`pydoop.hdfs` — HDFS API

This module allows you to connect to an HDFS installation, read and write files and get information on files, directories and global filesystem properties.

Configuration

The hdfs module is built on top of `libhdfs`, in turn a JNI wrapper around the Java fs code: therefore, for the module to work properly, the Java class path must include all relevant Hadoop jars. Pydoop tries to populate the class path automatically by calling `hadoop classpath`, so make sure the `hadoop` command is in the `PATH` on all cluster nodes. If your Hadoop configuration directory is in a non-standard location, also ensure that the `HADOOP_CONF_DIR` env var is set to the appropriate value.

Another important environment variable for this module is `LIBHDFS_OPTS`, used to set options for the JVM on top of which it runs. To control the heap size, for instance, you could set `LIBHDFS_OPTS` to `"-Xms32m -Xmx512m"`.

class `pydoop.hdfs.`**`hdfs`**(host='default', port=0, user=None, groups=None)

A handle to an HDFS instance.

Parameters:
- host (str) – hostname or IP address of the HDFS NameNode. Set to an empty string (and `port` to 0) to connect to the local file system; set to `'default'` (and `port` to 0) to connect to the default (i.e., the one defined in the Hadoop configuration files) file system.
- port (int) – the port on which the NameNode is listening
- user (str) – the Hadoop domain user name. Defaults to the current UNIX user. Note that, in MapReduce applications, since tasks are spawned by the JobTracker, the default user will be the one that started the JobTracker itself.
- groups (list) – ignored. Included for backwards compatibility.

Note: when connecting to the local file system, `user` is ignored (i.e., it will always be the current UNIX user).

`capacity`()

Return the raw capacity of the filesystem.

Return type:	int
Returns:	filesystem capacity

`chmod`(path, mode)

Change file mode bits.

Parameters:
- path (str) – the path to the file or directory
- mode (int) – the bitmask to set it to (e.g., 0777)

Raises: `IOError`

`chown`(path, user='', group='')

Change file owner and group.

Parameters:
- path (str) – the path to the file or directory
- user (str) – Hadoop username. Set to '' if only setting group
- group (str) – Hadoop group name. Set to '' if only setting user

Raises: `IOError`

`close`()

Close the HDFS handle (disconnect).

copy(from_path, to_hdfs, to_path)

Copy file from one filesystem to another.

Parameters:	• from_path (str) – the path of the source file • to_hdfs (`hdfs`) – destination filesystem • to_path (str) – the path of the destination file
Raises:	`IOError`

create_directory(path)

Create directory `path` (non-existent parents will be created as well).

Parameters:	path (str) – the path of the directory
Raises:	`IOError`

default_block_size()

Get the default block size.

Return type:	int
Returns:	the default blocksize

delete(path, recursive=True)

Delete `path`.

Parameters:	• path (str) – the path of the file or directory • recursive (bool) – if `path` is a directory, delete it recursively when `True`
Raises:	`IOError` when `recursive` is `False` and directory is non-empty

exists(path)

Check if a given path exists on the filesystem.

Parameters:	path (str) – the path to look for
Return type:	bool
Returns:	`True` if `path` exists

get_hosts(path, start, length)

Get hostnames where a particular block (determined by pos and blocksize) of a file is stored. Due to replication, a single block could be present on multiple hosts.

Parameters:	• path (str) – the path of the file • start (int) – the start of the block • length (int) – the length of the block
Return type:	list
Returns:	list of hosts that store the block

get_path_info(path)

Get information about `path` as a dict of properties.

The return value, based upon `fs.FileStatus` from the Java API, has the following fields:

- `block_size`: HDFS block size of `path`
- `group`: group associated with `path`
- `kind`: `'file'` or `'directory'`
- `last_access`: last access time of `path`
- `last_mod`: last modification time of `path`
- `name`: fully qualified path name
- `owner`: owner of `path`
- `permissions`: file system permissions associated with `path`

- `replication`: replication factor of `path`
- `size`: size in bytes of `path`

Parameters:	path (str) – a path in the filesystem
Return type:	dict
Returns:	path information
Raises:	`IOError`

`host`

The actual hdfs hostname (empty string for the local fs).

