Overview

The File System (FS) shell includes various shell-like commands that directly interact with the Hadoop Distributed File System (HDFS) as well as other file systems that Hadoop supports, such as Local FS, HFTP FS, S3 FS, and others. The FS shell is invoked by:

```
bin/hadoop fs <args>
```

All FS shell commands take path URIs as arguments. The URI format is `scheme://authority/path`. For HDFS the scheme is `hdfs`, and for the Local FS the scheme is `file`. The scheme and authority are optional. If not specified, the default scheme specified in the configuration is used. An HDFS file or directory such as /parent/child can be specified as `hdfs://namenodehost/parent/child` or simply as `/parent/child` (given that your configuration is set to point to `hdfs://namenodehost`).

Most of the commands in FS shell behave like corresponding Unix commands. Differences are described with each of the commands. Error information is sent to stderr and the output is sent to stdout.

If HDFS is being used, `hdfs dfs` is a synonym.

Relative paths can be used. For HDFS, the current working directory is the HDFS home directory `/user/<username>` that often has to be created manually. The HDFS home directory can also be implicitly accessed, e.g., when using the HDFS trash folder, the `.Trash` directory in the home directory.

See the Commands Manual for generic shell options.

appendToFile

Usage: `hadoop fs -appendToFile <localsrc> ... <dst>`

Append single src, or multiple srcs from local file system to the destination file system. Also reads input from stdin and appends to destination file system.

- `hadoop fs -appendToFile localfile /user/hadoop/hadoopfile`
- `hadoop fs -appendToFile localfile1 localfile2 /user/hadoop/hadoopfile`
- `hadoop fs -appendToFile localfile hdfs://nn.example.com/hadoop/hadoopfile`
- `hadoop fs -appendToFile - hdfs://nn.example.com/hadoop/hadoopfile` Reads the input from stdin.

Exit Code:

Returns 0 on success and 1 on error.

cat

Usage: `hadoop fs -cat [-ignoreCrc] URI [URI ...]`

Copies source paths to stdout.

Options

- The `-ignoreCrc` option disables checkshum verification.

Example:

- `hadoop fs -cat hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2`
- `hadoop fs -cat file:///file3 /user/hadoop/file4`

Exit Code:

Returns 0 on success and -1 on error.

checksum

Usage: `hadoop fs -checksum URI`

Returns the checksum information of a file.

Example:

- `hadoop fs -checksum hdfs://nn1.example.com/file1`
- `hadoop fs -checksum file:///etc/hosts`

chgrp

Usage: `hadoop fs -chgrp [-R] GROUP URI [URI ...]`

Change group association of files. The user must be the owner of files, or else a super-user. Additional information is in the Permissions Guide.

Options

- The -R option will make the change recursively through the directory structure.

chmod

Usage: `hadoop fs -chmod [-R] <MODE[,MODE]... | OCTALMODE> URI [URI ...]`

Change the permissions of files. With -R, make the change recursively through the directory structure. The user must be the owner of the file, or else a super-user. Additional information is in the Permissions Guide.

Options

- The -R option will make the change recursively through the directory structure.

chown

Usage: `hadoop fs -chown [-R] [OWNER][:[GROUP]] URI [URI ]`

Change the owner of files. The user must be a super-user. Additional information is in the Permissions Guide.

Options

- The -R option will make the change recursively through the directory structure.

copyFromLocal

Usage: `hadoop fs -copyFromLocal <localsrc> URI`

Similar to the `fs -put` command, except that the source is restricted to a local file reference.

Options:

- `-p` : Preserves access and modification times, ownership and the permissions. (assuming the permissions can be propagated across filesystems)
- `-f` : Overwrites the destination if it already exists.
- `-l` : Allow DataNode to lazily persist the file to disk, Forces a replication factor of 1. This flag will result in reduced durability. Use with care.
- `-d` : Skip creation of temporary file with the suffix `._COPYING_`.

copyToLocal

Usage: `hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>`

Similar to get command, except that the destination is restricted to a local file reference.

count

Usage: `hadoop fs -count [-q] [-h] [-v] [-x] [-t [<storage type>]] [-u] <paths>`

Count the number of directories, files and bytes under the paths that match the specified file pattern. Get the quota and the usage. The output columns with -count are: DIR_COUNT, FILE_COUNT, CONTENT_SIZE, PATHNAME

The -u and -q options control what columns the output contains. -q means show quotas, -u limits the output to show quotas and usage only.

