ESM Archiving Documentation

Release 4.0.0

Paul Gierz

Jun 09, 2020

Contents:

ESM Archiving gives you modern tools for putting your run on the tape

This project helps you in uploading and downloading your simulations to a tape archive. It provides the command `esm_archive` which can be used to generate, upload, and retrieve simulation archives in the form of zipped tarballs. Additionally, it provides a plug-in functionality for the `esm_runscripts` tool; allowing you to create archives while your simulations runs. As the project is written in Python, any of the functionality can also be embedded in other scripts.

CHAPTER 1

Pre-Requisites

Note that for this software to work, you need **python 3.6** or newer. For archiving on **mistral.dkrz.de** you need to have the **pigz** module loaded for parallel zipping.

CHAPTER 2

Installing

Run the following:

```
# Set up a modern git and python environment:
# Maybe needed on some machines:
module load git
# On Ollie:
module load python3
# On Mistral:
module load anaconda3
# Install the archive project:
pip install git+https://github.com/esm-tools/esm_archiving
```

CHAPTER 3

Usage

Once installed, you get the new binary `esm_archiving`. You can generate tarballs for a standard run:

```
esm_archive create /path/to/exp 1860-01-01 1870-01-01
```

Then upload to the tape server:

```
esm_archive upload /path/to/exp 1860-01-01 1870-01-01
```

For more detailed descriptions, see the documentation.

CHAPTER 4

Roadmap

There are a few issues which shows what is planned for the near future:

- Support for `hssrv2.awi.de` tape archive
- Download and unpack functionality
- Integrity checks for the tarballs

Please feel free to add to the list by opening an issue with the tag enhancement

CHAPTER 5

Benchmarked Tests

`esm_archiving` is tested against a few standard runs to ensure everything works smoothly. The table below shows which experiments are tested

Please note that currently, the benchmark run is still in production. Automatic testing will resume once the data are available.

Experiment	ESM Runscript Version	Model
`AWIESM1.1_benchmark_001`	?	AWIESM 1.1

5.1 Usage

This section describes the usage of the `esm_archiving` tool. It can be used from the command line, from other Python scripts, or as a plugin for the ESM Infrastructure.

5.1.1 Command Line Interface

After installation, you have a new command in your path:

```
esm_archive
```

Passing in the argument `--help` will show available subcommands:

```
Usage: esm_archive [OPTIONS] COMMAND [ARGS]...

  Console script for esm_archiving.

Options:
  --version              Show the version and exit.
  --write_local_config   Write a local configuration YAML file in the current
                         working directory
```

(continues on next page)

(continued from previous page)

```
  --write_config        Write a global configuration YAML file in
                        ~/.config/esm_archiving/
  --help                Show this message and exit.

Commands:
  create
  upload
```

To use the tool, you can first `create` a tar archive and then use `upload` to put it onto the tape server.

Creating tarballs

Use `esm_archive create` to generate tar files from an experiment:

```
esm_archive create /path/to/top/of/experiment start_date end_date
```

The arguments `start_date` and `end_date` should take the form `YYYY-MM-DD`. A complete example would be:

```
esm_archive create /work/ab0246/a270077/from_ba0989/AWICM/LGM_6hours 1850-01-01 1851-
↪01-01
```

The archiving tool will automatically pack up all files it finds matching these dates in the `outdata` and `restart` directories and generate logs in the top of the experiment folder. Note that the final date (1851-01-1 in this example) is **not included**. During packing, you get a progress bar indicating when the tarball is finished.

