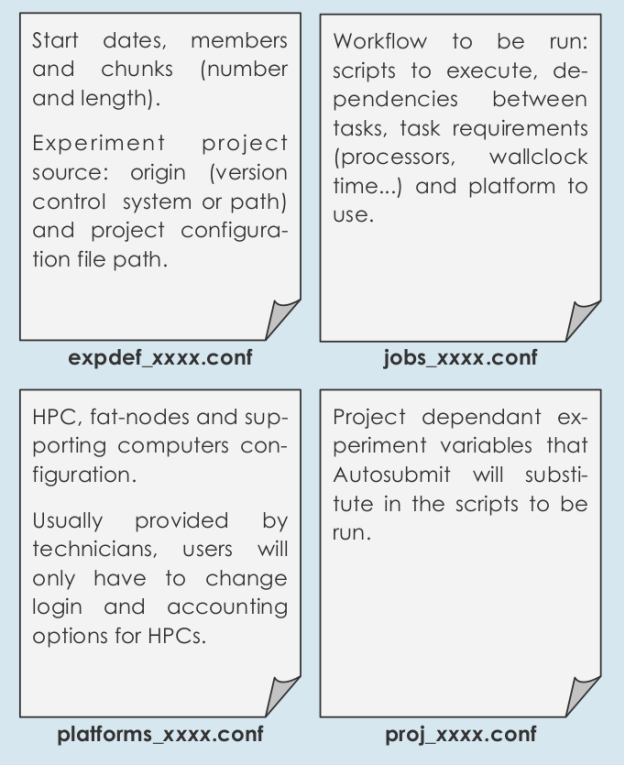
After that, you are expected to run the command:

```
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:



1.2: Configuration files content

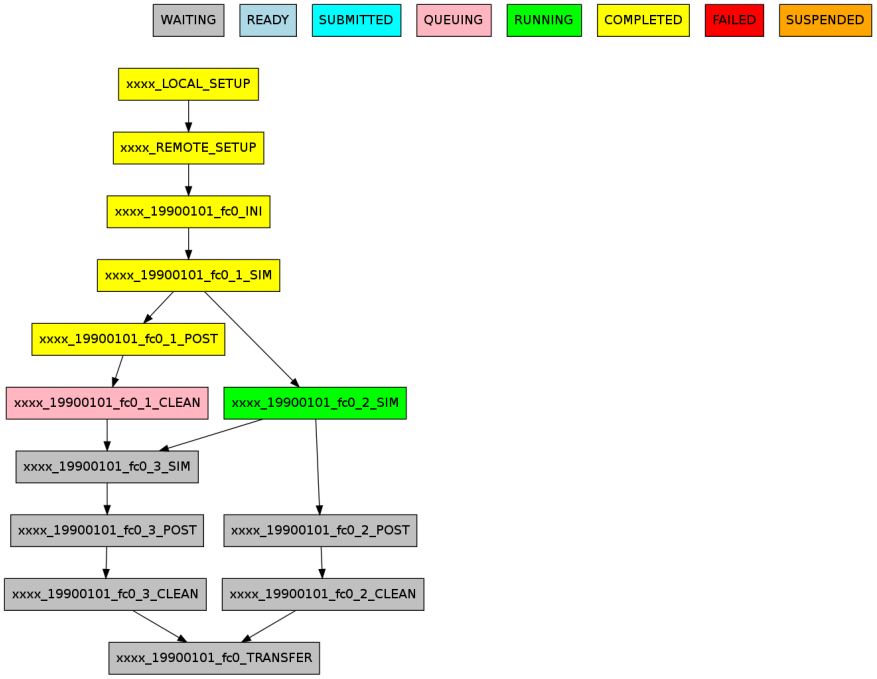
```
autosubmit run xxxx
```

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 appear 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.



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

CHAPTER 2

Tutorial

2.1 Quick start guide

2.1.1 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.

2.1.2 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.

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 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.

Examples:

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

```
[DEFAULT]
# Experiment identifier
# No need to change
EXPID = cxxx
# HPC name.
# No need to change
HPCARCH = ithaca

[experiment]
# Supply the list of start dates. Available formats: YYYYMMDD YYYYMMDDhh YYYYMMDDhhmm
# Also you can use an abreviated sintax for multiple dates with common parts:
# 200001[01 15] <=> 20000101 20000115
# DATELIST = 19600101 19650101 19700101
# DATELIST = 1960[0101 0201 0301]
DATELIST = 19900101
# Supply the list of members. LIST = fc0 fc1 fc2 fc3 fc4
MEMBERS = fc0
# Chunk size unit. STRING = hour, day, month, year
CHUNKSIZEUNIT = month
# Chunk size. NUMERIC = 4, 6, 12
CHUNKSIZE = 1
# Total number of chunks in experiment. NUMERIC = 30, 15, 10
NUMCHUNKS = 2
# Calendar used. LIST: standard, noleap
CALENDAR = standard

[rerun]
# Is a rerun or not? [Default: Do set FALSE]. BOOLEAN = TRUE, FALSE
RERUN = FALSE
# 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 =
```

```
[project]
# Select project type. STRING = git, svn, local, none
# If PROJECT_TYPE is set to none, Autosubmit self-contained dummy templates will be used
PROJECT_TYPE = git
# Destination folder name for project. type = STRING, default = leave empty,
PROJECT_DESTINATION = model

# If PROJECT_TYPE is not git, no need to change
[git]
# Repository URL  STRING = 'https://github.com/torvalds/linux.git'
PROJECT_ORIGIN = https://gitlab.cfu.local/cfu/auto-ecearth3.git
# Select branch or tag, STRING, default = 'master',
# help = {'master' (default), 'develop', 'v3.1b', ...}
PROJECT_BRANCH = develop
# type = STRING, default = leave empty, help = if model branch is a TAG leave empty
PROJECT_COMMIT =

# If PROJECT_TYPE is not svn, no need to change
[svn]
# type = STRING, help = 'https://svn.ec-earth.org/ecearth3'
PROJECT_URL =
# Select revision number. NUMERIC = 1778
PROJECT_REVISION =

# If PROJECT_TYPE is not local, no need to change
[local]
# type = STRING, help = /foo/bar/ecearth
PROJECT_PATH =

# If PROJECT_TYPE is none, no need to change
[project_files]
# Where is PROJECT CONFIGURATION file location relative to project root path
FILE_PROJECT_CONF = templates/ecearth3/ecearth3.conf
# Where is JOBS CONFIGURATION file location relative to project root path
FILE_JOBS_CONF = templates/common/jobs.conf
```

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

```
# Example job with all options specified

## Job name
# [JOBNAME]
## Script to execute. If not specified, job will be omited from workflow.
## Path relative to the project directory
# FILE =
## Platform to execute the job. If not specificied, defaults to HPCARCH in expedf file.
## LOCAL is always defined and referes to current machine
# PLATFORM =
## Queue to add the job to. If not specificied, uses PLATFORM default.
# QUEUE =
## Defines dependencies from job as a list of parents jobs separed by spaces.
## Dependencies to jobs in previous chunk, member o startdate, use -(DISTANCE)
# DEPENDENCIES = INI SIM-1 CLEAN-2
## Define if jobs runs once, once per stardate, once per member or once per chunk. Options: once, da
## If not specified, defaults to once
# RUNNING = once
## Specifies that job has only to be run after X dates, members or chunk. A job will always be create
## If not specified, defaults to 1
```

