hpsspy Documentation

Release 0.7.1.dev341

Benjamin Alan Weaver

Jul 17, 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION

HPSSPy is a Python package for interacting with the HPSS tape storage system at NERSC. It is currently being developed on GitHub.

CHAPTER

TWO

REQUIREMENTS

HPSSPy assumes that the HPSS utilities hsi and htar are installed. As of 2023, these utilities are only available within the NERSC environment.

HPSSPy expects these utilities to exist in the directory `${HPSS_DIR}/bin`, so be sure the environment variable `HPSS_DIR` is defined.

CHAPTER

THREE

CONTENTS

3.1 HPSSPy Configuration

3.1.1 Introduction

The primary HPSSPy command-line program **missing_from_hpss** is configured with a JSON file. Both the JSON standard and the Python `json` library are very strict. There is a very quick way to check the validity of JSON files however:

```
python -c 'import json; j = open("config.json"); data = json.load(j); j.close()'
```

where `"config.json"` should be replaced with the name of the file to be tested.

The top-level JSON container should be an "object", equivalent to a Python `dict`. The simplest possible file that satisfies this requirement is:

```
{
}
```

Obviously, that's not very much to go on. You will need further data described below.

3.1.2 Metadata

The configuration file should contain a top-level keyword `"__config__"`. The value should itself be a `dict`, containing some important metadata:

```
{
    "__config__": {
        "root": "/global/project/projectdirs/my_project",
        "hpss_root": "/nersc/projects/my_project",
        "physical_disks": ["my_project"]
    }
}
```

root/
: The directory that contains *all* the data associated with the project.

hpss_root/
: The path on the HPSS tape system that will contain the backups.

physical_disks/
: If the data are spread across several physical disks and linked into the root path via symlinks, the various physical

disks need to be listed here. If the value is equivalent to `False`, *e.g.*, `[null, false, []]` this is means that the `"root"` disk contains all the physical data. If the value of is equivalent to a one-item list containing `os.path.basename(root)`, then this *also* means that the `"root"` disk contains all the physical data. A list of simple names generates the physical disks by substitution on the basename of the `"root"` value. More complicated configurations are possible, see *hpsspy.scan.physical_disks()*.

3.1.3 Sections

Inside the root directory, as described above, there may be several top-level directories. For the purposes of this documentation, these are called "sections" or "releases". The terms are interchangeable. Each section has configuration items that describe its structure:

```
{
    "__config__": {
        "root": "/projects/my_project",
        "hpss_root": "/hpss/projects/my_project",
        "physical_disks": ["my_project"]
    },
    "data": {
        "__exclude__": [],
        "d1": {
            "d1/batch/.*$": "d1/batch.tar",
            "d1/([^/]+\\.txt)$": "d1/\\1",
            "d1/templates/[^/]+$": "d1/templates/templates_files.tar"
        }
    }
}
```

The **missing_from_hpss** command works on one section at a time. The name of the section is passed on the command-line:

```
missing_from_hpss config.json data
```

This would read the `"data"` section above.

Each section should have an `"__exclude__"` keyword, whose value is a list of files to be ignored. In the example above, in order to ignore the file `/projects/my_project/data/d1/README.html`, the `"__exclude__"` value would be `["d1/README.html"]`. Note that this is relative to the path `/projects/my_project/data`, since `"data"` is the section being processed. Generally, this should only be used for a handful of top-level files, like README files. For more precise exclusion, see the `"EXCLUDE"` statement below.

In the special case where a section contains only files, and no subdirectories, the special pseudo-subdirectory `"__top__"` can be used to contain the configuration.

3.1.4 Mapping File Names to HPSS Archives

Within a section, each immediate subdirectory should be described with a keyword in the configuration file. **missing_from_hpss** will complain if not, but it won't necessarily cause it to fail. In the example above, `/projects/my_project/data/d1` is configured.

There are many possible ways to bundle files for archiving. Generally you want to make archives as large as possible, without spilling onto multiple tapes. However, with highly structured, deeply-nested directory structures, this isn't always the best way to do it from a data *retrieval* viewpoint.

Consider this scenario. `/projects/my_project/data` has been archived to ten tape archives called `data00.tar`, `data01.tar`, ... `data09.tar`. The file `/projects/my_project/data/d1/templates/d1_template_05.fits` needs to be recovered. Which tape archive contains it?