Please be aware that are size limits in place on DKRZ's tape server. Any tar files **larger than 500 Gb will be trucated**. For more information, see: https://www.dkrz.de/up/systems/hpss/hpss

Uploading tarballs

A second command `esm_archive upload` allows you to put tarballs onto to tape server at DKRZ:

```
esm_archive upload /path/to/top/of/experiment start_date end_date
```

The signature is the same as for the `create` subcommand. Note that for this to work; you need to have a properly configured `.netrc` file in your home directory:

```
$ cat ~/.netrc
machine tape.dkrz.de login a270077 password OMITTED
```

This file needs to be readable/writable **only** for you, e.g. `chmod 600`. The archiving program will then be able to automatically log into the tape server and upload the tarballs. Again, more information about logging onto the tape server without password authentication can be found here: https://www.dkrz.de/up/help/faq/hpss/how-can-i-use-the-hpss-tape-archive-without-typing-my-password-every-time-e-g-in-scripts-or-jobs

5.1.2 Library Usage

To use ESM Archiving in a project:

```
import esm_archiving
```

This gives you a few functions you can integrate into your Python programs. They are documented in the API. Perhaps immediately useful are:

- get_files_for_date_range
- sum_tar_lists
- split_list_due_to_size_limit
- pack_tarfile
- archive_mistral

5.1.3 Plugin to ESM Runs

Warning: This functionality is still under construction

The library described above can also be used as a plug-in to automatically generate and upload tarballs as the simulation runs. Still under construction. . .

5.2 Configuring ESM Archive

When run from either the command line or in library mode (note **not** as an ESM Plugin), `esm_archiving` can be configured to how it looks for specific files. The configuration file is called `esm_archiving_config`, should be written in YAML, and have the following format:

```
echam:  # The model name
    archive: # archive seperator **required**
        # Frequency specification (how often
        # a datestamp is generated to look for)
        frequency: "1M"
        # Date format specification
        date_format: "%Y%m"
```

By default, `esm_archive` looks in the following locations:

1. Current working directory
2. **Any files in the XDG Standard:** https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html

If nothing is found, the program reverts to the hard-coded defaults, found in `esm_archiving/esm_archiving/config.py`

Note: In future, it might be changed that the program will look for an experiment specific configuration based upon the path it is given during the `create` or `upload` step.

5.2.1 Generating a configuration

You can use the command line switches `--write_local_config` and `--write_config` to generate configuration files either in the current working directory, or in the global directory for your user account defined by the XDG standard (typically ~/.config/esm_archiving):

```
$ esm_archive --write_local_config
Writing local (experiment) configuration...

$ esm_archive --write_config
Writing global (user) configuration...
```

5.3 API

5.3.1 esm_archiving

esm_archiving package

Subpackages

esm_archiving.external package

Submodules

esm_archiving.external.pypftp module

esm_archiving.external.pypftp.**upload**(*source*, *destination*)

esm_archiving.external.pypftp.**download**(*source*, *destination*)

class esm_archiving.external.pypftp.**Pftp**(*username=None*, *password=None*)

Bases: `object`

HOST = 'tape.dkrz.de'

PORT = 4021

close()

cwd(*path*)
change working directory

directories(*path=None*)
gather directories at the given path

static download(*source*, *destination*)
uses pftp binary for transfering the file

exists(*path*)
check if a path exists

files(*path=None*)
gather files at the given path

is_connected()
check if the connection is still active

isdir(*pathname*)
Returns true if pathname refers to an existing directory

isfile(*pathname*)
Returns true if pathname refers to an existing file

islink(*pathname*)

listdir(*path=None*)
list directory contents

listing(*path=None*)
list directory contents

listing2(*path=None*)
directory listing in long form. similar to "ls -l"

makedirs(*path*)
Recursively create dirs as required walking up to an existing parent dir

mkdir(*path*)

mlsd(*path*)

pwd()
present working directory

quit()

reconnect()
reconnects to the ftp server

remove(*filename*)

removedirs(*path*)

rename(*from_name*, *to_name*)

rmdir(*path*)
remove directory

size(*pathname*)
Returns size of path in bytes

stat(*pathname*)
Returns stat of the path

static upload(*source*, *destination*)
uses pftp binary for transfering the file

walk(*path=None*)
recursively walk the directory tree from the given path. Similar to os.walk

walk_for_directories(*path=None*)
recursively gather directories

walk_for_files(*path=None*)
recursively gather files

Module contents

Submodules

esm_archiving.cli module

After installation, you have a new command in your path:

```
esm_archive
```

Passing in the argument `--help` will show available subcommands:

```
Usage: esm_archive [OPTIONS] COMMAND [ARGS]...

  Console script for esm_archiving.

Options:
  --version             Show the version and exit.
  --write_local_config  Write a local configuration YAML file in the current
                        working directory
  --write_config        Write a global configuration YAML file in
                        ~/.config/esm_archiving/
  --help                Show this message and exit.

Commands:
  create
  upload
```

To use the tool, you can first `create` a tar archive and then use `upload` to put it onto the tape server.