```
# FREQUENCY = 3
## On a job with FREQUENCY > 1, if True, the dependencies are evaluated against all
## jobs in the frequency interval, otherwise only evaluate dependencies against current
## iteration.
## If not specified, defaults to True
# WAIT = False
## Defines if job is only to be executed in reruns. If not specified, defaults to false.
# RERUN_ONLY = False
## Defines jobs needed to be rerun if this job is going to be rerun
# RERUN_DEPENDENCIES = RERUN INI LOCAL_SETUP REMOTE_SETUP TRANSFER
## Wallclock to be submitted to the HPC queue in format HH:MM
# WALLCLOCK = 00:05
## Processors number to be submitted to the HPC. If not specified, defaults to 1.
# PROCESSORS = 1
## Threads number to be submitted to the HPC. If not specified, defaults to 1.
# THREADS = 1
## Tasks number to be submitted to the HPC. If not specified, defaults to 1.
# TASKS = 1
## Memory requirements for the job in MB
# MEMORY = 4096
##  Number of retrials if a job fails. If not specified, defaults to the value given on experiment's
# RETRIALS = 4
## Some jobs can not be checked before running previous jobs. Set this option to false if that is the
# CHECK = False
## Select the interpreter that will run the job. Options: bash, python, r Default: bash
# TYPE = bash


[LOCAL_SETUP]
FILE = LOCAL_SETUP.sh
PLATFORM = LOCAL

[REMOTE_SETUP]
FILE = REMOTE_SETUP.sh
DEPENDENCIES = LOCAL_SETUP
WALLCLOCK = 00:05

[INI]
FILE = INI.sh
DEPENDENCIES = REMOTE_SETUP
RUNNING = member
WALLCLOCK = 00:05

[SIM]
FILE = SIM.sh
DEPENDENCIES = INI SIM-1 CLEAN-2
RUNNING = chunk
WALLCLOCK = 00:05
PROCESSORS = 2
THREADS = 1
TASKS = 1

[POST]
FILE = POST.sh
DEPENDENCIES = SIM
RUNNING = chunk
WALLCLOCK = 00:05
```

```
[CLEAN]
FILE = CLEAN.sh
DEPENDENCIES = POST
RUNNING = chunk
WALLCLOCK = 00:05

[TRANSFER]
FILE = TRANSFER.sh
PLATFORM = LOCAL
DEPENDENCIES = CLEAN
RUNNING = member
```

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

```
# Example platform with all options specified

## Platform name
# [PLAFORM]
## Queue type. Options: PBS, SGE, PS, LSF, ecaccess, SLURM
# TYPE =
## Version of queue manager to use. Needed only in PBS (options: 10, 11, 12) and ecaccess (options: p
# VERSION =
## Hostname of the HPC
# HOST =
## Project for the machine scheduler
# PROJECT =
## Budget account for the machine scheduler. If omitted, takes the value defined in PROJECT
# BUDGET =
## Option to add project name to host. This is required for some HPCs
# ADD_PROJECT_TO_HOST = False
## User for the machine scheduler
# USER =
## Path to the scratch directory for the machine
# SCRATCH_DIR = /scratch
## If true, autosubmit test command can use this queue as a main queue. Defaults to false
# TEST_SUITE = False
## If given, autosubmit will add jobs to the given queue
# QUEUE =
## If specified, autosubmit will run jobs with only one processor in the specified platform.
# SERIAL_PLATFORM = SERIAL_PLATFORM_NAME
## If specified, autosubmit will run jobs with only one processor in the specified queue.
## Autosubmit will ignore this configuration if SERIAL_PLATFORM is provided
# SERIAL_QUEUE = SERIAL_QUEUE_NAME
## Default Maximum number of jobs to be waiting in any platform queue
## Default = 3
# MAX_WAITING_JOBS = 3
## Default maximum number of jobs to be running at the same time at any platform
## Default = 6
# TOTAL_JOBS = 6

[ithaca]
# Queue type. Options: ps, SGE, LSF, SLURM, PBS, eceaccess
TYPE = SGE
HOST = ithaca
PROJECT = cfu
ADD_PROJECT_TO_HOST = true
USER = dmanubens
SCRATCH_DIR = /scratch/cfu
```