Now consider the scenario where the files in `/projects/my_project/data/d1/templates` have been archived to `/hpss/projects/my_project/data/d1/templates/d1_templates_files.tar`. Now is it easier to recover the file?

One should still try to make archives as big as possible, but generally speaking, long-term archiving of large, complex data sets should be done by **someone who actually knows the structure of the data set** .

In coding terms we describe a portion of a directory tree hierarchy using regular expressions to match *files* in that portion. Then we map files that match that regular expression to tape archive files.

Finally, it should be noted that the configuration of each section is organized by subdirectory in order to speed up the process of mapping files to backup files. Instead of looking through every possible configuration of files, only the configurations in a subdirectory need to be considered when examining files in that subdirectory.

3.1.5 Regular Expression Details

The HPSSPy package, and **missing_from_hpss** will validate the regular expressions used in the configuration file, in addition to checking the overall validity of the JSON file itself. That is, a bad regular expression will be rejected before it has any chance to "touch" any real data.

The regular expressions should follow Python's conventions, described in `re`. In addition to those conventions, this package imposes some additional requirements, conventions and idioms:

- Requirements
  - Backslashes must be escaped in JSON files. For example the metacharacter (match a single decimal digit) `\d` becomes `\\d`.
  - Regular expressions should end with the end-of-line marker `$`.
- Conventions
  - Any archive file name ending in `.tar` is assumed to be an HTAR file, and that command will be used to construct it.
  - Any archive file *not* ending in `.tar` will simply be copied to HPSS as is.
  - The special string `"EXCLUDE"` can be used to prevent backups of parts of a directory tree that might otherwise be archival. For example, `"d1/data/preproc/.*$" : "EXCLUDE"` would prevent the `preproc` directory from being backed up, even if other parts of `d1/data` were configured for backup.
  - The special string `"AUTOMATED"` behaves the same way as `"EXCLUDE"`, but is a human-readable way to denote data sets that are backed up by automation independently of **missing_from_hpss**, as opposed to not being backed up at all.
  - When constructing an archive file, **missing_from_hpss** will obtain the directory it needs to archive from the name of the *archive* file, not the regular expression itself. This is because regular expression *substition*

is performed on the archive file name. For example `batch.tar` means "archive a batch/ directory". For longer file names, any "prefix" of the file name will be stripped off, and the "suffix" of the file will be used. For example, `d1/data_d1_batch.tar` also means "archive a batch/ directory", because `data_d1_` recognized as a prefix and stripped off. In particular, this allows directory names to contain underscores.

  - An archive filename that ends with `_files.tar`, *e.g.* `foo/bar_files.tar` is a signal to **`missing_from_hpss`** to construct the archive file in a certain way, not by descending into a directory, but by constructing an explicit list of files and building an archive file out of that.

- Idioms
  - Archive the entire contents of a directory into a single file: `"foo/.*$" : "foo.tar"`.
  - Archive several subdirectories of a directory, each into their own file: `"foo/(bar|baz|flub)/.*$" : "foo/foo_\\1.tar"`. The name of the directory matched in parentheses will be substituted into the file name.
  - Archive arbitrary subdirectories of a *set* of subdirectories: `"d1/foo/(ab|bc|cd|de|ef)/([^/]+)/.*$" : "d1/foo/\\1/d1_foo_\\1_\\2.tar"`
  - Match files in a directory, but not any files in any subdirectory: `"foo/[^/]+$" : "foo_files.tar"`. See also the `_files.tar` convention mentioned above.
  - Group some but not all subdirectories in a directory into a single archive file for efficiency: `"foo/([0-9])([0-9][0-9])/.*$" : "foo/foo_\\1XX.tar"`. Note the ending of the archive file, and that the directories have to have a very uniform naming convention (three and only three digits in this example). Also, the placeholder `X` needs to be at the *end* of the file name.
  - Do not create an archive file, just copy the file, as is, to HPSS: `"d1/README\\.txt$" : "d1/README.txt"`. Similarly, for a set of TXT files: `"d1/([^/]+\\.txt)$" : "d1/\\1"`.
  - An example with lots of substitutions:

```
"d1/foo/([0-9a-zA-Z_-]+)/sub-([0-9]+)/([0-9]+)/.*$" : "d1/foo/\\1/spectra-\\2/\\
↪1_spectra-\\2_\\3.tar"
```

Finally, for truly monumentally-complicated directory trees, there is a JSON file included with this distribution describing the SDSS data tree that can be used for examples. To view the equivalent files and directories for section `"dr12"`, for example, visit https://data.sdss.org/sas/dr12.