`list_directory`(path)

Get list of files and directories for `path`.

Parameters:	path (str) – the path of the directory
Return type:	list
Returns:	list of files and directories in `path`
Raises:	`IOError`

`move`(from_path, to_hdfs, to_path)

Move file from one filesystem to another.

Parameters:	• from_path (str) – the path of the source file • to_hdfs – destination filesystem • to_path (str) – the path of the destination file
Raises:	`IOError`

`open_file`(path, mode='r', buff_size=0, replication=0, blocksize=0, encoding=None, errors=None)

Open an HDFS file.

Supported opening modes are “r”, “w”, “a”. In addition, a trailing “t” can be added to specify text mode (e.g., “rt” = open for reading text).

Pass 0 as `buff_size`, `replication` or `blocksize` if you want to use the “configured” values, i.e., the ones set in the Hadoop configuration files.

Parameters:	• path (str) – the full path to the file • mode (str) – opening mode • buff_size (int) – read/write buffer size in bytes • replication (int) – HDFS block replication • blocksize (int) – HDFS block size
Rtpye:	`hdfs_file`
Returns:	handle to the open file

`port`

The actual hdfs port (0 for the local fs).

`rename`(from_path, to_path)

Rename file.

Parameters:	• from_path (str) – the path of the source file • to_path (str) – the path of the destination file
Raises:	`IOError`

`set_replication`(path, replication)

Set the replication of `path` to `replication`.

Parameters:	• path (str) – the path of the file • replication (int) – the replication value
Raises:	`IOError`

`set_working_directory`(path)

Set the working directory to `path`. All relative paths will be resolved relative to it.

Parameters:	path (str) – the path of the directory
Raises:	`IOError`

`used`()

Return the total raw size of all files in the filesystem.

Return type:	int
Returns:	total size of files in the file system

`user`

The user associated with this HDFS connection.

`utime`(path, mtime, atime)

Change file last access and modification times.

Parameters:	• path (str) – the path to the file or directory • mtime (int) – new modification time in seconds • atime (int) – new access time in seconds
Raises:	`IOError`

`walk`(top)

Generate infos for all paths in the tree rooted at `top` (included).

The `top` parameter can be either an HDFS path string or a dictionary of properties as returned by `get_path_info()`.

Parameters:	top (str, dict) – an HDFS path or path info dict
Return type:	iterator
Returns:	path infos of files and directories in the tree rooted at `top`
Raises:	`IOError`; `ValueError` if `top` is empty

`working_directory`()

Get the current working directory.

Return type:	str
Returns:	current working directory

`pydoop.hdfs.`**`default_is_local`**()

Is Hadoop configured to use the local file system?

By default, it is. A DFS must be explicitly configured.

`pydoop.hdfs.`**`open`**(hdfs_path, mode='r', buff_size=0, replication=0, blocksize=0, user=None, encoding=None, errors=None)

Open a file, returning an **`hdfs_file`** object.

`hdfs_path` and `user` are passed to `split()`, while the other args are passed to the **`hdfs_file`** constructor.

`pydoop.hdfs.`**`dump`**(data, hdfs_path, **kwargs)

Write `data` to `hdfs_path`.

Keyword arguments are passed to `open()`, except for `mode`, which is forced to `"w"` (or `"wt"` for text data).

`pydoop.hdfs.`**`load`**(hdfs_path, **kwargs)

Read the content of `hdfs_path` and return it.

Keyword arguments are passed to `open()`. The *"mode"* kwarg must be readonly.

`pydoop.hdfs.`**`cp`**(src_hdfs_path, dest_hdfs_path, **kwargs)

Copy the contents of `src_hdfs_path` to `dest_hdfs_path`.

If `src_hdfs_path` is a directory, its contents will be copied recursively. Source file(s) are opened for reading and copies are opened for writing. Additional keyword arguments, if any, are handled like in `open()`.

`pydoop.hdfs.`**`put`**(src_path, dest_hdfs_path, **kwargs)

Copy the contents of `src_path` to `dest_hdfs_path`.