```
TEST_SUITE = True
```

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

```
[config]
# Experiment identifier
# No need to change
EXPID =
# No need to change.
# Autosubmit version identifier
AUTOSUBMIT_VERSION =
# Default maximum number of jobs to be waiting in any platform
# Default = 3
MAXWAITINGJOBS = 3
# Default maximum number of jobs to be running at the same time at any platform
# Default = 6
TOTALJOBS = 6
# Time (seconds) between connections to the HPC queue scheduler to poll already submitted jobs status
# Default = 10
SAFETYSLEEPTIME = 10
# Number of retrials if a job fails. Can ve override at job level
# Default = 0
RETRIALS = 0
```

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/`

2.1.3 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/proj_cxxx.conf
```

```
[common]
# No need to change.
MODEL = ecearth
# No need to change.
VERSION = v3.1
# No need to change.
```

```
TEMPLATE_NAME = ecearth3
# Select the model output control class. STRING = Option
# listed under the section : https://earth.bsc.es/wiki/doku.php?id=overview_outclasses
OUTCLASS = specs
# After transferring output at /cfunas/exp remove a copy available at permanent storage of HPC
# [Default: Do set "TRUE"]. BOOLEAN = TRUE, FALSE
MODEL_output_remove = TRUE
# Activate cmorization [Default: leave empty]. BOOLEAN = TRUE, FALSE
CMORIZATION = TRUE
# Essential if cmorization is activated.
# STRING =  (http://www.specs-fp7.eu/wiki/images/1/1c/SPECS_standard_output.pdf)
CMORFAMILY =
# Supply the name of the experiment associated (if there is any) otherwise leave it empty.
# STRING (with space) = seasonal r1p1, seaiceinit r?p?
ASSOCIATED_EXPERIMENT =
# Essential if cmorization is activated (Forcing). STRING = Nat,Ant (Nat and Ant is a single option)
FORCING =
# Essential if cmorization is activated (Initialization description). STRING = N/A
INIT_DESCR =
# Essential if cmorization is activated (Physics description). STRING = N/A
PHYS_DESCR =
# Essential if cmorization is activated (Associated model). STRING = N/A
ASSOC_MODEL =

[grid]
# AGCM grid resolution, horizontal (truncation T) and vertical (levels L).
# STRING = T159L62, T255L62, T255L91, T511L91, T799L62 (IFS)
IFS_resolution = T511L91
# OGCM grid resolution. STRING = ORCA1L46, ORCA1L75, ORCA025L46, ORCA025L75 (NEMO)
NEMO_resolution = ORCA025L75

[oasis]
# Coupler (OASIS) options.
OASIS3 = yes
# Number of pseduo-parallel cores for coupler [Default: Do set "7"]. NUMERIC = 1, 7, 10
OASIS_nproc = 7
# Handling the creation of coupling fields dynamically [Default: Do set "TRUE"].
# BOOLEAN = TRUE, FALSE
OASIS_flds = TRUE

[ifs]
# Atmospheric initial conditions ready to be used.
# STRING = ID found here : https://earth.bsc.es/wiki/doku.php?id=initial_conditions:atmospheric
ATM_ini =
# A different IC member per EXPID member ["PERT"] or which common IC member
# for all EXPID members ["fc0" / "fc1"]. String = PERT/fc0/fc1...
ATM_ini_member =
# Set timestep (in sec) w.r.t resolution.
# NUMERIC = 3600 (T159), 2700 (T255), 900 (T511), 720 (T799)
IFS_timestep = 900
# Number of parallel cores for AGCM component. NUMERIC = 28, 100
IFS_nproc = 640
# Coupling frequency (in hours) [Default: Do set "3"]. NUMERIC = 3, 6
RUN_coupFreq = 3
# Post-procssing frequency (in hours) [Default: Do set "6"]. NUMERIC = 3, 6
NFRP = 6
# [Default: Do set "TRUE"]. BOOLEAN = TRUE, FALSE
LCMIP5 = TRUE
```

```
# Choose RCP value [Default: Do set "2"]. NUMERIC = 0, 1=3-PD, 2=4.5, 3=6, 4=8.5
NRCP = 0
# [Default: Do set "TRUE"]. BOOLEAN = TRUE, FALSE
LHVOLCA = TRUE
# [Default: Do set "0"]. NUMERIC = 1850, 2005
NFIXYR = 0
# Save daily output or not [Default: Do set "FALSE"]. BOOLEAN = TRUE, FALSE
SAVEDDA = FALSE
# Save reduced daily output or not [Default: Do set "FALSE"]. BOOLEAN = TRUE, FALSE
ATM_REDUCED_OUTPUT = FALSE
# Store grib codes from SH files [User need to refer defined  ppt* files for the experiment]
ATM_SH_CODES =
# Store levels against "ATM_SH_CODES" e.g: level1,level2,level3, ...
ATM_SH_LEVELS =
# Store grib codes from GG files [User need to refer defined  ppt* files for the experiment]
ATM_GG_CODES =
# Store levels against "ATM_GG_CODES" (133.128, 246.128, 247.128, 248.128)
# e.g: level1,level2,level3, ...
ATM_GG_LEVELS =
# SPPT stochastic physics active or not [Default: set "FALSE"]. BOOLEAN = TRUE, FALSE
LSPPT = FALSE
# Write the perturbation patterns for SPPT or not [Default: set "FALSE"].
# BOOLEAN = TRUE, FALSE
LWRITE_ARP =
# Number of scales for SPPT [Default: set 3]. NUMERIC = 1, 2, 3
NS_SPPT =
# Standard deviations of each scale [Default: set 0.50,0.25,0.125]
# NUMERIC values separated by ,
SDEV_SPPT =
# Decorrelation times (in seconds) for each scale [Default: set 2.16E4,2.592E5,2.592E6]
# NUMERIC values separated by ,
TAU_SPPT =
# Decorrelation lengths (in meters) for each scale [Default: set 500.E3,1000.E3,2000.E3]
# NUMERIC values separated by ,
XLCOR_SPPT =
# Clipping ratio (number of standard deviations) for SPPT [Default: set 2] NUMERIC
XCLIP_SPPT =
# Stratospheric tapering in SPPT [Default: set "TRUE"]. BOOLEAN = TRUE, FALSE
LTAPER_SPPT =
# Top of stratospheric tapering layer in Pa [Default: set to 50.E2] NUMERIC
PTAPER_TOP =
# Bottom of stratospheric tapering layer in Pa [Default: set to 100.E2] NUMERIC
PTAPER_BOT =
## ATMOSPHERIC NUDGING PARAMETERS ##
# Atmospheric nudging towards reinterpolated ERA-Interim data. BOOLEAN = TRUE, FALSE
ATM_NUDGING = FALSE
# Atmospheric nudging reference data experiment name. [T255L91: b0ir]
ATM_refnud =
# Nudge vorticity. BOOLEAN = TRUE, FALSE
NUD_VO =
# Nudge divergence. BOOLEAN = TRUE, FALSE
NUD_DI =
# Nudge temperature. BOOLEAN = TRUE, FALSE
NUD_TE =
# Nudge specific humidity. BOOLEAN = TRUE, FALSE
NUD_Q =
# Nudge liquid water content. BOOLEAN = TRUE, FALSE
NUD_QL =
```

```
# Nudge ice water content. BOOLEAN = TRUE, FALSE
NUD_QI =
# Nudge cloud fraction. BOOLEAN = TRUE, FALSE
NUD_QC =
# Nudge log of surface pressure. BOOLEAN = TRUE, FALSE
NUD_LP =
# Relaxation coefficient for vorticity. NUMERIC in ]0,inf[;
# 1 means half way between model value and ref value
ALPH_VO =
# Relaxation coefficient for divergence. NUMERIC in ]0,inf[;
# 1 means half way between model value and ref value
ALPH_DI =
# Relaxation coefficient for temperature. NUMERIC in ]0,inf[;
# 1 means half way between model value and ref value
ALPH_TE =
# Relaxation coefficient for specific humidity. NUMERIC in ]0,inf[;
# 1 means half way between model value and ref value
ALPH_Q =
# Relaxation coefficient for log surface pressure. NUMERIC in ]0,inf[;
# 1 means half way between model value and ref value
ALPH_LP =
# Nudging area Northern limit [Default: Do set "90"]
NUD_NLAT =
# Nudging area Southern limit [Default: Do set "-90"]
NUD_SLAT =
# Nudging area Western limit NUMERIC in [0,360] [Default: Do set "0"]
NUD_WLON =
# Nudging area Eastern limit NUMERIC in [0,360] [Default: Do set "360"; E<W will span Greenwich]
NUD_ELON =
# Nudging vertical levels : lower level [Default: Do set "1"]
NUD_VMIN =
# Nudging vertical levels : upper level [Default: Do set to number of vertical levels]
NUD_VMAX =

[nemo]
# Ocean initial conditions ready to be used. [Default: leave empty].
# STRING = ID found here : https://earth.bsc.es/wiki/doku.php?id=initial_conditions:oceanic
OCEAN_ini =
# A different IC member per EXPID member ["PERT"] or which common IC member
# for all EXPID members ["fc0" / "fc1"]. String = PERT/fc0/fc1...
OCEAN_ini_member =
# Set timestep (in sec) w.r.t resolution. NUMERIC = 3600 (ORCA1), 1200 (ORCA025)
NEMO_timestep = 1200
# Number of parallel cores for OGCM component. NUMERIC = 16, 24, 36
NEMO_nproc = 960
# Ocean Advection Scheme [Default: Do set "tvd"]. STRING = tvd, cen2
ADVSCH = cen2
# Nudging activation. BOOLEAN = TRUE, FALSE
OCEAN_NUDGING = FALSE
# Toward which data to nudge; essential if "OCEAN_NUDGING" is TRUE.
# STRING = fa9p, s4, glorys2v1
OCEAN_NUDDATA = FALSE
# Rebuild and store restarts to HSM for an immediate prediction experiment.
# BOOLEAN = TRUE, FALSE
OCEAN_STORERST = FALSE

[ice]
# Sea-Ice Model [Default: Do set "LIM2"]. STRING = LIM2, LIM3
```

```
ICE = LIM3
# Sea-ice initial conditions ready to be used. [Default: leave empty].
# STRING = ID found here : https://earth.bsc.es/wiki/doku.php?id=initial_conditions:sea_ice
ICE_ini =
# A different IC member per EXPID member ["PERT"] or which common IC member
# for all EXPID members ["fc0" / "fc1"]. String = PERT/fc0/fc1...
ICE_ini_member =
# Set timestep (in sec) w.r.t resolution. NUMERIC = 3600 (ORCA1), 1200 (ORCA025)
LIM_timestep = 1200

[pisces]
# Activate PISCES (TRUE) or not (FALSE) [Default: leave empty]
PISCES = FALSE
# PISCES initial conditions ready to be used. [Default: leave empty].
# STRING = ID found here : https://earth.bsc.es/wiki/doku.php?id=initial_conditions:biogeochemistry
PISCES_ini =
# Set timestep (in sec) w.r.t resolution. NUMERIC = 3600 (ORCA1), 3600 (ORCA025)
PISCES_timestep = 3600
```

Finally, you can launch Autosubmit *run* in background and with `nohup` (continue running although the user who launched the process logs out).

```
nohup autosubmit run cxxx &
```

2.1.4 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 3

Installation

3.1 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
```

Hint: To check if autosubmit has been installed run `autosubmit -v`. This command will print autosubmit's current version

Hint: To read autosubmit's readme file, run `autosubmit readme`

Hint: To see the changelog, use `autosubmit changelog`

3.2 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 configure instructions after executing:

```
autosubmit configure
```

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

As Autosubmit has an email notifications feature, you have also to configure a SMTP server and an email account from where the notifications will be sent.

There is the BSC configuration by default, it can helps you as an example.

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 4

Usage

4.1 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
-archive	Clean, compress and remove from the experiments' folder a finalized experiment
-unarchive	Restores an archived experiment

4.2 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..."
```

4.3 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.