Creating tarballs

Use `esm_archive create` to generate tar files from an experiment:

```
esm_archive create /path/to/top/of/experiment start_date end_date
```

The arguments `start_date` and `end_date` should take the form `YYYY-MM-DD`. A complete example would be:

```
esm_archive create /work/ab0246/a270077/from_ba0989/AWICM/LGM_6hours 1850-01-01 1851-
↪01-01
```

The archiving tool will automatically pack up all files it finds matching these dates in the `outdata` and `restart` directories and generate logs in the top of the experiment folder. Note that the final date (1851-01-1 in this example) is **not included**. During packing, you get a progress bar indicating when the tarball is finished.

Please be aware that are size limits in place on DKRZ's tape server. Any tar files **larger than 500 Gb will be trucated**. For more information, see: https://www.dkrz.de/up/systems/hpss/hpss

Uploading tarballs

A second command `esm_archive upload` allows you to put tarballs onto to tape server at DKRZ:

```
esm_archive upload /path/to/top/of/experiment start_date end_date
```

The signature is the same as for the `create` subcommand. Note that for this to work; you need to have a properly configured `.netrc` file in your home directory:

```
$ cat ~/.netrc
machine tape.dkrz.de login a270077 password OMITTED
```

This file needs to be readable/writable **only** for you, e.g. `chmod 600`. The archiving program will then be able to automatically log into the tape server and upload the tarballs. Again, more information about logging onto the tape server without password authentication can be found here: https://www.dkrz.de/up/help/faq/hpss/how-can-i-use-the-hpss-tape-archive-without-typing-my-password-every-time-e-g-in-scripts-or-jobs

esm_archiving.config module

When run from either the command line or in library mode (note **not** as an ESM Plugin), `esm_archiving` can be configured to how it looks for specific files. The configuration file is called `esm_archiving_config`, should be written in YAML, and have the following format:

```
echam:  # The model name
    archive: # archive seperator **required**
        # Frequency specification (how often
        # a datestamp is generated to look for)
        frequency: "1M"
        # Date format specification
        date_format: "%Y%m"
```

By default, `esm_archive` looks in the following locations:

1. Current working directory
2. **Any files in the XDG Standard:** https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html

If nothing is found, the program reverts to the hard-coded defaults, found in `esm_archiving/esm_archiving/config.py`

Note: In future, it might be changed that the program will look for an experiment specific configuration based upon the path it is given during the `create` or `upload` step.

Generating a configuration

You can use the command line switches `--write_local_config` and `--write_config` to generate configuration files either in the current working directory, or in the global directory for your user account defined by the XDG standard (typically ~/.config/esm_archiving):

```
$ esm_archive --write_local_config
Writing local (experiment) configuration...

$ esm_archive --write_config
Writing global (user) configuration...
```

`esm_archiving.config.`**`load_config`**`()`

Loads the configuration from one of the default configuration directories. If none can be found, returns the hard-coded default configuration.

Returns A representation of the configuration used for archiving.