The output columns with -count -q are: QUOTA, REMAINING_QUOTA, SPACE_QUOTA, REMAINING_SPACE_QUOTA, DIR_COUNT, FILE_COUNT, CONTENT_SIZE, PATHNAME

The output columns with -count -u are: QUOTA, REMAINING_QUOTA, SPACE_QUOTA, REMAINING_SPACE_QUOTA

The -t option shows the quota and usage for each storage type. The -t option is ignored if -u or -q option is not given. The list of possible parameters that can be used in -t option(case insensitive except the parameter "‶): ″‶, ″all‶, ″ram_disk‶, ″ssd‶, ″disk‶ or ″archive".

The -h option shows sizes in human readable format.

The -v option displays a header line.

The -x option excludes snapshots from the result calculation. Without the -x option (default), the result is always calculated from all INodes, including all snapshots under the given path. The -x option is ignored if -u or -q option is given.

Example:

- `hadoop fs -count hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2`
- `hadoop fs -count -q hdfs://nn1.example.com/file1`
- `hadoop fs -count -q -h hdfs://nn1.example.com/file1`
- `hadoop fs -count -q -h -v hdfs://nn1.example.com/file1`
- `hadoop fs -count -u hdfs://nn1.example.com/file1`
- `hadoop fs -count -u -h hdfs://nn1.example.com/file1`
- `hadoop fs -count -u -h -v hdfs://nn1.example.com/file1`

Exit Code:

Returns 0 on success and -1 on error.

cp

Usage: `hadoop fs -cp [-f] [-p | -p[topax]] URI [URI ...] <dest>`

Copy files from source to destination. This command allows multiple sources as well in which case the destination must be a directory.

‘raw.*’ namespace extended attributes are preserved if (1) the source and destination filesystems support them (HDFS only), and (2) all source and destination pathnames are in the /.reserved/raw hierarchy. Determination of whether raw.* namespace xattrs are preserved is independent of the -p (preserve) flag.

Options:

- The -f option will overwrite the destination if it already exists.
- The -p option will preserve file attributes [topx] (timestamps, ownership, permission, ACL, XAttr). If -p is specified with no *arg*, then preserves timestamps, ownership, permission. If -pa is specified, then preserves permission also because ACL is a super-set of permission. Determination of whether raw namespace extended attributes are preserved is independent of the -p flag.

Example:

- `hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2`
- `hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir`

Exit Code:

Returns 0 on success and -1 on error.

createSnapshot

See HDFS Snapshots Guide.

deleteSnapshot

See HDFS Snapshots Guide.

df

Usage: `hadoop fs -df [-h] URI [URI ...]`

Displays free space.

Options:

- The -h option will format file sizes in a "human-readable" fashion (e.g 64.0m instead of 67108864)

Example:

- `hadoop dfs -df /user/hadoop/dir1`

du

Usage: `hadoop fs -du [-s] [-h] [-x] URI [URI ...]`

Displays sizes of files and directories contained in the given directory or the length of a file in case its just a file.

Options:

- The -s option will result in an aggregate summary of file lengths being displayed, rather than the individual files. Without the -s option, calculation is done by going 1-level deep from the given path.
- The -h option will format file sizes in a "human-readable" fashion (e.g 64.0m instead of 67108864)
- The -x option will exclude snapshots from the result calculation. Without the -x option (default), the result is always calculated from all INodes, including all snapshots under the given path.