4.4 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..."
```

4.5 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

4.6 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
```

4.7 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.

4.8 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
```

4.9 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
```

Hint: Very large plots may be a problem for some pdf and image viewers. If you are having trouble with your usual monitoring tool, try using svg output and opening it with Google Chrome with the SVG Navigator extension installed.

4.10 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] [-ft] [-fp] [-o {pdf,png,ps,svg}] expid

  expid                 experiment identifier

  -h, --help            show this help message and exit
  -ft FILTER_TYPE, --filter_type FILTER_TYPE
                        Select the job type to filter the list of jobs
  -fp FILTER_PERIOD, --filter_period FILTER_PERIOD
                        Select the period of time to filter the jobs
                        from current time to the past in number of hours back
  -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
```

4.11 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.

4.12 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
```

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

```
nohup autosubmit run cxxx &
```

4.14 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.

4.15 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
  -mc, --model_conf     overwrite model conf file
  -jc, --jobs_conf      overwrite jobs conf file
```

Example:

```
autosubmit refresh cxxx
```

4.16 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.

4.17 How to add a new job

To add a new job, open the <experiments_directory>/cxxx/conf/jobs_cxxx.conf file where cxxx is the experiment identifier and add this text:

```
[new_job]
FILE = <new_job_template>
```

This will create a new job named "new_job" that will be executed once at the default platform. This job will user the template located at <new_job_template> (path is relative to project folder).

This is the minimun job definition and usually is not enough. You usually will need to add some others parameters:

- PLATFORM: allows you to execute the job in a platform of yout choice. It must be defined in the experiment's platforms.conf file or to have the value 'LOCAL' that always refer to the machine running Autosubmit
- RUNNING: defines if jobs runs only once or once per stardate, member or chunk. Options are: once, date, member, chunk
- DEPENDENCIES: defines dependencies from job as a list of parents jobs separed by spaces. For example, if 'new_job' has to wait for "old_job" to finish, you must add the line "DEPENDENCIES = old_job". For dependencies to jobs running in previous chunks, members or startdates, use -(DISTANCE). For example, for a job "SIM" waiting for the previous "SIM" job to finish, you have to add "DEPENDENCIES = SIM-1"

For jobs running in HPC platforms, usually you have to provide information about processors, wallclock times and more . To do this use:

- WALLCLOCK: wallclock time to be submitted to the HPC queue in format HH:MM
- PROCESSORS: processors number to be submitted to the HPC. If not specified, defaults to 1.
- THREADS: threads number to be submitted to the HPC. If not specified, defaults to 1.
- TASKS: tasks number to be submitted to the HPC. If not specified, defaults to 1.
- QUEUE: queue to add the job to. If not specificied, uses PLATFORM default.

There are also another, less used features that you can use:

- FREQUENCY: specifies that a job has only to be run after X dates, members or chunk. A job will always be created for the last one. If not specified, defaults to 1
- SYNCHRONIZE: specifies that a job with RUNNING=chunk, has to synchronize its dependencies chunks at a 'date' or 'member' level, which means that the jobs will be unified: one per chunk for all members or dates. If not specified, the synchronization is for each chunk of all the experiment.
- RERUN_ONLY: determines if a job is only to be executed in reruns. If not specified, defaults to false.
- RERUN_DEPENDENCIES: defines the jobs to be rerun if this job is going to be rerunned. Syntax is identical to the used in DEPENDENCIES

Example:

```
[SIM]
FILE = templates/ecearth3/ecearth3.sim
DEPENDENCIES = INI SIM-1 CLEAN-2
RUNNING = chunk
WALLCLOCK = 04:00
PROCESSORS = 1616
THREADS = 1
TASKS = 1
```

4.18 How to add a new platform

To add a new platform, open the <experiments_directory>/cxxx/conf/platforms_cxxx.conf file where cxxx is the experiment identifier and add this text:

```
[new_platform]
TYPE = <platform_type>
HOST = <host_name>
PROJECT = <project>
USER = <user>
SCRATCH = <scratch_dir>
```

This will create a platform named "new_platform". The options specified are all mandatory:

- TYPE: queue type for the platform. Options supported are PBS, SGE, PS, LSF, ecaccess and SLURM and

also the options supported by saga-python library.

- HOST: hostname of the platform
- PROJECT: project for the machine scheduler.
- USER: user for the machine scheduler
- SCRATCH_DIR: path to the scratch directory of the machine

Warning: With some platform types, Autosubmit may also need the version, forcing you to add the parameter VERSION. These platforms are PBS (options: 10, 11, 12) and ecaccess (options: pbs, loadleveler)

Some platforms may require to run serial jobs in a different queue or platform. To avoid changing the job configuration, you can specify what platform or queue to use to run serial jobs assigned to this platform:

- SERIAL_PLATFORM: if specified, Autosubmit will run jobs with only one processor in the specified platform.
- SERIAL_QUEUE: if specified, Autosubmit will run jobs with only one processor in the specified queue. Autosubmit will ignore this configuration if SERIAL_PLATFORM is provided

There are some other paramaters that you must need to specify:

- BUDGET: budget account for the machine scheduler. If omited, takes the value defined in PROJECT
- ADD_PROJECT_TO_HOST = option to add project name to host. This is required for some HPCs
- QUEUE: if given, Autosubmit will add jobs to the given queue instead of platformn's default queue
- TEST_SUITE: if true, autosubmit test command can use this queue as a main queue. Defaults to false
- MAX_WAITING_JOBS: maximum number of jobs to be waiting in this platform.
- TOTAL_JOBS: maximum number of jobs to be running at the same time in this platform.

Example:

```
[platform]
TYPE = SGE
HOST = hostname
PROJECT = my_project
ADD_PROJECT_TO_HOST = true
USER = my_user
SCRATCH_DIR = /scratch
TEST_SUITE = True
```

4.19 How to archive an experiment

To archive the experiment, use the command:

```
autosubmit archive EXPID
```

EXPID is the experiment identifier.

Warning: this command calls implicitly the clean command. Check clean command documentation.

Warning: experiment will be unusable after archiving. If you want to use it, you will need to call first the unarchive command

Options:

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

  expid                 experiment identifier

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

Example:

```
autosubmit archive cxxx
```

Hint: Archived experiment will be stored as a tar.gz file on a folder named after the year of the last COMPLETED file date. If not COMPLETED file is present, it will be stored in the folder matching the date at the time the archive command was run.

4.20 How to unarchive an experiment

To unarchive an experiment, use the command:

```
autosubmit unarchive EXPID
```

EXPID is the experiment identifier.

Options:

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

  expid                 experiment identifier

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

Example:

```
autosubmit unarchive cxxx
```

4.21 How to configure email notifications

To configure the email notifications, you have to follow two configuration steps:

1. First you have to enable email notifications and set the accounts where you will receive it.

Edit `autosubmit_cxxx.conf` in the `conf` folder of the experiment.

Hint: Remember that you can define more than one email address divided by a whitespace.

Example:

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

```
[mail]
# Enable mail notifications
# Default = False
NOTIFICATIONS = True
# Mail address where notifications will be received
TO =  jsmith@example.com  rlewis@example.com
```

2. Then you have to define for which jobs you want to be notified.

Edit `jobs_cxxx.conf` in the `conf` folder of the experiment.

Hint: You will be notified every time the job changes its status to one of the statuses defined on the parameter `NOTIFY_ON`

Hint: Remember that you can define more than one job status divided by a whitespace.

Example:

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