Return type dict

`esm_archiving.config.`**`write_config_yaml`**`(`*path=None*`)`

esm_archiving.esm_archiving module

This is the `esm_archiving` module.

exception `esm_archiving.esm_archiving.`**`DatestampLocationError`**

Bases: `Exception`

`esm_archiving.esm_archiving.`**`archive_mistral`**(*tfile*, *rtfile=None*)

Puts the `tfile` to the tape archive using `tape_command`

Parameters

- **`tfile`** (*`str`*) – The full path of the file to put to tape
- **`rtfile`** (*`str`*) – The filename on the remote tape server. Defaults to None, in which case a replacement is performed to keep as much of the filename the same as possible. Example: /work/ab0246/a270077/experiment.tgz –> /hpss/arch/ab0246/a270077/experiment.tgz

Returns

Return type None

`esm_archiving.esm_archiving.`**`check_tar_lists`**(*tar_lists*)

`esm_archiving.esm_archiving.`**`delete_original_data`**(*tfile*, *force=False*)

Erases data which is found in the tar file.

Parameters

- **`tfile`** (*`str`*) – Path to the tarfille whose data should be erased.
- **`force`** (*`bool`*) – If False, asks the user if they really want to delete their files. Otherwise just does this silently. Default is `False`

Returns

Return type None

`esm_archiving.esm_archiving.`**`determine_datestamp_location`**(*files*)

Given a list of files; figures where the datestamp is by checking if it varies.

Parameters **`files`** (*`list`*) – A list (longer than 1!) of files to check

Returns A slice object giving the location of the datestamp

Return type slice

Raises *DatestampLocationError :* – Raised if there is more than one slice found where the numbers vary over different files -or- if the length of the file list is not longer than 1.

`esm_archiving.esm_archiving.`**`determine_potential_datestamp_locations`**(*filepattern*)

For a filepattern, gives back index of potential date locations

Parameters **`filepattern`** (*`str`*) – The filepattern to check.

Returns A list of slice object which you can use to cut out dates from the filepattern

Return type list

`esm_archiving.esm_archiving.`**`find_indices_of`**(*char*, *in_string*)

Finds indicies of a specific character in a string

Parameters

- **`char`** (*`str`*) – The character to look for
- **`in_string`** (*`str`*) – The string to look in

Yields *int* – Each round of the generator gives you the next index for the desired character.

`esm_archiving.esm_archiving.`**`get_files_for_date_range`**(*filepattern*, *start_date*, *stop_date*, *frequency*, *date_format='%Y%m%d'*)

Creates a list of files for specified start/stop dates

Parameters

- **`filepattern`** (*`str`*) – A filepattern to replace dates in
- **`start_date`** (*`str`*) – The starting date, in a pandas-friendly date format
- **`stop_date`** (*`str`*) – Ending date, pandas friendly. Note that for end dates, you need to **add one month** to assure that you get the last step in your list!
- **`frequency`** (*`str`*) – Frequency of dates, pandas friendly
- **`date_format`** (*`str`*) – How dates should be formatted, defaults to %Y%m%d

Returns A list of strings for the filepattern with correct date stamps.

Return type list

Example

```
>>> filepattern =  "LGM_24hourly_PMIP4_echam6_BOT_mm_>>>DATE<<<.nc"
>>> get_files_for_date_range(filepattern, "1890-07", "1891-11", "1M", date_format=
↪"%Y%m")
[
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189007.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189008.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189009.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189010.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189011.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189012.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189101.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189102.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189103.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189104.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189105.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189106.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189107.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189108.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189109.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189110.nc",
]
```

`esm_archiving.esm_archiving.`**`get_list_from_filepattern`**(*filepattern*)

`esm_archiving.esm_archiving.`**`group_files`**(*top*, *filetype*)

Generates quasi-regexes for a specific filetype, replacing all numbers with #.

Parameters

- **`top`** (*`str`*) – Where to start looking (this should normally be top of the experiment)
- **`filetype`** (*`str`*) – Which files to go through (e.g. outdata, restart, etc…)

Returns A dictonary containing keys for each folder found in `filetype`, and values as lists of files with strings where numbers are replaced by #.

Return type dict

`esm_archiving.esm_archiving.`**`group_indexes`**(*index_list*)

Splits indexes into tuples of monotonically ascending values.

Parameters **`list`** – The list to split up

Returns A list of tuples, so that you can get only one group of ascending tuples.