`src_path` is forced to be interpreted as an ordinary local path (see `abspath()`). The source file is opened for reading and the copy is opened for writing. Additional keyword arguments, if any, are handled like in `open()`.

`pydoop.hdfs.`**`get`**(src_hdfs_path, dest_path, **kwargs)

Copy the contents of `src_hdfs_path` to `dest_path`.

`dest_path` is forced to be interpreted as an ordinary local path (see `abspath()`). The source file is opened for reading and the copy is opened for writing. Additional keyword arguments, if any, are handled like in `open()`.

`pydoop.hdfs.`**`mkdir`**(hdfs_path, user=None)

Create a directory and its parents as needed.

`pydoop.hdfs.`**`rm`**(hdfs_path, recursive=True, user=None)

Remove a file or directory.

If `recursive` is `True` (the default), directory contents are removed recursively.

`pydoop.hdfs.`**`lsl`**(hdfs_path, user=None, recursive=False)

Return a list of dictionaries of file properties.

If `hdfs_path` is a file, there is only one item corresponding to the file itself; if it is a directory and `recursive` is `False`, each list item corresponds to a file or directory contained by it; if it is a directory and `recursive` is `True`, the list contains one item for every file or directory in the tree rooted at `hdfs_path`.

`pydoop.hdfs.`**`ls`**(hdfs_path, user=None, recursive=False)

Return a list of hdfs paths.

Works in the same way as `lsl()`, except for the fact that list items are hdfs paths instead of dictionaries of properties.

`pydoop.hdfs.`**`chmod`**(hdfs_path, mode, user=None)

Change file mode bits.

Parameters:
- path (string) – the path to the file or directory
- mode (int) – the bitmask to set it to (e.g., 0777)

`pydoop.hdfs.`**`move`**(src, dest, user=None)

Move or rename src to dest.

`pydoop.hdfs.`**`chown`**(hdfs_path, user=None, group=None, hdfs_user=None)

See `fs.hdfs.chown()`.

`pydoop.hdfs.`**`rename`**(from_path, to_path, user=None)

See `fs.hdfs.rename()`.

`pydoop.hdfs.`**`renames`**(from_path, to_path, user=None)

Rename `from_path` to `to_path`, creating parents as needed.

`pydoop.hdfs.`**`stat`**(path, user=None)

Performs the equivalent of `os.stat()` on `path`, returning a **`StatResult`** object.

`pydoop.hdfs.`**`access`**(path, mode, user=None)

Perform the equivalent of `os.access()` on `path`.

pydoop.hdfs.path – Path Name Manipulations

class `pydoop.hdfs.path.`**`StatResult`**(path_info)

Mimics the object type returned by `os.stat()`.

Objects of this class are instantiated from dictionaries with the same structure as the ones returned by `get_path_info()`.

Attributes starting with `st_` have the same meaning as the corresponding ones in the object returned by `os.stat()`, although some of them may not make sense for an HDFS path (in this case, their value will be set to 0). In addition, the `kind`, `name` and `replication` attributes are available, with the same values as in the input dict.

`pydoop.hdfs.path.`**`abspath`**(hdfs_path, user=None, local=False)

Return an absolute path for `hdfs_path`.

The `user` arg is passed to `split()`. The `local` argument forces `hdfs_path` to be interpreted as an ordinary local path:

```
>>> import os
>>> os.chdir('/tmp')
>>> import pydoop.hdfs.path as hpath
>>> hpath.abspath('file:/tmp')
'file:/tmp'
>>> hpath.abspath('file:/tmp', local=True)
'file:/tmp/file:/tmp'
```

Note that this function always return a full URI:

```
>>> import pydoop.hdfs.path as hpath
>>> hpath.abspath('/tmp')
'hdfs://localhost:9000/tmp'
```

`pydoop.hdfs.path.`**`access`**(path, mode, user=None)

Perform the equivalent of `os.access()` on `path`.

`pydoop.hdfs.path.`**`basename`**(hdfs_path)

Return the final component of `hdfs_path`.

`pydoop.hdfs.path.`**`dirname`**(hdfs_path)

Return the directory component of `hdfs_path`.

`pydoop.hdfs.path.`**`exists`**(hdfs_path, user=None)

Return `True` if `hdfs_path` exists in the default HDFS.