3.2 Using HPSSPy

3.2.1 Introduction

The primary *command-line* interface to HPSSPy is the script **`missing_from_hpss`**, which is automatically generated by the package install process. If you need to generate this script manually, it is equivalent to:

```
#!/usr/bin/env python
from sys import exit
from hpsspy.scan import main
exit(main())
```

3.2.2 Options

There are several of command-line options. `missing_from_hpss --help` will display all of them. Just the short versions of the commands are shown here.

-c DIR	Cache files (described below) are written to `$HOME/cache` by default. This option allows the user to choose any directory.
-D	Delete and recreate the disk cache file (described below).
-E	Exit if an error is detected while processing files on disk or on HPSS.
-H	Delete and recreate the HPSS cache file (described below).
-l N	Limit archive files to this size in GB. The default is 1024 GB (1 TB).
-p	Issue the HPSS commands necessary to actually back up the files found that need to be backed up.
-r N	Issue a progress report on how many files have been analyzed after `N` files (default 10,000).
-t	Test mode. Try not to make any changes. Also pretend that there are no files backed up to HPSS.
-v	Print *lots* of extra information.
--version	Print a version string and exit.

Besides the options described above, **`missing_from_hpss`** requires two positional arguments:

```
missing_from_hpss config.json section
```

The two arguements are the path to a configuration file and a section of that file to process. These are extensively described in the *configuration document*.

3.2.3 Cache Files

`missing_from_hpss` uses a few cache files primarily to reduce memory footprint. These files will be stored in `$HOME/cache` by default. The files are:

Disk Cache
: A CSV file of the form `disk_cache_<section>.csv`, where `<section>` is the section (as defined above) specified on the command-line. The columns are file name, file size in bytes and modification time.

HPSS Cache
: A CSV file of the form `hpss_cache_<section>.csv`, where `<section>` is the section (as defined above) specified on the command-line. The columns are file name, file size in bytes and modification time.

Missing File Cache
: A JSON file of the form `missing_files_<section>.json`, where `<section>` is the section (as defined above) specified on the command-line. It contains a map of HPSS archive files to the files that belong in that archive. In addition the size of the resulting files (modulo small overheads from the archive file creation process) will be saved to this file.

These files are *not* cleaned up by default because they are very useful for debugging purposes.

3.2.4 Testing and Quality Assurance

To test a configuration file just run **missing_from_hpss** with the `--test` option as described above. Aside from creating cache files in a directory as described above, this mode will not alter any of the data, neither on disk nor on HPSS.

In addition to validating JSON files and regular expressions, as described in the *configuration document*, **missing_from_hpss** will:

1. Make sure all regular expressions are actually used.
2. Make sure all files actually match *one and only one* regular expression.
3. Create a manifest file containing the actual files on disk matched and the archive file they map to. This is one and the same as the "Missing File Cache" described above.
4. Make sure that all archive file sizes are less than a user-defined limit (default 1 TB), configurable on the command-line.

3.2.5 HPSSPy Library

For programmatic access to HPSS, the *HPSSPy library* provides equvalents of `os` and `os.path` that operate on the HPSS filesystem.

3.3 HPSSPy API

3.3.1 hpsspy

Python interface to the HPSS system.

exception hpsspy.**HpssError**

Generic exception class for HPSS Errors.

exception hpsspy.**HpssOSError**

HPSS Errors that are similar to OSError.

3.3.2 hpsspy.os

Reproduces some features of the Python built-in `os`.

hpsspy.os.**chmod**(*path*, *mode*)

Reproduces the behavior of `os.chmod()` for HPSS files.

Parameters

- **path** (`str`) – File to chmod.
- **mode** (`str` or `int`) – Desired file permissions. This mode will be converted to a string.