The du returns three columns with the following format:

```
size disk_space_consumed_with_all_replicas full_path_name
```

Example:

- `hadoop fs -du /user/hadoop/dir1 /user/hadoop/file1 hdfs://nn.example.com/user/hadoop/dir1`

Exit Code: Returns 0 on success and -1 on error.

dus

Usage: `hadoop fs -dus <args>`

Displays a summary of file lengths.

Note: This command is deprecated. Instead use `hadoop fs -du -s`.

expunge

Usage: `hadoop fs -expunge`

Permanently delete files in checkpoints older than the retention threshold from trash directory, and create new checkpoint.

When checkpoint is created, recently deleted files in trash are moved under the checkpoint. Files in checkpoints older than `fs.trash.interval` will be permanently deleted on the next invocation of `-expunge` command.

If the file system supports the feature, users can configure to create and delete checkpoints periodically by the parameter stored as `fs.trash.checkpoint.interval` (in core-site.xml). This value should be smaller or equal to `fs.trash.interval`.

Refer to the HDFS Architecture guide for more information about trash feature of HDFS.

find

Usage: `hadoop fs -find <path> ... <expression> ...`

Finds all files that match the specified expression and applies selected actions to them. If no *path* is specified then defaults to the current working directory. If no expression is specified then defaults to -print.

The following primary expressions are recognised:

- -name pattern
  -iname pattern

  Evaluates as true if the basename of the file matches the pattern using standard file system globbing. If -iname is used then the match is case insensitive.

- -print
  -print0

  Always evaluates to true. Causes the current pathname to be written to standard output. If the -print0 expression is used then an ASCII NULL character is appended.

The following operators are recognised:

- expression -a expression
  expression -and expression
  expression expression

  Logical AND operator for joining two expressions. Returns true if both child expressions return true. Implied by the juxtaposition of two expressions and so does not need to be explicitly specified. The second expression will not be applied if the first fails.

Example:

```
hadoop fs -find / -name test -print
```

Exit Code:

Returns 0 on success and -1 on error.

get

Usage: `hadoop fs -get [-ignorecrc] [-crc] [-p] [-f] <src> <localdst>`

Copy files to the local file system. Files that fail the CRC check may be copied with the -ignorecrc option. Files and CRCs may be copied using the -crc option.

Example:

- `hadoop fs -get /user/hadoop/file localfile`
- `hadoop fs -get hdfs://nn.example.com/user/hadoop/file localfile`

Exit Code:

Returns 0 on success and -1 on error.

Options:

- `-p` : Preserves access and modification times, ownership and the permissions. (assuming the permissions can be propagated across filesystems)
- `-f` : Overwrites the destination if it already exists.
- `-ignorecrc` : Skip CRC checks on the file(s) downloaded.
- `-crc`: write CRC checksums for the files downloaded.

getfacl

Usage: `hadoop fs -getfacl [-R] <path>`

Displays the Access Control Lists (ACLs) of files and directories. If a directory has a default ACL, then getfacl also displays the default ACL.

Options:

- -R: List the ACLs of all files and directories recursively.
- *path*: File or directory to list.

Examples:

- `hadoop fs -getfacl /file`
- `hadoop fs -getfacl -R /dir`

Exit Code:

Returns 0 on success and non-zero on error.

getfattr

Usage: `hadoop fs -getfattr [-R] -n name | -d [-e en] <path>`

Displays the extended attribute names and values (if any) for a file or directory.

Options:

- -R: Recursively list the attributes for all files and directories.
- -n name: Dump the named extended attribute value.
- -d: Dump all extended attribute values associated with pathname.
- -e *encoding*: Encode values after retrieving them. Valid encodings are “text”, “hex”, and “base64”. Values encoded as text strings are enclosed in double quotes ("), and values encoded as hexadecimal and base64 are prefixed with 0x and 0s, respectively.
- *path*: The file or directory.

Examples:

- `hadoop fs -getfattr -d /file`
- `hadoop fs -getfattr -R -n user.myAttr /dir`

Exit Code:

Returns 0 on success and non-zero on error.

getmerge

Usage: `hadoop fs -getmerge [-nl] <src> <localdst>`

Takes a source directory and a destination file as input and concatenates files in src into the destination local file. Optionally -nl can be set to enable adding a newline character (LF) at the end of each file. -skip-empty-file can be used to avoid unwanted newline characters in case of empty files.