`pydoop.hdfs.path.`**`expanduser`**(path)

Replace initial `~` or `~user` with the user’s home directory.

NOTE: if the default file system is HDFS, the `~user` form is expanded regardless of the user's existence.

`pydoop.hdfs.path.`**`expandvars`**(path)

Expand environment variables in `path`.

`pydoop.hdfs.path.`**`getatime`**(path, user=None)

Get time of last access of `path`.

`pydoop.hdfs.path.`**`getctime`**(path, user=None)

Get time of creation / last metadata change of `path`.

`pydoop.hdfs.path.`**`getmtime`**(path, user=None)

Get time of last modification of `path`.

`pydoop.hdfs.path.`**`getsize`**(path, user=None)

Get size, in bytes, of `path`.

`pydoop.hdfs.path.`**`isabs`**(path)

Return `True` if `path` is absolute.

A path is absolute if it is a full URI (see `isfull()`) or starts with a forward slash. No check is made to determine whether `path` is a valid HDFS path.

`pydoop.hdfs.path.`**`isdir`**(path, user=None)

Return `True` if `path` refers to a directory.

`pydoop.hdfs.path.`**`isfile`**(path, user=None)

Return `True` if `path` refers to a file.

`pydoop.hdfs.path.`**`isfull`**(path)

Return `True` if `path` is a full URI (starts with a scheme followed by a colon).

No check is made to determine whether `path` is a valid HDFS path.

`pydoop.hdfs.path.`**`islink`**(path, user=None)

Return `True` if `path` is a symbolic link.

Currently this function always returns `False` for non-local paths.

`pydoop.hdfs.path.`**`ismount`**(path)

Return `True` if `path` is a mount point.

This function always returns `False` for non-local paths.

`pydoop.hdfs.path.`**`join`**(*parts)

Join path name components, inserting `/` as needed.

If any component is an absolute path (see `isabs()`), all previous components will be discarded. However, full URIs (see `isfull()`) take precedence over incomplete ones:

```
>>> import pydoop.hdfs.path as hpath
>>> hpath.join('bar', '/foo')
'/foo'
>>> hpath.join('hdfs://host:1/', '/foo')
'hdfs://host:1/foo'
```

Note that this is not the reverse of `split()`, but rather a specialized version of `os.path.join()`. No check is made to determine whether the returned string is a valid HDFS path.

`pydoop.hdfs.path.`**`kind`**(path, user=None)

Get the kind of item ("file" or "directory") that the path references.

Return `None` if `path` doesn't exist.

`pydoop.hdfs.path.`**`normpath`**(path)

Normalize `path`, collapsing redundant separators and up-level refs.

`pydoop.hdfs.path.`**`parse`**(hdfs_path)

Parse the given path and return its components.

Parameters:	hdfs_path (str) – an HDFS path, e.g., `hdfs://localhost:9000/user/me`
Return type:	tuple
Returns:	scheme, netloc, path

`pydoop.hdfs.path.`**`realpath`**(path)

Return `path` with symlinks resolved.

Currently this function returns non-local paths unchanged.

`pydoop.hdfs.path.`**`samefile`**(path1, path2, user=None)

Return `True` if both path arguments refer to the same path.

`pydoop.hdfs.path.`**`split`**(hdfs_path, user=None)

Split `hdfs_path` into a (hostname, port, path) tuple.

Parameters:

- hdfs_path (str) – an HDFS path, e.g., `hdfs://localhost:9000/user/me`
- user (str) – user name used to resolve relative paths, defaults to the current user

Return type: tuple

Returns: hostname, port, path

`pydoop.hdfs.path.`**`splitext`**(path)

Same as `os.path.splitext()`.

`pydoop.hdfs.path.`**`splitpath`**(hdfs_path)

Split `hdfs_path` into a (`head`, `tail`) pair, according to the same rules as `os.path.split()`.

`pydoop.hdfs.path.`**`stat`**(path, user=None)

Performs the equivalent of `os.stat()` on `path`, returning a `StatResult` object.