Raises

HpssOSError – If the underlying **hsi** reports an error.

hpsspy.os.**listdir**(*path*)

List the contents of an HPSS directory, similar to `os.listdir()`.

Parameters
path (`str`) – Directory to examine.

Returns
A list of *HpssFile* objects.

Return type
`list`

Raises
HpssOSError – If the underlying **hsi** reports an error.

hpsspy.os.**lstat**(*path*)

Perform the equivalent of `os.lstat()` on the HPSS file *path*.

Parameters
path (`str`) – Path to file or directory.

Returns
An object that contains information similar to the data returned by `os.stat()`.

Return type
HpssFile

Raises
HpssOSError – If the underlying **hsi** reports an error.

hpsspy.os.**makedirs**(*path*, *mode=None*)

Reproduces the behavior of `os.makedirs()`.

Parameters

- **path** (`str`) – Directory to create.
- **mode** (`str`, optional) – String representation of the octal directory mode.

Raises
HpssOSError – If the underlying **hsi** reports an error.

Notes

Unlike `os.makedirs()`, attempts to create existing directories raise no exception.

hpsspy.os.**mkdir**(*path*, *mode=None*)

Reproduces the behavior of `os.mkdir()`.

Parameters

- **path** (`str`) – Directory to create.
- **mode** (`str`, optional) – String representation of the octal directory mode.

Raises
HpssOSError – If the underlying **hsi** reports an error.

Notes

Unlike os.mkdir(), attempts to create existing directories raise no exception.

hpsspy.os.**stat**(*path*, *follow_symlinks=True*)

Perform the equivalent of os.stat() on the HPSS file *path*.

Parameters

- **path** (str) – Path to file or directory.
- **follow_symlinks** (bool, optional) – If False, makes *stat()* behave like os.lstat().

Returns
An object that contains information similar to the data returned by os.stat().

Return type
HpssFile

Raises
HpssOSError – If the underlying **hsi ls** reports an error.

hpsspy.os.**walk**(*top*, *topdown=True*, *onerror=None*, *followlinks=False*)

Traverse a directory tree on HPSS, similar to os.walk().

Parameters

- **top** (str) – Starting directory.
- **topdown** (bool, optional) – Direction to traverse the directory tree.
- **onerror** (*callable, optional*) – Call this function if an error is detected.
- **followlinks** (bool, optional) – If True symlinks to directories are treated as directories.

Returns
This function can be used in the same way as os.walk().

Return type
iterable

3.3.3 hpsspy.os.path

Reproduces some features of the Python built-in os.path.

hpsspy.os.path.**isdir**(*path*)

Reproduces the behavior of os.path.isdir() for HPSS files.

Parameters
path (str) – Path to the file.

Returns
True if *path* is a directory.

Return type
bool

hpsspy.os.path.**isfile**(*path*)

Reproduces the behavior of os.path.isfile() for HPSS files.

Parameters
path (str) – Path to the file.

Returns
True if *path* is a file.

Return type
bool

hpsspy.os.path.**islink**(*path*)

Reproduces the behavior of os.path.islink() for HPSS files.

Parameters
path (str) – Path to the file.

Returns
True if *path* is a symlink.

Return type
bool

3.3.4 hpsspy.scan

Functions for scanning directory trees to find files in need of backup.

hpsspy.scan.**_options**()

Parse command-line options.

Returns
The parsed command-line arguments.

Return type
argparse.Namespace

hpsspy.scan.**compile_map**(*old_map*, *section*)

Compile the regular expressions in a map.

Parameters

- **old_map** (dict) – A dictionary containing regular expressions to compile.
- **section** (str) – An initial key to determine the section of the dictionary of interest. Typically, this will be a top-level directory.

Returns
A new dictionary containing compiled regular expressions.

Return type
dict

hpsspy.scan.**extract_directory_name**(*filename*)

Extract a directory name from a HTAR *filename* that may contain various prefixes.

Parameters
filename (str) – Name of HTAR file, including directory path.

Returns
Name of a directory.

Return type
str

hpsspy.scan.**files_to_hpss**(*hpss_map_cache*, *section*)

Create a map of files on disk to HPSS files.

Parameters

- **hpss_map_cache** (str) – Data file containing the map.
- **section** (str) – An initial key to determine the section of the dictionary of interest. Typically, this will be a top-level directory.