Examples:

- `hadoop fs -getmerge -nl /src /opt/output.txt`
- `hadoop fs -getmerge -nl /src/file1.txt /src/file2.txt /output.txt`

Exit Code:

Returns 0 on success and non-zero on error.

help

Usage: `hadoop fs -help`

Return usage output.

ls

Usage: `hadoop fs -ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] <args>`

Options:

- -C: Display the paths of files and directories only.
- -d: Directories are listed as plain files.
- -h: Format file sizes in a human-readable fashion (eg 64.0m instead of 67108864).
- -q: Print ? instead of non-printable characters.
- -R: Recursively list subdirectories encountered.
- -t: Sort output by modification time (most recent first).
- -S: Sort output by file size.
- -r: Reverse the sort order.
- -u: Use access time rather than modification time for display and sorting.

For a file ls returns stat on the file with the following format:

```
permissions number_of_replicas userid groupid filesize modification_date modification_
```

For a directory it returns list of its direct children as in Unix. A directory is listed as:

```
permissions userid groupid modification_date modification_time dirname
```

Files within a directory are order by filename by default.

Example:

- `hadoop fs -ls /user/hadoop/file1`

Exit Code:

Returns 0 on success and -1 on error.

lsr

Usage: `hadoop fs -lsr <args>`

Recursive version of ls.

Note: This command is deprecated. Instead use `hadoop fs -ls -R`

mkdir

Usage: `hadoop fs -mkdir [-p] <paths>`

Takes path uri's as argument and creates directories.

Options:

- The -p option behavior is much like Unix mkdir -p, creating parent directories along the path.

Example:

- `hadoop fs -mkdir /user/hadoop/dir1 /user/hadoop/dir2`
- `hadoop fs -mkdir hdfs://nn1.example.com/user/hadoop/dir hdfs://nn2.example.com/user/hadoop/dir`

Exit Code:

Returns 0 on success and -1 on error.

moveFromLocal

Usage: `hadoop fs -moveFromLocal <localsrc> <dst>`

Similar to put command, except that the source localsrc is deleted after it's copied.

moveToLocal

Usage: `hadoop fs -moveToLocal [-crc] <src> <dst>`

Displays a "Not implemented yet" message.

mv

Usage: `hadoop fs -mv URI [URI ...] <dest>`

Moves files from source to destination. This command allows multiple sources as well in which case the destination needs to be a directory. Moving files across file systems is not permitted.

Example:

- `hadoop fs -mv /user/hadoop/file1 /user/hadoop/file2`
- `hadoop fs -mv hdfs://nn.example.com/file1 hdfs://nn.example.com/file2 hdfs://nn.example.com/file3 hdfs://nn.example.com/dir1`

Exit Code:

Returns 0 on success and -1 on error.

put

Usage: `hadoop fs -put [-f] [-p] [-l] [-d] [ - | <localsrc1> .. ]. <dst>`

Copy single src, or multiple srcs from local file system to the destination file system. Also reads input from stdin and writes to destination file system if the source is set to "-"

Copying fails if the file already exists, unless the -f flag is given.

Options:

- `-p` : Preserves access and modification times, ownership and the permissions. (assuming the permissions can be propagated across filesystems)
- `-f` : Overwrites the destination if it already exists.
- `-l` : Allow DataNode to lazily persist the file to disk, Forces a replication factor of 1. This flag will result in reduced durability. Use with care.
- `-d` : Skip creation of temporary file with the suffix `._COPYING_`.

Examples:

- `hadoop fs -put localfile /user/hadoop/hadoopfile`
- `hadoop fs -put -f localfile1 localfile2 /user/hadoop/hadoopdir`
- `hadoop fs -put -d localfile hdfs://nn.example.com/hadoop/hadoopfile`
- `hadoop fs -put - hdfs://nn.example.com/hadoop/hadoopfile` Reads the input from stdin.