```
[LOCAL_SETUP]
FILE = LOCAL_SETUP.sh
PLATFORM = LOCAL
NOTIFY_ON = FAILED COMPLETED
```

CHAPTER 5

Defining the workflow

One of the most important step that you have to do when planning to use autosubmit for an experiment is the definition of the workflow the experiment will use. On this section you will learn about the workflow definition syntax so you will be able to exploit autosubmit's full potential

> **Warning:** This section is NOT intended to show how to define your jobs. Please go to Tutorial section for a comprehensive list of job options.

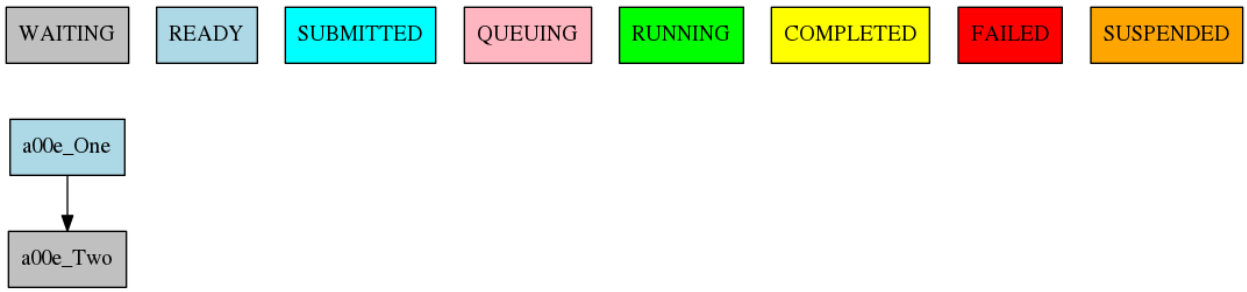
5.1 Simple workflow

The simplest workflow that can be defined it is a sequence of two jobs, with the second one triggering at the end of the first. To define it, we define the two jobs and then add a DEPENDECIES attribute on the second job referring to the first one.

It is important to remember when defining workflows that DEPENDENCIES on autosubmit always refer to jobs that should be finished before launching the job that has the DEPENDENCIES attribute.

```
[One]
FILE = one.sh

[Two]
FILE = two.sh
DEPENDENCIES = One
```

The resulting workflow can be seen on figure 5.1



5.1: Example showing a simple workflow with two sequential jobs

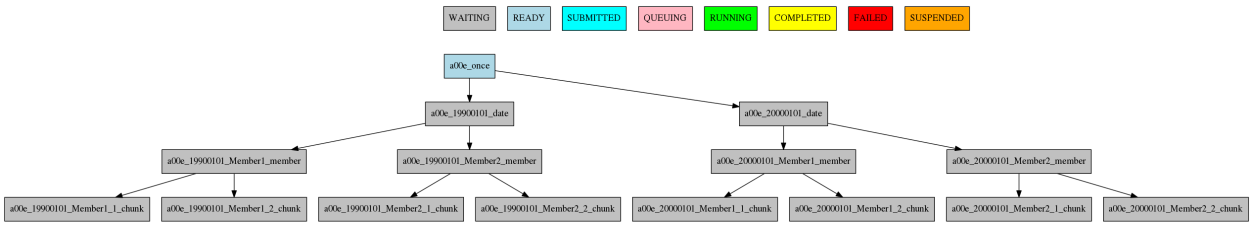
5.2 Running jobs once per startdate, member or chunk

Autosubmit is capable of running ensembles made of various startdates and members. It also has the capability to divide member execution on different chunks.

To set at what level a job has to run you have to use the RUNNING attribute. It has four posible values: once, date, member and chunk corresponding to running once, once per startdate, once per member or once per chunk respectively.

```
[once]
FILE = Once.sh

[date]
FILE = date.sh
DEPENDENCIES = once
RUNNING = date

[member]
FILE = Member.sh
DEPENDENCIES = date
RUNNING = member

[chunk]
FILE = Chunk.sh
DEPENDENCIES = member
RUNNING = chunk
```

The resulting workflow can be seen on figure 5.2 for a experiment with 2 startdates, 2 members and 2 chunks.



5.2: Example showing how to run jobs once per startdate, member or chunk.

5.3 Dependencies

Dependencies on autosubmit were introduced on the first example, but in this section you will learn about some special cases that will be very useful on your workflows.

5.3.1 Dependencies with previous jobs

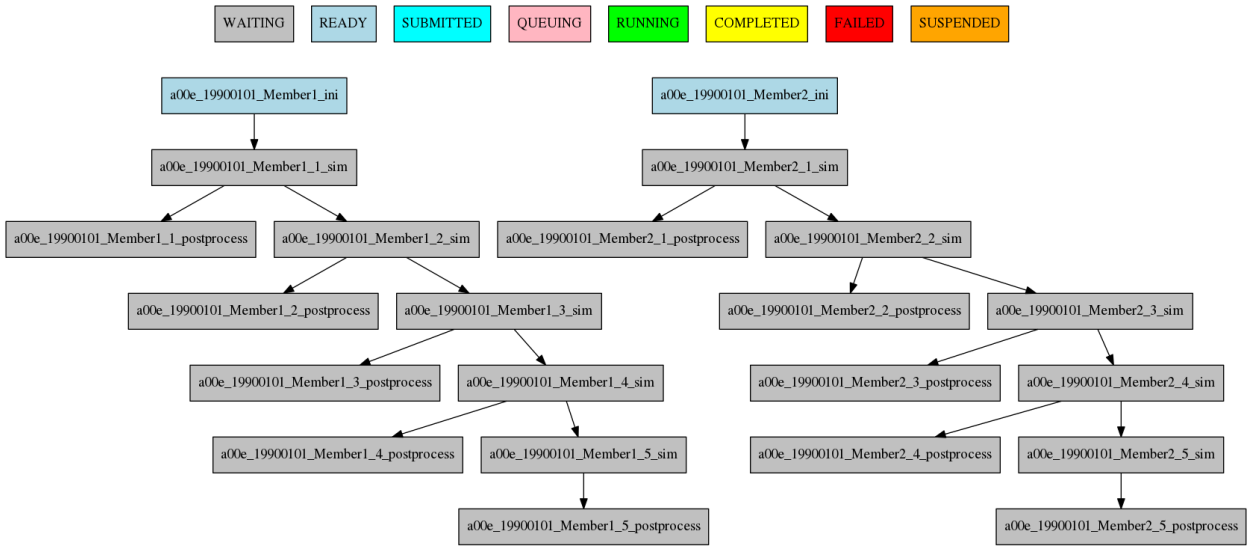
Autosubmit can manage dependencies between jobs that are part of different chunks, members or startdates. The next example will show how to make wait a simulation job for the previous chunk of the simulation. To do that, we add sim-1 on the DEPENDENCIES attribute. As you can see, you can add as much dependencies as you like separated by spaces