Return type list

Example

```
>>> indexes = [0, 1, 2, 3, 12, 13, 15, 16]
>>> group_indexes(indexes)
[(0, 1, 2, 3), (12, 13), (15, 16)]
```

`esm_archiving.esm_archiving.`**`log_tarfile_contents`**(*tfile*)

Generates a log of the tarball contents

Parameters **`tfile`** (*`str`*) – The path for the tar file to generate a log for

Returns

Return type None

Warning: Note that for this function to work, you need to have write permission in the directory where the tarball is located. If not, this will probably raise an OSError. I can imagine giving the location of the log path as an argument; but would like to see if that is actually needed before implementing it...

`esm_archiving.esm_archiving.`**`pack_tarfile`**(*flist*, *wdir*, *outname*)

Creates a compressed tarball (`outname`) with all files found in `flist`.

Parameters

- **`flist`** (*`list`*) – A list of files to include in this tarball
- **`wdir`** (*`str`*) – The directory to "change" to when packing up the tar file. This will (essentially) be used in the tar command as the -C option by stripping off the beginning of the flist
- **`outname`** (*`str`*) – The output file name

Returns The output file name

Return type str

`esm_archiving.esm_archiving.`**`purify_expid_in`**(*model_files*, *expid*, *restore=False*)

Puts or restores >>>EXPID<<< marker in filepatterns

Parameters

- **`model_files`** (*`dict`*) – The model files for archiving
- **`expid`** (*`str`*) – The experiment ID to purify or restore
- **`restore`** (*`bool`*) – Set experiment ID back from the temporary marker

Returns Dictionary containing keys for each model, values for file patterns

Return type dict

`esm_archiving.esm_archiving.`**`query_yes_no`**`(`*question*, *default='yes'*`)`

Ask a yes/no question via `input()` and return their answer.

"question" is a string that is presented to the user. "default" is the presumed answer if the user just hits <Enter>.

It must be "yes" (the default), "no" or None (meaning an answer is required of the user).

The "answer" return value is True for "yes" or False for "no".

Note: Shamelessly stolen from StackOverflow It's not hard to implement, but Paul is lazy. . .

Parameters

- **`question`** (*`str`*) – The question you'd like to ask the user
- **`default`** (*`str`*) – The presumed answer for `question`. Defaults to "yes".

Returns True if the user said yes, False if the use said no.

Return type bool

`esm_archiving.esm_archiving.`**`run_command`**`(`*command*`)`

Runs `command` and directly prints output to screen.

Parameters **`command`** (*`str`*) – The command to run, with pipes, redirects, whatever

Returns **rc** – The return code of the subprocess.

Return type int

`esm_archiving.esm_archiving.`**`sort_files_to_tarlists`**`(`*model_files*, *start_date*, *end_date*, *config*`)`

`esm_archiving.esm_archiving.`**`split_list_due_to_size_limit`**`(`*in_list*, *slimit*`)`

`esm_archiving.esm_archiving.`**`stamp_filepattern`**`(`*filepattern*, *force_return=False*`)`

Transforms # in filepatterns to >>>DATE<<< and replaces other numbers back to original

Parameters

- **`filepattern`** (*`str`*) – Filepattern to get date stamps for
- **`force_return`** (*`bool`*) – Returns the list of filepatterns even if it is longer than 1.

Returns New filepattern, with >>>DATE<<<

Return type str

`esm_archiving.esm_archiving.`**`stamp_files`**`(`*model_files*`)`

Given a sttandard file dictioanry (keys: model names, values: filepattern); figures out where the date probably is, and replaces the `#` sequence with a >>>DATE<<< stamp.

Parameters **`model_files`** (*`dict`*) – Dictionary of keys (model names) where values are lists of files for each model.

Returns As the input, but replaces the filepatterns with the >>>DATE<<< stamp.

Return type dict

`esm_archiving.esm_archiving.`**`sum_tar_lists`**`(`*tar_lists*`)`

Sums up the amount of space in the tar lists dictionary

Given `tar_lists`, which is generally a dicitonary consisting of keys (model names) and values (files to be tarred), figures out how much space the **raw, uncompressed** files would use. Generally the compressed tarball will take up less space.