Exit Code:

Returns 0 on success and -1 on error.

renameSnapshot

See HDFS Snapshots Guide.

rm

Usage: `hadoop fs -rm [-f] [-r |-R] [-skipTrash] [-safely] URI [URI ...]`

Delete files specified as args.

If trash is enabled, file system instead moves the deleted file to a trash directory (given by FileSystem#getTrashRoot).

Currently, the trash feature is disabled by default. User can enable trash by setting a value greater than zero for parameter `fs.trash.interval` (in core-site.xml).

See expunge about deletion of files in trash.

Options:

- The -f option will not display a diagnostic message or modify the exit status to reflect an error if the file does not exist.
- The -R option deletes the directory and any content under it recursively.
- The -r option is equivalent to -R.
- The -skipTrash option will bypass trash, if enabled, and delete the specified file(s) immediately. This can be useful when it is necessary to delete files from an over-quota directory.
- The -safely option will require safety confirmation before deleting directory with total number of files greater than `hadoop.shell.delete.limit.num.files` (in core-site.xml, default: 100). It can be used with -skipTrash to prevent accidental deletion of large directories. Delay is expected when walking over large directory recursively to count the number of files to be deleted before the confirmation.

Example:

- `hadoop fs -rm hdfs://nn.example.com/file /user/hadoop/emptydir`

Exit Code:

Returns 0 on success and -1 on error.

rmdir

Usage: `hadoop fs -rmdir [--ignore-fail-on-non-empty] URI [URI ...]`

Delete a directory.

Options:

- `--ignore-fail-on-non-empty`: When using wildcards, do not fail if a directory still contains files.

Example:

- `hadoop fs -rmdir /user/hadoop/emptydir`

rmr

Usage: `hadoop fs -rmr [-skipTrash] URI [URI ...]`

Recursive version of delete.

Note: This command is deprecated. Instead use `hadoop fs -rm -r`

setfacl

Usage: `hadoop fs -setfacl [-R] [-b |-k -m |-x <acl_spec> <path>] |[--set <acl_spec> <path>]`

Sets Access Control Lists (ACLs) of files and directories.

Options:

- -b: Remove all but the base ACL entries. The entries for user, group and others are retained for compatibility with permission bits.
- -k: Remove the default ACL.
- -R: Apply operations to all files and directories recursively.
- -m: Modify ACL. New entries are added to the ACL, and existing entries are retained.
- -x: Remove specified ACL entries. Other ACL entries are retained.
- `--set`: Fully replace the ACL, discarding all existing entries. The *acl_spec* must include entries for user, group, and others for compatibility with permission bits.
- *acl_spec*: Comma separated list of ACL entries.
- *path*: File or directory to modify.

Examples:

- `hadoop fs -setfacl -m user:hadoop:rw- /file`
- `hadoop fs -setfacl -x user:hadoop /file`
- `hadoop fs -setfacl -b /file`
- `hadoop fs -setfacl -k /dir`
- `hadoop fs -setfacl --set user::rw-,user:hadoop:rw-,group::r--,other::r-- /file`
- `hadoop fs -setfacl -R -m user:hadoop:r-x /dir`
- `hadoop fs -setfacl -m default:user:hadoop:r-x /dir`

Exit Code:

Returns 0 on success and non-zero on error.

setfattr

Usage: `hadoop fs -setfattr -n name [-v value] | -x name <path>`

Sets an extended attribute name and value for a file or directory.

Options:

- -b: Remove all but the base ACL entries. The entries for user, group and others are retained for compatibility with permission bits.
- -n name: The extended attribute name.
- -v value: The extended attribute value. There are three different encoding methods for the value. If the argument is enclosed in double quotes, then the value is the string inside the quotes. If the argument is prefixed with 0x or 0X, then it is taken as a hexadecimal number. If the argument begins with 0s or 0S, then it is taken as a base64 encoding.
- -x name: Remove the extended attribute.
- *path*: The file or directory.