```
[ini]
FILE = ini.sh
RUNNING = member
```

```
[sim]
FILE = sim.sh
DEPENDENCIES = ini sim-1
RUNNING = chunk

[postprocess]
FILE = postprocess.sh
DEPENDENCIES = sim
RUNNING = chunk
```

The resulting workflow can be seen on figure 5.3

Warning: Autosubmit simplifies the dependencies, so the final graph usually does not show all the lines that you may expect to see. In this example you can see that there are no lines between the ini and the sim jobs for chunks 2 to 5 because that dependency is redundant with the one on the previous sim



5.3: Example showing dependencies between sim jobs on different chunks.

5.3.2 Dependencies between running levels

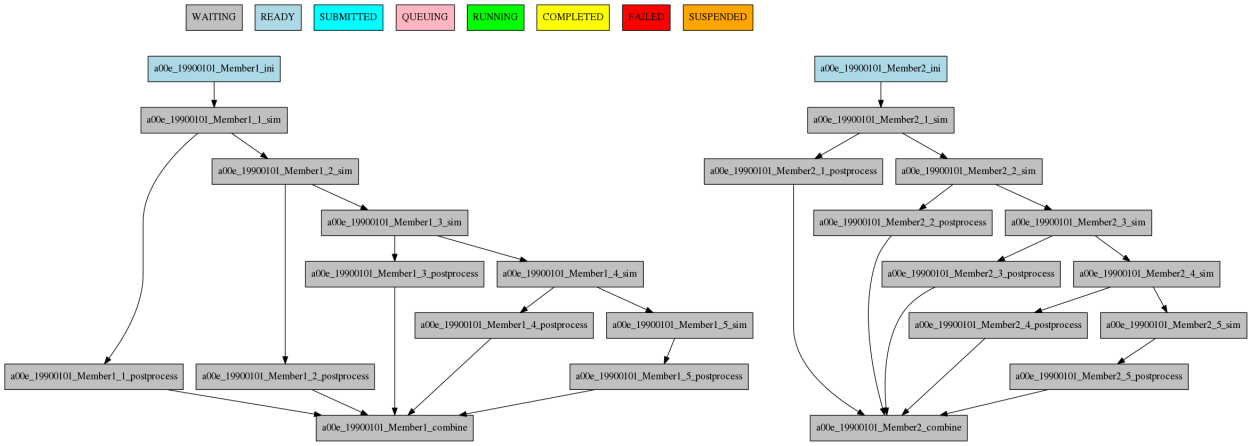
On the previous examples we have seen that when a job depends on a job on a higher level (a running chunk job depending on a member running job) all jobs wait for the higher running level job to be finished. That is the case on the ini sim dependency on the next example.

In the other case, a job depending on a lower running level job, the higher level job will wait for ALL the lower level jobs to be finished. That is the case of the postprocess combine dependency on the next example.

```
[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = ini sim-1
RUNNING = chunk
```

```
[postprocess]
FILE = postprocess.sh
DEPENDENCIES = sim
RUNNING = chunk

[combine]
FILE = combine.sh
DEPENDENCIES = postprocess
RUNNING = member
```

The resulting workflow can be seen on figure 5.4



5.4: Example showing dependencies between jobs running at different levels.

5.4 Job frequency

Some times you just don't need a job to be run on every chunk or member. For example, you may want to launch the postprocessing job after various chunks have completed. This behaviour can be achieved by using the FREQUENCY attribute. You can specify an integer I on this attribute and the job will run only once for each I iterations on the running level.

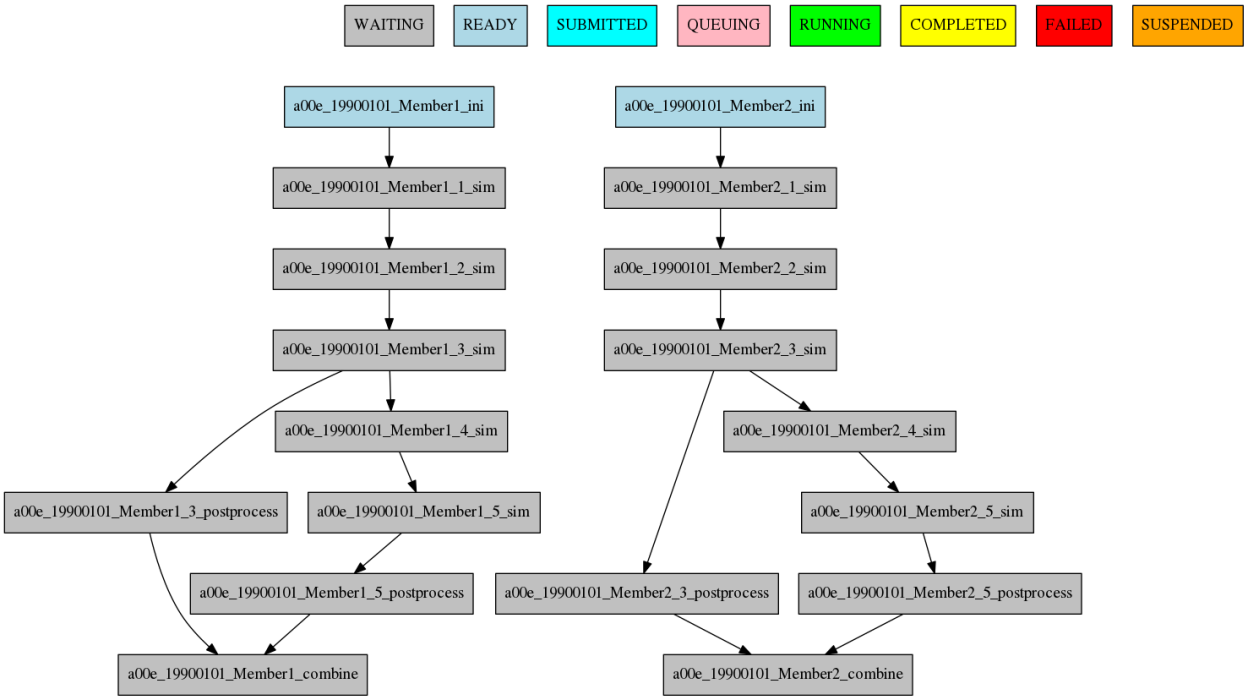
Hint: You don't need to adjust the frequency to be a divisor of the total jobs. A job will always execute at the last iteration of its running level

---

```
[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = ini sim-1
RUNNING = chunk

[postprocess]
FILE = postprocess.sh
DEPENDENCIES = sim
```

```
RUNNING = chunk
FREQUENCY = 3

[combine]
FILE = combine.sh
DEPENDENCIES = postprocess
RUNNING = member
```

The resulting workflow can be seen on figure 5.5



5.5: Example showing dependencies between jobs running at different frequencies.

5.5 Job synchronize

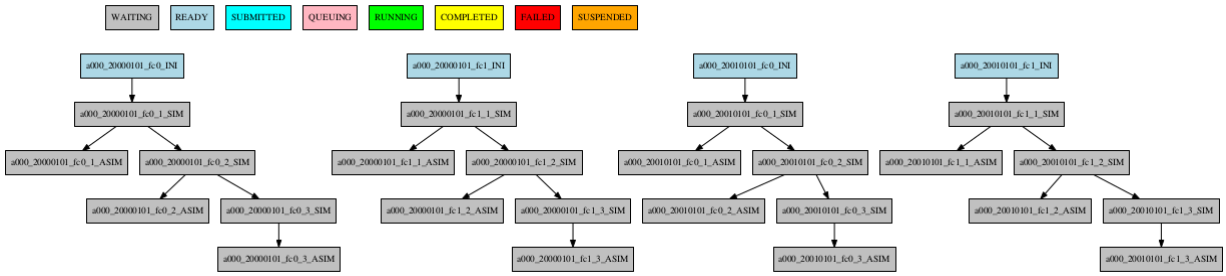
Some times when you have a job running at chunk level, and this job has dependencies, you could want not to run a job for each experiment chunk, but to run once for all member/date dependencies, maintaining the chunk granularity, in this cases you can use the SYNCHRONIZE job parameter to determine which kind of synchronization do you want. See the below examples with and without this parameter.

Hint: This job parameter was thought to work with jobs with RUNNING parameter equals to 'chunk'.