Returns
A tuple contiaining the compiled mapping and an additional configuration dictionary.

Return type
tuple()

hpsspy.scan.**find_missing**(*hpss_map*, *hpss_files*, *disk_files_cache*, *missing_files*, *report=10000*, *limit=1024.0*)

Compare HPSS files to disk files.

Parameters

- **hpss_map** (dict) – A mapping of file names to HPSS files.
- **hpss_files** (dict) – The list of actual HPSS files.
- **disk_files_cache** (str) – Name of the disk cache file.
- **missing_files** (str) – Name of the file that will contain the list of missing files.
- **report** (int, optional) – Print an informational message when N files have been scanned.
- **limit** (float, optional) – HPSS archive files should be smaller than this size (in GB).

Returns
True if no serious problems were found.

Return type
bool

hpsspy.scan.**iterrsplit**(*s*, *c*)

Split string *s* on *c* and rejoin on *c* from the end of *s*.

Parameters

- **s** (str) – String to split
- **c** (str) – Split on this string.

Returns
Iteratively return the joined parts of *s*.

Return type
str

hpsspy.scan.**main**()

Entry-point for command-line scripts.

Returns
An integer suitable for passing to sys.exit().

Return type
int

hpsspy.scan.**physical_disks**(*release_root*, *config*)

Convert a root path into a list of physical disks containing data.

Parameters

- **release_root** (str) – The "official" path to the data.
- **config** (dict) – A dictionary containing path information.

Returns

A tuple containing the physical disk paths.

Return type

tuple()

hpsspy.scan.**process_missing**(*missing_cache*, *disk_root*, *hpss_root*, *dirmode='2770'*, *test=False*)

Convert missing files into HPSS commands.

Parameters

- **missing_cache** (str) – Name of a JSON file containing the missing file data.
- **disk_root** (str) – Missing files are relative to this root on disk.
- **hpss_root** (str) – Missing files are relative to this root on HPSS.
- **dirmode** (str, optional) – Create directories on HPSS with this mode (default drwxrws---).
- **test** (bool, optional) – Test mode. Try not to make any changes.

hpsspy.scan.**scan_disk**(*disk_roots*, *disk_files_cache*, *overwrite=False*)

Scan a directory tree on disk and cache the files found there.

Parameters

- **disk_roots** (list) – Name(s) of a directory in which to start the scan.
- **disk_files_cache** (str) – Name of a file to hold the cache.
- **overwrite** (bool, optional) – If True, ignore any existing cache files.

Returns

Returns True if the cache is populated and ready to read.

Return type

bool

hpsspy.scan.**scan_hpss**(*hpss_root*, *hpss_files_cache*, *overwrite=False*)

Scan a directory on HPSS and return the files found there.

Parameters

- **hpss_root** (str) – Name of a directory in which to start the scan.
- **hpss_files_cache** (str) – Name of a file to hold the cache.
- **overwrite** (bool, optional) – If True, ignore any existing cache files.

Returns

The set of files found on HPSS, with size and modification time.

Return type

dict

hpsspy.scan.**validate_configuration**(*config*)

Check the configuration file for validity.

Parameters
config (str) – Name of the configuration file.

Returns
An integer suitable for passing to sys.exit().

Return type
int

3.3.5 hpsspy.util

Low-level utilities.

class hpsspy.util.**HpssFile**(**args*)

This class is used to store and access an HPSS file's metadata.

Parameters
args (*iterable*) – This object this will normally be initialized by a tuple produced by *hpsspy.os.listdir()*.

hpss_path

Path on the HPSS filesystem.

Type
str

raw_type

Raw type string.

Type
str

raw_permission

Raw permission string.

Type
str

st_nlink

Number of hard links.

Type
int

st_uid

Owner's name.

Type
str

st_gid

Group name.

Type
str

st_size

File size in bytes.

Type
int

raw_dow

Day-of-week of modification time.

Type
str

raw_month

Month of modification time.

Type
str

raw_day

Day of modification time.

Type
int

raw_hms

H:M:S of modification time.

Type
str

raw_year

Year of modification time.

Type
int

raw_name

Name of file.

Type
str

ishtar

`True` if the file is an htar file.