Examples:

- `hadoop fs -setfattr -n user.myAttr -v myValue /file`
- `hadoop fs -setfattr -n user.noValue /file`
- `hadoop fs -setfattr -x user.myAttr /file`

Exit Code:

Returns 0 on success and non-zero on error.

setrep

Usage: `hadoop fs -setrep [-R] [-w] <numReplicas> <path>`

Changes the replication factor of a file. If *path* is a directory then the command recursively changes the replication factor of all files under the directory tree rooted at *path*.

Options:

- The -w flag requests that the command wait for the replication to complete. This can potentially take a very long time.
- The -R flag is accepted for backwards compatibility. It has no effect.

Example:

- `hadoop fs -setrep -w 3 /user/hadoop/dir1`

Exit Code:

Returns 0 on success and -1 on error.

stat

Usage: `hadoop fs -stat [format] <path> ...`

Print statistics about the file/directory at <path> in the specified format. Format accepts filesize in blocks (%b), type (%F), group name of owner (%g), name (%n), block size (%o), replication (%r), user name of owner(%u), and modification date (%y, %Y). %y shows UTC date as "yyyy-MM-dd HH:mm:ss" and %Y shows milliseconds since January 1, 1970 UTC. If the format is not specified, %y is used by default.

Example:

- `hadoop fs -stat "%F %u:%g %b %y %n" /file`

Exit Code: Returns 0 on success and -1 on error.

tail

Usage: `hadoop fs -tail [-f] URI`

Displays last kilobyte of the file to stdout.

Options:

- The -f option will output appended data as the file grows, as in Unix.

Example:

- `hadoop fs -tail pathname`

Exit Code: Returns 0 on success and -1 on error.

test

Usage: `hadoop fs -test -[defsz] URI`

Options:

- -d: f the path is a directory, return 0.
- -e: if the path exists, return 0.
- -f: if the path is a file, return 0.
- -s: if the path is not empty, return 0.
- -r: if the path exists and read permission is granted, return 0.
- -w: if the path exists and write permission is granted, return 0.
- -z: if the file is zero length, return 0.

Example:

- `hadoop fs -test -e filename`

text

Usage: `hadoop fs -text <src>`

Takes a source file and outputs the file in text format. The allowed formats are zip and TextRecordInputStream.

touchz

Usage: `hadoop fs -touchz URI [URI ...]`

Create a file of zero length. An error is returned if the file exists with non-zero length.

Example:

- `hadoop fs -touchz pathname`

Exit Code: Returns 0 on success and -1 on error.

truncate

Usage: `hadoop fs -truncate [-w] <length> <paths>`

Truncate all files that match the specified file pattern to the specified length.

Options:

- The `-w` flag requests that the command waits for block recovery to complete, if necessary. Without -w flag the file may remain unclosed for some time while the recovery is in progress. During this time file cannot be reopened for append.

Example:

- `hadoop fs -truncate 55 /user/hadoop/file1 /user/hadoop/file2`
- `hadoop fs -truncate -w 127 hdfs://nn1.example.com/user/hadoop/file1`

usage

Usage: `hadoop fs -usage command`

Return the help for an individual command.

Working with Object Storage

The Hadoop FileSystem shell works with Object Stores such as Amazon S3, Azure WASB and OpenStack Swift.

```
# Create a directory
hadoop fs -mkdir s3a://bucket/datasets/

# Upload a file from the cluster filesystem
hadoop fs -put /datasets/example.orc s3a://bucket/datasets/

# touch a file
hadoop fs -touchz wasb://yourcontainer@youraccount.blob.core.windows.net/touched
```

Unlike a normal filesystem, renaming files and directories in an object store usually takes time proportional to the size of the objects being manipulated. As many of the filesystem shell operations use renaming as the final stage in operations, skipping that stage can avoid long delays.