`pydoop.hdfs.path.`**`unparse`**(scheme, netloc, path)

Construct a path from its three components (see `parse()`).

pydoop.hdfs.fs – File System Handles

`pydoop.hdfs.fs.`**`default_is_local`**()

Is Hadoop configured to use the local file system?

By default, it is. A DFS must be explicitly configured.

class `pydoop.hdfs.fs.`**`hdfs`**(host='default', port=0, user=None, groups=None)

A handle to an HDFS instance.

Parameters:

- host (str) – hostname or IP address of the HDFS NameNode. Set to an empty string (and `port` to 0) to connect to the local file system; set to `'default'` (and `port` to 0) to connect to the default (i.e., the one defined in the Hadoop configuration files) file system.
- port (int) – the port on which the NameNode is listening

- user (str) – the Hadoop domain user name. Defaults to the current UNIX user. Note that, in MapReduce applications, since tasks are spawned by the JobTracker, the default user will be the one that started the JobTracker itself.
- groups (list) – ignored. Included for backwards compatibility.

Note: when connecting to the local file system, `user` is ignored (i.e., it will always be the current UNIX user).

`capacity`()

Return the raw capacity of the filesystem.

Return type:	int
Returns:	filesystem capacity

`chmod`(path, mode)

Change file mode bits.

Parameters:
- path (str) – the path to the file or directory
- mode (int) – the bitmask to set it to (e.g., 0777)

Raises: `IOError`

`chown`(path, user='', group='')

Change file owner and group.

Parameters:
- path (str) – the path to the file or directory
- user (str) – Hadoop username. Set to '' if only setting group
- group (str) – Hadoop group name. Set to '' if only setting user

Raises: `IOError`

`close`()

Close the HDFS handle (disconnect).

`copy`(from_path, to_hdfs, to_path)

Copy file from one filesystem to another.

Parameters:
- from_path (str) – the path of the source file
- to_hdfs (`hdfs`) – destination filesystem
- to_path (str) – the path of the destination file

Raises: `IOError`

`create_directory`(path)

Create directory `path` (non-existent parents will be created as well).

Parameters:	path (str) – the path of the directory
Raises:	`IOError`

`default_block_size`()

Get the default block size.

Return type:	int
Returns:	the default blocksize

`delete`(path, recursive=True)

Delete `path`.

Parameters:
- path (str) – the path of the file or directory

- recursive (bool) – if `path` is a directory, delete it recursively when **`True`**

Raises: **`IOError`** when `recursive` is **`False`** and directory is non-empty

`exists`(path)

Check if a given path exists on the filesystem.

Parameters: path (str) – the path to look for

Return type: bool

Returns: **`True`** if `path` exists

`get_hosts`(path, start, length)

Get hostnames where a particular block (determined by pos and blocksize) of a file is stored. Due to replication, a single block could be present on multiple hosts.

Parameters:
- path (str) – the path of the file
- start (int) – the start of the block
- length (int) – the length of the block

Return type: list

Returns: list of hosts that store the block

`get_path_info`(path)

Get information about `path` as a dict of properties.

The return value, based upon `fs.FileStatus` from the Java API, has the following fields:

- `block_size`: HDFS block size of `path`
- `group`: group associated with `path`
- `kind`: `'file'` or `'directory'`
- `last_access`: last access time of `path`
- `last_mod`: last modification time of `path`
- `name`: fully qualified path name
- `owner`: owner of `path`
- `permissions`: file system permissions associated with `path`
- `replication`: replication factor of `path`
- `size`: size in bytes of `path`

Parameters: path (str) – a path in the filesystem

Return type: dict

Returns: path information

Raises: **`IOError`**

`host`

The actual hdfs hostname (empty string for the local fs).

`list_directory`(path)

Get list of files and directories for `path`.

Parameters: path (str) – the path of the directory

Return type: list

Returns: list of files and directories in `path`

Raises: **`IOError`**

`move`(from_path, to_hdfs, to_path)

Move file from one filesystem to another.

Parameters:
- from_path (str) – the path of the source file

- to_hdfs – destination filesystem
- to_path (str) – the path of the destination file

Raises: `IOError`

`open_file`(path, mode='r', buff_size=0, replication=0, blocksize=0, encoding=None, errors=None)

Open an HDFS file.