Parameters **`tar_lists`** (*`dict`*) – Dictionary of file lists to be summed up. Reports every sum as a value for the key of that particular list.

Returns Keys are the same as in the input, values are the sums (in bytes) of all files present within the list.

Return type dict

`esm_archiving.esm_archiving.`**`sum_tar_lists_human_readable`**(*tar_lists*)

As `sum_tar_lists` but gives back strings with human-readable sizes.

Module contents

Top-level package for ESM Archiving.

`esm_archiving.`**`archive_mistral`**(*tfile*, *rtfile=None*)

Puts the `tfile` to the tape archive using `tape_command`

Parameters

- **`tfile`** (*`str`*) – The full path of the file to put to tape
- **`rtfile`** (*`str`*) – The filename on the remote tape server. Defaults to None, in which case a replacement is performed to keep as much of the filename the same as possible. Example: /work/ab0246/a270077/experiment.tgz –> /hpss/arch/ab0246/a270077/experiment.tgz

Returns

Return type None

`esm_archiving.`**`check_tar_lists`**(*tar_lists*)

`esm_archiving.`**`delete_original_data`**(*tfile*, *force=False*)

Erases data which is found in the tar file.

Parameters

- **`tfile`** (*`str`*) – Path to the tarfille whose data should be erased.
- **`force`** (*`bool`*) – If False, asks the user if they really want to delete their files. Otherwise just does this silently. Default is `False`

Returns

Return type None

`esm_archiving.`**`determine_datestamp_location`**(*files*)

Given a list of files; figures where the datestamp is by checking if it varies.

Parameters **`files`** (*`list`*) – A list (longer than 1!) of files to check

Returns A slice object giving the location of the datestamp

Return type slice

Raises *DatestampLocationError :* – Raised if there is more than one slice found where the numbers vary over different files -or- if the length of the file list is not longer than 1.

`esm_archiving.`**`determine_potential_datestamp_locations`**(*filepattern*)

For a filepattern, gives back index of potential date locations

Parameters **`filepattern`** (*`str`*) – The filepattern to check.

Returns A list of slice object which you can use to cut out dates from the filepattern

Return type list

esm_archiving.**find_indices_of**(*char*, *in_string*)

Finds indicies of a specific character in a string

Parameters

- **char** (*str*) – The character to look for
- **in_string** (*str*) – The string to look in

Yields *int* – Each round of the generator gives you the next index for the desired character.

esm_archiving.**get_files_for_date_range**(*filepattern*, *start_date*, *stop_date*, *frequency*, *date_format='%Y%m%d'*)

Creates a list of files for specified start/stop dates

Parameters

- **filepattern** (*str*) – A filepattern to replace dates in
- **start_date** (*str*) – The starting date, in a pandas-friendly date format
- **stop_date** (*str*) – Ending date, pandas friendly. Note that for end dates, you need to **add one month** to assure that you get the last step in your list!
- **frequency** (*str*) – Frequency of dates, pandas friendly
- **date_format** (*str*) – How dates should be formatted, defaults to %Y%m%d

Returns A list of strings for the filepattern with correct date stamps.

Return type list

Example

```
>>> filepattern =  "LGM_24hourly_PMIP4_echam6_BOT_mm_>>>DATE<<<.nc"
>>> get_files_for_date_range(filepattern, "1890-07", "1891-11", "1M", date_format=
↪"%Y%m")
[
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189007.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189008.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189009.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189010.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189011.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189012.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189101.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189102.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189103.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189104.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189105.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189106.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189107.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189108.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189109.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189110.nc",
]
```

esm_archiving.**get_list_from_filepattern**(*filepattern*)

esm_archiving.**group_files**(*top*, *filetype*)

Generates quasi-regexes for a specific filetype, replacing all numbers with #.

Parameters

- **top** (*str*) – Where to start looking (this should normally be top of the experiment)
- **filetype** (*str*) – Which files to go through (e.g. outdata, restart, etc...)