```
[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = INI SIM-1
```

```
RUNNING = chunk

[ASIM]
FILE = asim.sh
DEPENDENCIES = SIM
RUNNING = chunk
```

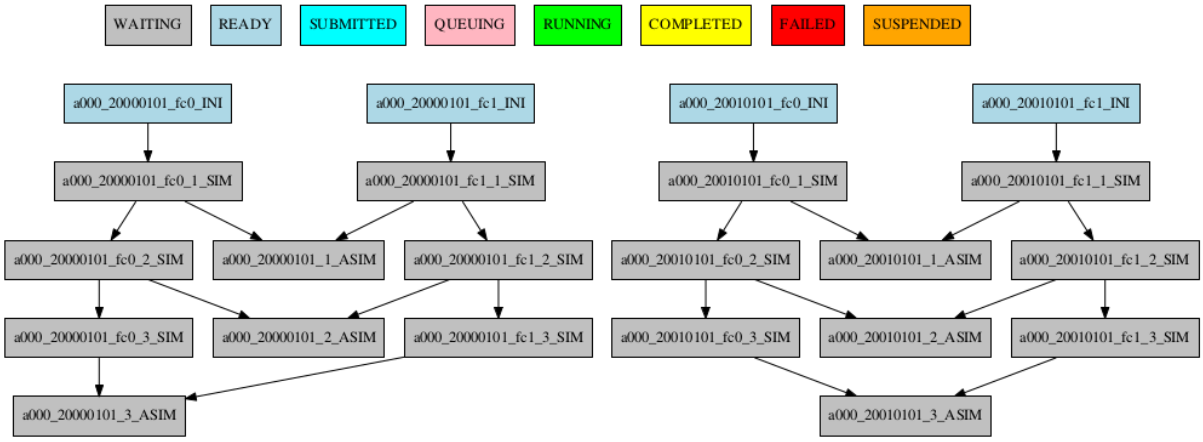
The resulting workflow can be seen on figure 5.6



5.6: Example showing dependencies between chunk jobs running without synchronize.

```
[ASIM]
SYNCHRONIZE = member
```

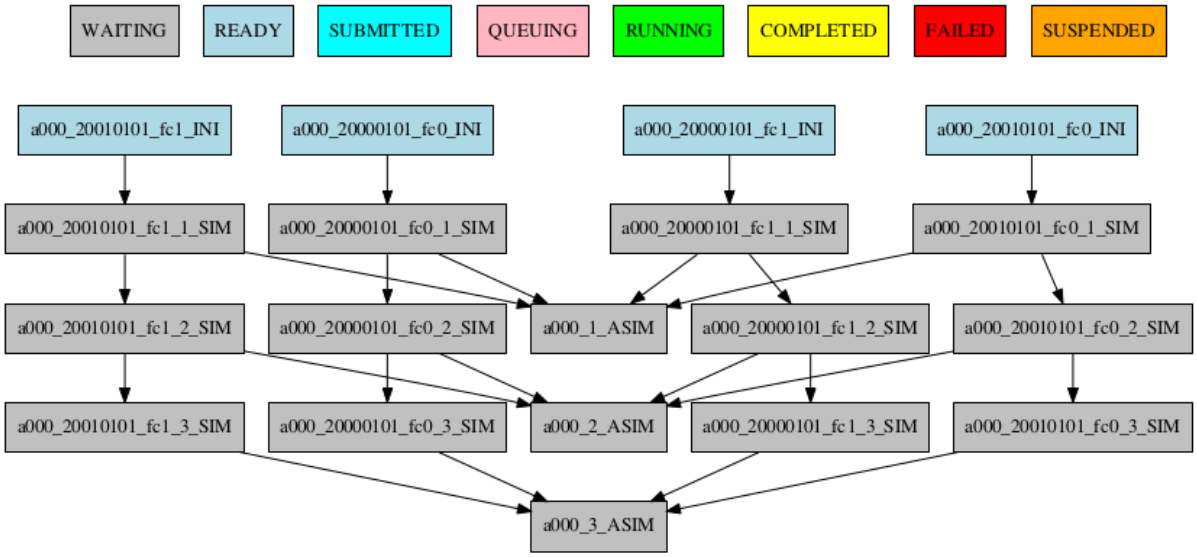
The resulting workflow of setting SYNCHRONIZE parameter to 'member' can be seen on figure 5.7



5.7: Example showing dependencies between chunk jobs running with member synchronize.

```
[ASIM]
SYNCHRONIZE = date
```

The resulting workflow of setting SYNCHRONIZE parameter to 'date' can be seen on figure 5.8



5.8: Example showing dependencies between chunk jobs running with date synchronize.

5.6 Rerun dependencies

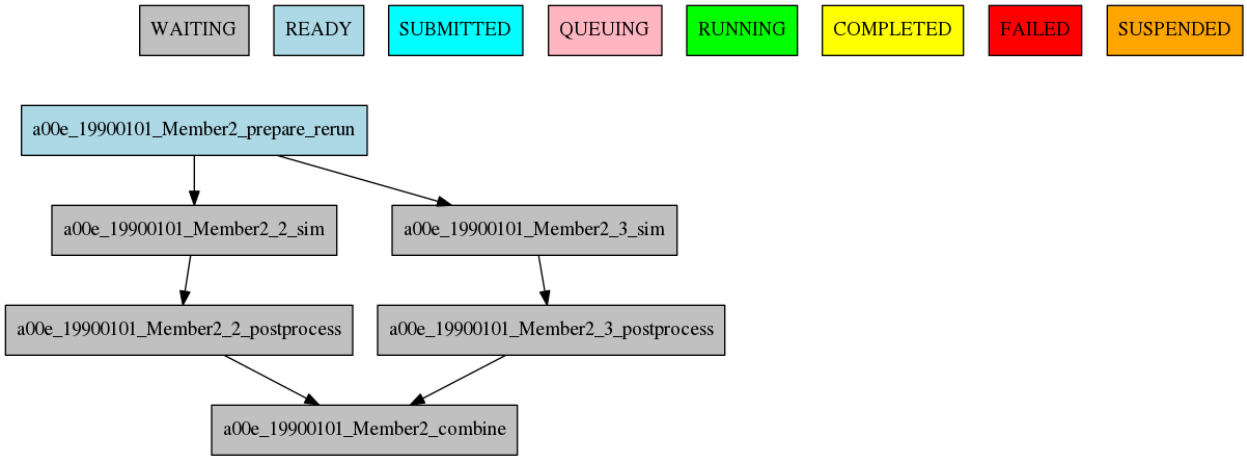
Autosubmit has the possibility to rerun some chunks of the experiment without affecting everything else. In this case, autosubmit will automatically rerun all jobs of that chunk. If some of this jobs need another one on the workflow you have to add the RERUN_DEPENDENCIES attribute and specify which jobs to rerun.

It is also usual that you will have some code that it is needed only in the case of a rerun. You can add this jobs to the workflow as usual and set the attribute RERUN_ONLY to true. This jobs will be omitted from the workflow in the normal case, but will appear on the reruns.

```
[prepare_rerun]
FILE = prepare_rerun.sh
RERUN_ONLY = true
RUNNING = member

[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = ini combine prepare_rerun
RERUN_DEPENDENCIES = combine prepare_rerun
RUNNING = chunk

[postprocess]
FILE = postprocess.sh
DEPENDENCIES = sim
RUNNING = chunk

[combine]
FILE = combine.sh
DEPENDENCIES = postprocess
```

```
RUNNING = member
```

The resulting workflow can be seen on figure 5.9 for a rerun of chunks 2 and 3 of member 2.



5.9: Example showing a rerun workflow for chunks 2 and 3.

CHAPTER 6

Troubleshooting

6.1 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 -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}
    (-fl 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
    -fl 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 -fl "cxxx_20101101_fc3_21_sim cxxx_20111101_fc4_26_sim" -t READY -s
autosubmit setstatus cxxx -fc [ 19601101 [ fc1 [1] ] ] -t READY -s
autosubmit setstatus cxxx -fs FAILED -t READY -s
autosubmit setstatus cxxx -ft TRANSFER -t SUSPENDED -s
```

This script has two mandatory arguments.

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

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

The second argument has four alternatives, the -fl, -fc, -fs and -ft; with those we can apply a filter for the jobs we want to change:

- **The -fl variable 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 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.