In particular, the `put` and `copyFromLocal` commands should both have the `-d` options set for a direct upload.

```
# Upload a file from the cluster filesystem
hadoop fs -put -d /datasets/example.orc s3a://bucket/datasets/

# Upload a file from under the user's home directory in the local filesystem.
# Note it is the shell expanding the "~", not the hadoop fs command
hadoop fs -copyFromLocal -d -f ~/datasets/devices.orc s3a://bucket/datasets/

# create a file from stdin
# the special "-" source means "use stdin"
echo "hello" | hadoop fs -put -d -f - wasb://yourcontainer@youraccount.blob.core.windo
```

Objects can be downloaded and viewed:

```
# copy a directory to the local filesystem
hadoop fs -copyToLocal s3a://bucket/datasets/

# copy a file from the object store to the cluster filesystem.
hadoop fs -get wasb://yourcontainer@youraccount.blob.core.windows.net/hello.txt /examp
```

```
# print the object
hadoop fs -cat wasb://yourcontainer@youraccount.blob.core.windows.net/hello.txt

# print the object, unzipping it if necessary
hadoop fs -text wasb://yourcontainer@youraccount.blob.core.windows.net/hello.txt

## download log files into a local file
hadoop fs -getmerge wasb://yourcontainer@youraccount.blob.core.windows.net/logs\* log
```

Commands which list many files tend to be significantly slower than when working with HDFS or other filesystems

```
hadoop fs -count s3a://bucket/
hadoop fs -du s3a://bucket/
```

Other slow commands include `find`, `mv`, `cp` and `rm`.

Find

This can be very slow on a large store with many directories under the path supplied.

```
# enumerate all files in the object store's container.
hadoop fs -find s3a://bucket/ -print

# remember to escape the wildcards to stop the shell trying to expand them first
hadoop fs -find s3a://bucket/datasets/ -name \*.txt -print
```

Rename

The time to rename a file depends on its size.

The time to rename a directory depends on the number and size of all files beneath that directory.

```
hadoop fs -mv s3a://bucket/datasets s3a://bucket/historical
```

If the operation is interrupted, the object store will be in an undefined state.

Copy

```
hadoop fs -cp s3a://bucket/datasets s3a://bucket/historical
```

The copy operation reads each file and then writes it back to the object store; the time to complete depends on the amount of data to copy, and the bandwidth in both directions between the local computer and the object store.

The further the computer is from the object store, the longer the copy takes

Deleting objects

The `rm` command will delete objects and directories full of objects. If the object store is *eventually consistent*, `fs ls` commands and other accessors may briefly return the details of the now-deleted objects; this is an artifact of object stores which cannot be avoided.

If the filesystem client is configured to copy files to a trash directory, this will be in the bucket; the `rm` operation will then take time proportional to the size of the data. Furthermore, the deleted files will continue to incur storage costs.

To avoid this, use the the `-skipTrash` option.

```
hadoop fs -rm -skipTrash s3a://bucket/dataset
```

Data moved to the `.Trash` directory can be purged using the `expunge` command. As this command only works with the default filesystem, it must be configured to make the default filesystem the target object store.

```
hadoop fs -expunge -D fs.defaultFS=s3a://bucket/
```

Overwriting Objects

If an object store is *eventually consistent*, then any operation which overwrites existing objects may not be immediately visible to all clients/queries. That is: later operations which query the same object's status or contents may get the previous object. This can sometimes surface within the same client, while reading a single object.

Avoid having a sequence of commands which overwrite objects and then immediately work on the updated data; there is a risk that the previous data will be used instead.

Timestamps

Timestamps of objects and directories in Object Stores may not follow the behavior of files and directories in HDFS.

1. The creation and initial modification times of an object will be the time it was created on the object store; this will be at the end of the write process, not the beginning.
2. The timestamp will be taken from the object store infrastructure's clock, not that of the client.
3. If an object is overwritten, the modification time will be updated.
4. Directories may or may not have valid timestamps. They are unlikely to have their modification times updated when an object underneath is updated.
5. The `atime` access time feature is not supported by any of the object stores found in the Apache Hadoop codebase.

Consult the `DistCp` documentation for details on how this may affect the `distcp -update` operation.