Returns A dictonary containing keys for each folder found in `filetype`, and values as lists of files with strings where numbers are replaced by #.

Return type dict

`esm_archiving.`**`group_indexes`**(*index_list*)

Splits indexes into tuples of monotonically ascending values.

Parameters **list** – The list to split up

Returns A list of tuples, so that you can get only one group of ascending tuples.

Return type list

Example

```
>>> indexes = [0, 1, 2, 3, 12, 13, 15, 16]
>>> group_indexes(indexes)
[(0, 1, 2, 3), (12, 13), (15, 16)]
```

`esm_archiving.`**`log_tarfile_contents`**(*tfile*)

Generates a log of the tarball contents

Parameters **tfile** (*str*) – The path for the tar file to generate a log for

Returns

Return type None

Warning: Note that for this function to work, you need to have write permission in the directory where the tarball is located. If not, this will probably raise an OSError. I can imagine giving the location of the log path as an argument; but would like to see if that is actually needed before implementing it...

`esm_archiving.`**`pack_tarfile`**(*flist*, *wdir*, *outname*)

Creates a compressed tarball (`outname`) with all files found in `flist`.

Parameters

- **flist** (*list*) – A list of files to include in this tarball
- **wdir** (*str*) – The directory to "change" to when packing up the tar file. This will (essentially) be used in the tar command as the -C option by stripping off the beginning of the flist
- **outname** (*str*) – The output file name

Returns The output file name

Return type str

`esm_archiving.`**`purify_expid_in`**(*model_files*, *expid*, *restore=False*)

Puts or restores >>>EXPID<<< marker in filepatterns

Parameters

- **model_files** (*dict*) – The model files for archiving
- **expid** (*str*) – The experiment ID to purify or restore

- **restore** (*bool*) – Set experiment ID back from the temporary marker

Returns Dictionary containing keys for each model, values for file patterns

Return type dict

`esm_archiving.`**`sort_files_to_tarlists`**(*model_files*, *start_date*, *end_date*, *config*)

`esm_archiving.`**`split_list_due_to_size_limit`**(*in_list*, *slimit*)

`esm_archiving.`**`stamp_filepattern`**(*filepattern*, *force_return=False*)

Transforms # in filepatterns to >>>DATE<<< and replaces other numbers back to original

Parameters

- **filepattern** (*str*) – Filepattern to get date stamps for
- **force_return** (*bool*) – Returns the list of filepatterns even if it is longer than 1.

Returns New filepattern, with >>>DATE<<<

Return type str

`esm_archiving.`**`stamp_files`**(*model_files*)

Given a sttandard file dictioanry (keys: model names, values: filepattern); figures out where the date probably is, and replaces the # sequence with a >>>DATE<<< stamp.

Parameters **model_files** (*dict*) – Dictionary of keys (model names) where values are lists of files for each model.

Returns As the input, but replaces the filepatterns with the >>>DATE<<< stamp.

Return type dict

`esm_archiving.`**`sum_tar_lists`**(*tar_lists*)

Sums up the amount of space in the tar lists dictionary

Given `tar_lists`, which is generally a dicitonary consisting of keys (model names) and values (files to be tarred), figures out how much space the **raw, uncompressed** files would use. Generally the compressed tarball will take up less space.

Parameters **tar_lists** (*dict*) – Dictionary of file lists to be summed up. Reports every sum as a value for the key of that particular list.

Returns Keys are the same as in the input, values are the sums (in bytes) of all files present within the list.

Return type dict

`esm_archiving.`**`sum_tar_lists_human_readable`**(*tar_lists*)

As `sum_tar_lists` but gives back strings with human-readable sizes.

5.4 Credits

5.4.1 Development Lead

- Paul Gierz <pgierz@awi.de>

5.4.2 Contributors

None yet. Why not be the first?

5.5 History

5.5.1 0.1.0 (2020-02-28)

- Preliminary work

5.6 Indices and tables

- genindex
- modindex
- search

Python Module Index

Index