Supported opening modes are "r", "w", "a". In addition, a trailing "t" can be added to specify text mode (e.g., "rt" = open for reading text).

Pass 0 as `buff_size`, `replication` or `blocksize` if you want to use the "configured" values, i.e., the ones set in the Hadoop configuration files.

Parameters:
- path (str) – the full path to the file
- mode (str) – opening mode
- buff_size (int) – read/write buffer size in bytes
- replication (int) – HDFS block replication
- blocksize (int) – HDFS block size

Rtpye: `hdfs_file`

Returns: handle to the open file

`port`

The actual hdfs port (0 for the local fs).

`rename`(from_path, to_path)

Rename file.

Parameters:
- from_path (str) – the path of the source file
- to_path (str) – the path of the destination file

Raises: `IOError`

`set_replication`(path, replication)

Set the replication of `path` to `replication`.

Parameters:
- path (str) – the path of the file
- replication (int) – the replication value

Raises: `IOError`

`set_working_directory`(path)

Set the working directory to `path`. All relative paths will be resolved relative to it.

Parameters: path (str) – the path of the directory

Raises: `IOError`

`used`()

Return the total raw size of all files in the filesystem.

Return type: int

Returns: total size of files in the file system

`user`

The user associated with this HDFS connection.

`utime`(path, mtime, atime)

Change file last access and modification times.

Parameters:
- path (str) – the path to the file or directory

	• mtime (int) – new modification time in seconds • atime (int) – new access time in seconds
Raises:	`IOError`

`walk`(top)

Generate infos for all paths in the tree rooted at `top` (included).

The `top` parameter can be either an HDFS path string or a dictionary of properties as returned by `get_path_info()`.

Parameters:	top (str, dict) – an HDFS path or path info dict
Return type:	iterator
Returns:	path infos of files and directories in the tree rooted at `top`
Raises:	`IOError`; `ValueError` if `top` is empty

`working_directory`()

Get the current working directory.

Return type:	str
Returns:	current working directory

pydoop.hdfs.file – HDFS File Objects

class `pydoop.hdfs.file.`**`FileIO`**(raw_hdfs_file, fs, mode, encoding=None, errors=None)

Instances of this class represent HDFS file objects.

Objects from this class should not be instantiated directly. To open an HDFS file, use `open_file()`, or the top-level `open` function in the hdfs package.

`available`()

Number of bytes that can be read from this input stream without blocking.

Return type:	int
Returns:	available bytes

`close`()

Close the file.

`flush`()

Force any buffered output to be written.

`fs`

The file’s hdfs instance.

`name`

The file’s fully qualified name.

`next`()

Return the next input line, or raise **`StopIteration`** when EOF is hit.

`pread`(position, length)

Read `length` bytes of data from the file, starting from `position`.

Parameters:	• position (int) – position from which to read • length (int) – the number of bytes to read
Return type:	string
Returns:	the chunk of data read from the file

read(length=-1)

Read `length` bytes from the file. If `length` is negative or omitted, read all data until EOF.

Parameters:	length (int) – the number of bytes to read
Return type:	string
Returns:	the chunk of data read from the file

readline()

Read and return a line of text.

Return type:	str
Returns:	the next line of text in the file, including the newline character

seek(position, whence=0)

Seek to `position` in file.

Parameters:

- position (int) – offset in bytes to seek to
- whence (int) – defaults to `os.SEEK_SET` (absolute); other values are `os.SEEK_CUR` (relative to the current position) and `os.SEEK_END` (relative to the file's end).

size

The file's size in bytes. This attribute is initialized when the file is opened and updated when it is closed.

tell()

Get the current byte offset in the file.

Return type:	int
Returns:	current offset in bytes

write(data)

Write `data` to the file.

Parameters:	data (bytes) – the data to be written to the file
Return type:	int
Returns:	the number of bytes written

class `pydoop.hdfs.file.`**local_file**(fs, name, mode)

Support class to handle local files.

Object of this type have the same interface as **`FileIO`** (and should also be obtained via higher level methods rather than instantiated directly), but act as handles to local files.