Type
bool

htar_contents()

Return (and cache) the contents of an htar file.

Returns
List containing the contents.

Return type
list

property isdir

`True` if the file is a directory or a symbolic link that points to a directory.

property islink

True if the file is a symbolic link.

property name

Name of the file.

property path

Full path to the file.

property readlink

Destination of symbolic link.

property st_mode

File permission mode.

property st_mtime

File modification time.

hpsspy.util.**get_hpss_dir**()

Return the directory containing HPSS commands.

Returns
Full path to the directory containing HPSS commands.

Return type
str

Raises
KeyError – If the HPSS_DIR environment variable has not been set.

hpsspy.util.**get_tmpdir**(***kwargs*)

Return the path to a suitable temporary directory.

Resolves the path to the temporary directory in the following order:

1. If tmpdir is present as a keyword argument, the value is returned.
2. If TMPDIR is set, its value is returned.
3. If neither are set, /tmp is returned.

Parameters
kwargs (dict) – Keyword arguments from another function may be passed to this function. If tmpdir is present as a key, its value will be returned.

Returns
The name of a temporary directory.

Return type
str

hpsspy.util.**hsi**(**args*, ***kwargs*)

Run **hsi** with arguments.

Parameters

- **args** (tuple()) – Arguments to be passed to **hsi**.
- **tmpdir** (str, optional) – Write temporary files to this directory. Defaults to the value returned by *hpsspy.util.get_tmpdir()*. This option must be passed as a keyword!

Returns
The standard output from **hsi**.

Return type
str

Raises
KeyError – If the HPSS_DIR environment variable has not been set.

hpsspy.util.**htar**(**args*)

Run **htar** with arguments.

Parameters
args (tuple()) – Arguments to be passed to **htar**.

Returns
The standard output and standard error from **htar**.

Return type
tuple()

Raises
KeyError – If the HPSS_DIR environment variable has not been set.

3.4 Release Notes

3.4.1 0.7.1 (unreleased)

- No changes yet.

3.4.2 0.7.0 (2023-07-17)

- **missing_from_hpss** will proceed through all stages, even if serious errors are detected, to facilitate batch processing. The older behavior can be enabled with `--exit-on-error` (PR #15).
- Fix error handling for a variety of corner cases (PR #15).
- Increase test coverage to 100% (PR #15).

3.4.3 0.6.1 (2022-05-20)

- Bumped version due to malformed PyPI upload.

3.4.4 0.6.0 (2022-05-20)

- Reorganization of package structure and metadata; no changes to user-facing API.
- Support full-precision timestamps on HPSS files (PR #14).

3.4.5 0.5.1 (2019-08-20)

- Unused patterns and over-large backup files no longer trigger a critical error (PR #12).

3.4.6 0.5.0 (2019-05-18)

This release drops support for Python 2.

- Remove all Python 2 code (PR #8).
- Support fine-grained exclusion in configuration files (PR #10).
- Avoid commonly-used names for metadata in configuration files (PR #10).
- Detect newer files on disk that map to older HPSS files (PR #10).
- Allow top-level directories to contain only files (PR #10).

3.4.7 0.4.2 (2019-01-29)

- Further fixes for mapping HTAR file names back to directories (PR #6).

3.4.8 0.4.1 (2019-01-16)

- Handle directory names that contain underscore characters; improve test coverage (PR #4).

3.4.9 0.4.0 (2017-08-10)

- Add `--version` option.
- Add Python 3.6, remove 3.3.
- Add many quality-assurance checks and additional documentation (PR #2).

3.4.10 0.3.0 (2017-01-18)

- General refresh of Python code, documentation, test suite. However, no major changes to the API.
- Command-line inputs are no longer rigidly restricted to SDSS or DESI.

3.4.11 0.2.1 (2015-04-22)

- Fixed some setup.py errors, no code changes.

3.4.12 0.2.0 (2015-04-22)

- Moved configuration items to JSON files.
- Started adding support for DESI.
- Add tests to util subpackage.
- Add `__future__` statements.
- Clean up API documentation.
- Minor bug fixes.

3.4.13 0.1.0 (2015-03-25)

- Initial release. Used to scan all SDSS data.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

Symbols

C

E

F

G

H

I

L

M

N

P

R

S

T

V

W