6.2 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
```

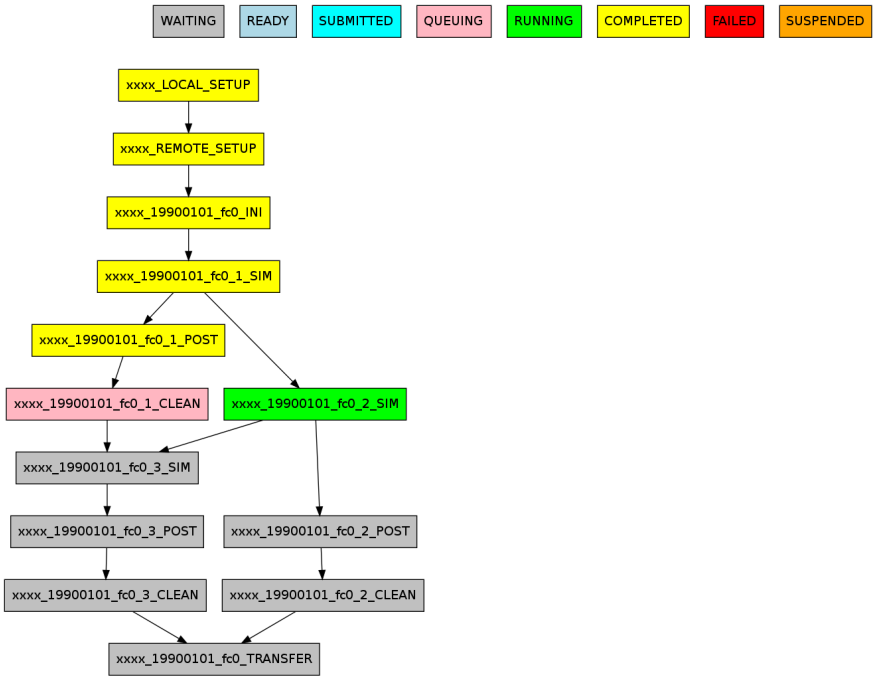
Note: A running instance of Autosubmit will check the existance of avobe file after checking already submitted jobs. It may take some time, depending on the setting `SAFETYSLEEPTIME`.

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 7

Developing a project

Autosubmit is used at BSC 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.



7.1: 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.

Warning: Autosubmit automatically adds small shell script code blocks in the header and the tailer of your scripts, to control the workflow. Please, remove any exit command in the end of your scripts, e.g. `exit 0`.

Important: For a complete reference on how to develop an EC-Earth project, please have a look in the following wiki page: https://earth.bsc.es/wiki/doku.php?id=models:models

CHAPTER 8

Variables reference

Autosubmit uses a variable substitution system to facilitate the development of the templates. This variables can be used on the template in the form %VARIABLE_NAME%.

8.1 Job variables

This variables are relatives to the current job.

- **TASKTYPE**: type of the job, as given on job configuration file.
- **JOBNAME**: current job full name.
- **FAIL_COUNT**: number of failed attempts to run this job.
- **SDATE**: current startdate.
- **MEMBER**: current member.
- **CHUNK**: current chunk.
- **DAY_BEFORE**: day before the startdate
- **Chunk_End_IN_DAYS**: chunk's length in days
- **Chunk_START_DATE**: chunk's start date
- **Chunk_START_YEAR**: chunk's start year
- **Chunk_START_MONTH**: chunk's start month
- **Chunk_START_DAY**: chunk's start day
- **Chunk_START_HOUR**: chunk's start hout
- **Chunk_END_DATE**: chunk's end date
- **Chunk_END_YEAR**: chunk's end year
- **Chunk_END_MONTH**: chunk's end month
- **Chunk_END_DAY**: chunk's end day
- **Chunk_END_HOUR**: chunk's end hour
- **PREV**: days since startdate at the chunk's start
- **Chunk_FIRST**: True if the current chunk is the first, false otherwise.
- **Chunk_LAST**: True if the current chunk is the last, false otherwise.

- **NUMPROC**: Number of processors that the job will use.
- **NUMTHREADS**: Number of threads that the job will use.
- **NUMTASK**: Number of tasks that the job will use.
- **WALLCLOCK**: Number of processors that the job will use.
- **NOTIFY_ON**: Determine the job statuses you want to be notified.

8.2 Platform variables

This variables are relative to the platoforms defined on the jobs conf. A full set of the next variables are defined for each platform defined on the platforms configuration file, substituting {PLATFORM_NAME} for each platform's name. Also, a suite of varables is defined for the current platform where {PLATFORM_NAME} is substituted by CURRENT.

- **{PLATFORM_NAME}_ARCH**: Platform name
- **{PLATFORM_NAME}_HOST**: Platform url
- **{PLATFORM_NAME}_USER**: Platform user
- **{PLATFORM_NAME}_PROJ**: Platform project
- **{PLATFORM_NAME}_BUDG**: Platform budget
- **{PLATFORM_NAME}_TYPE**: Platform scheduler type
- **{PLATFORM_NAME}_VERSION**: Platform scheduler version
- **{PLATFORM_NAME}_SCRATCH_DIR**: Platform's scratch folder path
- **{PLATFORM_NAME}_ROOTDIR**: Platform's experiment folder path

---

Hint: The variables `_USER`, `_PROJ` and `_BUDG` has no value on the LOCAL platform.

It is also defined a suite of variables for the experiment's default platform:

- **HPCARCH**: Default HPC platform name
- **HPCHOST**: Default HPC platform url
- **HPCUSER**: Default HPC platform user
- **HPCPROJ**: Default HPC platform project
- **HPCBUDG**: Default HPC platform budget
- **HPCTYPE**: Default HPC platform scheduler type
- **HPCVERSION**: Default HPC platform scheduler version
- **SCRATCH_DIR**: Default HPC platform scratch folder path
- **HPCROOTDIR**: Default HPC platform experiment's folder path

8.3 Project variables

- **NUMCHUNKS**: number of chunks of the experiment

- **CHUNKSIZE**: size of each chunk
- **CHUNKSIZEUNIT**: unit of the chuk size. Can be hour, day, month or year.
- **CALENDAR**: calendar used for the experiment. Can be standard or noleap.
- **ROOTDIR**: local path to experiment's folder
- **PROJDIR**: local path to experiment's proj folder

CHAPTER 9

Module documentation

9.1 autosubmit

9.2 autosubmit.config

9.2.1 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.

9.2.2 autosubmit.config.config_common

9.2.3 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 formatting (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 formatting (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 formatting (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 formatting (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

9.3 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

- **error_on_inexistence** (*bool*) – if True, adds an error log if experiment does not exists
- **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*, *description*, *version*, *test=False*)
Creates a new experiment by copying an existing experiment

Parameters

- **test** (*bool*) – specifies if it is a test experiment
- **version** (*str*) – experiment's associated autosubmit version
- **name** (*str*) – identifier of experiment to copy
- **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.**get_autosubmit_version**(*expid*)
Get the minimun autosubmit version needed for the experiment

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

Returns If experiment exists returns the autosubmit version for it, if not returns None

Return type str

autosubmit.database.db_common.**last_name_used**(*test=False*)
Gets last experiment identifier used

Parameters **test** (*bool*) – flag for test experiments

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

Return type str

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

Parameters

- **version** (*str*) – autosubmit version associated to the experiment
- **test** (*bool*) – flag for test experiments
- **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

Parameters **check_version** (*bool*) – If true, check if the database is compatible with this autosubmit version

Returns connection object, cursor object

Return type sqlite3.Connection, sqlite3.Cursor

9.4 autosubmit.date

9.5 autosubmit.git

9.6 autosubmit.job

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

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

class autosubmit.job.job_common.**StatisticsSnippetR**
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_common.**Type**
Class to handle the status of a job

9.7 autosubmit.monitor

9.8 autosubmit.platform

a

T

U

W