Security model and operations

The security and permissions models of object stores are usually very different from those of a Unix-style filesystem; operations which query or manipulate permissions are generally unsupported.

Operations to which this applies include: `chgrp`, `chmod`, `chown`, `getfacl`, and `setfacl`. The related attribute commands `getfattr` and`setfattr` are also usually unavailable.

- Filesystem commands which list permission and user/group details, usually simulate these details.
- Operations which try to preserve permissions (example `fs -put -p`) do not preserve permissions for this reason. (Special case: `wasb://`, which preserves permissions but does not enforce them).

When interacting with read-only object stores, the permissions found in "list" and "stat" commands may indicate that the user has write access, when in fact they do not.

Object stores usually have permissions models of their own, models can be manipulated through store-specific tooling. Be aware that some of the permissions which an object store may provide (such as write-only paths, or different permissions on the root path) may be incompatible with the Hadoop filesystem clients. These tend to require full read and write access to the entire object store bucket/container into which they write data.

As an example of how permissions are mocked, here is a listing of Amazon's public, read-only bucket of Landsat images:

```
$ hadoop fs -ls s3a://landsat-pds/
Found 10 items
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/L8
-rw-rw-rw-   1 mapred      23764 2015-01-28 18:13 s3a://landsat-pds/index.html
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/landsat-pds_stats
-rw-rw-rw-   1 mapred        105 2016-08-19 18:12 s3a://landsat-pds/robots.txt
-rw-rw-rw-   1 mapred         38 2016-09-26 12:16 s3a://landsat-pds/run_info.json
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/runs
-rw-rw-rw-   1 mapred   27458808 2016-09-26 12:16 s3a://landsat-pds/scene_list.gz
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/tarq
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/tarq_corrupt
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/test
```

1. All files are listed as having full read/write permissions.
2. All directories appear to have full `rwx` permissions.
3. The replication count of all files is "1".
4. The owner of all files and directories is declared to be the current user (`mapred`).
5. The timestamp of all directories is actually that of the time the `-ls` operation was executed. This is because these directories are not actual objects in the store; they are simulated directories based on the existence of objects under their paths.

When an attempt is made to delete one of the files, the operation fails —despite the permissions shown by the `ls` command:

```
$ hadoop fs -rm s3a://landsat-pds/scene_list.gz
rm: s3a://landsat-pds/scene_list.gz: delete on s3a://landsat-pds/scene_list.gz:
  com.amazonaws.services.s3.model.AmazonS3Exception: Access Denied (Service: Amazon S
  Status Code: 403; Error Code: AccessDenied; Request ID: 1EF98D5957BCAB3D),
  S3 Extended Request ID: wi3veOXFuFqWBUCJgV3Z+NQVj9gWgZVdXlPU4KBbYMsw/gA+hyhRXcaQ+Po
```

This demonstrates that the listed permissions cannot be taken as evidence of write access; only object manipulation can determine this.

Note that the Microsoft Azure WASB filesystem does allow permissions to be set and checked, however the permissions are not actually enforced. This feature offers the ability for a HDFS directory tree to be backed up with DistCp, with its permissions preserved, permissions which may be restored when copying the directory back into HDFS. For securing access to the data in the object store, however, Azure's own model and tools must be used .

Commands of limited value

Here is the list of shell commands which generally have no effect —and may actually fail.

command	limitations
`appendToFile`	generally unsupported
`checksum`	the usual checksum is “NONE”
`chgrp`	generally unsupported permissions model; no-op
`chmod`	generally unsupported permissions model; no-op
`chown`	generally unsupported permissions model; no-op
`createSnapshot`	generally unsupported
`deleteSnapshot`	generally unsupported
`df`	default values are normally displayed
`getfacl`	may or may not be supported
`getfattr`	generally supported
`renameSnapshot`	generally unsupported
`setfacl`	generally unsupported permissions model
`setfattr`	generally unsupported permissions model
`setrep`	has no effect
`truncate`	generally unsupported

Different object store clients *may* support these commands: do consult the documentation and test against the target store.