python-pachyderm

Joe Doliner

Nov 01, 2021

DOCS

CHAPTER

ONE

OVERVIEW

python-pachyderm is a Python client that interacts with Pachyderm, a tool for version-controlled, automated, end-to-end data pipelines for data science. If you're not familiar with Pachyderm or its value, check out that first!

1.1 python_pachyderm

1.1.1 Mixins

Information

Exposes a mixin for each pachyderm service. These mixins should not be used directly; instead, you should use `python_pachyderm.Client()`. The mixins exist exclusively in order to provide better code organization (because we have several mixins, rather than one giant *Client* class.)

python_pachyderm.mixin.admin

class python_pachyderm.mixin.admin.**AdminMixin**

Methods

extract([url, no_objects, no_repos, ...])	Extracts cluster data for backup.
extract_pipeline(pipeline_name)	Extracts a pipeline for backup.
inspect_cluster()	Inspects a cluster.
restore(requests)	Restores a cluster.

extract(*url=None*, *no_objects=None*, *no_repos=None*, *no_pipelines=None*, *no_auth=None*, *no_enterprise=None*)

Extracts cluster data for backup. Yields *Op* objects.

Parameters

url [str, optional] A string specifying an object storage URL. If set, data will be extracted to this URL rather than returned.

no_objects [bool, optional] If true, will cause extract to omit objects (and tags.)

no_repos [bool, optional] If true, will cause extract to omit repos, commits and branches.

no_pipelines [bool, optional] If true, will cause extract to omit pipelines.

no_auth [bool, optional] If true, will cause extract to omit acls, tokens, etc.

no_enterprise [bool, optional] If true, will cause extract to omit any enterprise activation key (which may break auth restore)

extract_pipeline(*pipeline_name*)
Extracts a pipeline for backup. Returns an *Op* object.

Parameters

pipeline_name [str] The pipeline name to extract.

inspect_cluster()
Inspects a cluster. Returns a *ClusterInfo* object.

restore(*requests*)
Restores a cluster.

Parameters

requests [Iterator[RestoreRequest protobufs]] A generator of *RestoreRequest* objects.

python_pachyderm.mixin.auth

class python_pachyderm.mixin.auth.**AuthMixin**

Methods

activate_auth(subject[, github_token, ...])	Activates auth, creating an initial set of admins.
authenticate_github(github_token)	Authenticates a GitHub user to the Pachyderm cluster.
authenticate_id_token(id_token)	Authenticates a user to the Pachyderm cluster using an ID token issued by the OIDC provider.
authenticate_oidc(oidc_state)	Authenticates a user to the Pachyderm cluster via OIDC.
authenticate_one_time_password(one_time_password)	Authenticates a user to the Pachyderm cluster using a one-time password.
authorize(repo, scope)	Authorizes the user to a given repo/scope.
deactivate_auth()	Deactivates auth, removing all ACLs, tokens, and admins from the Pachyderm cluster and making all data publicly accessible.
extend_auth_token(token, ttl)	Extends an existing auth token.
extract_auth_tokens()	This maps to an internal function that is only used for migration.
get_acl(repo)	Gets the ACL of a repo.
get_admins()	Returns a list of strings specifying the cluster admins.
get_auth_configuration()	Gets the auth configuration.
get_auth_token(subject[, ttl])	Gets an auth token for a subject.
get_cluster_role_bindings()	Returns the current set of cluster role bindings.
get_groups([username])	Gets which groups the given *username* belongs to.
get_oidc_login()	Returns the OIDC login configuration.
get_one_time_password([subject, ttl])	If this *Client* is authenticated as an admin, you can generate a one-time password for any given *subject*.
get_scope(username, repos)	Gets the auth scope.

continues on next page

Table 2 – continued from previous page

get_users(group)	Gets which users below to the *given*.
modify_admins([add, remove])	Adds and/or removes admins.
modify_cluster_role_binding(principal[, roles])	Sets the list of admin roles for a principal.
modify_members(group[, add, remove])	Adds and/or removes members of a group.
restore_auth_token([token])	This maps to an internal function that is only used for migration.
revoke_auth_token(token)	Revokes an auth token.
set_acl(repo, entries)	Sets the ACL of a repo.
set_auth_configuration(configuration)	Set the auth configuration.
set_groups_for_user(username, groups)	Sets the group membership for a user.
set_scope(username, repo, scope)	Set the auth scope.
who_am_i()	Returns info about the user tied to this *Client*.

activate_auth(*subject*, *github_token=None*, *root_token=None*)

Activates auth, creating an initial set of admins. Returns a string that can be used for making authenticated requests.

Parameters

subject [str] If set to a github user (i.e. it has a 'github:' prefix or no prefix) then Pachyderm will confirm that it matches the user associated with *github_token*. If set to a robot user (i.e. it has a 'robot:' prefix), then Pachyderm will generate a new token for the robot user; this token will be the only way to administer this cluster until more admins are added.

github_token [str, optional] This is the token returned by GitHub and used to authenticate the caller. When Pachyderm is deployed locally, setting this value to a given string will automatically authenticate the caller as a GitHub user whose username is that string (unless this "looks like" a GitHub access code, in which case Pachyderm does retrieve the corresponding GitHub username)

root_token [str, optional] Unused

authenticate_github(*github_token*)

Authenticates a GitHub user to the Pachyderm cluster. Returns a string that can be used for making authenticated requests.

Parameters

github_token: str This is the token returned by GitHub and used to authenticate the caller. When Pachyderm is deployed locally, setting this value to a given string will automatically authenticate the caller as a GitHub user whose username is that string (unless this "looks like" a GitHub access code, in which case Pachyderm does retrieve the corresponding GitHub username.)

authenticate_id_token(*id_token*)

Authenticates a user to the Pachyderm cluster using an ID token issued by the OIDC provider. The token must include the Pachyderm client_id in the set of audiences to be valid. Returns a string that can be used for making authenticated requests.

Parameters

id_token [str] The ID token.

authenticate_oidc(*oidc_state*)

Authenticates a user to the Pachyderm cluster via OIDC. Returns a string that can be used for making authenticated requests.

Parameters

oidc_state [str] The OIDC state token.

`authenticate_one_time_password`(*one_time_password*)
Authenticates a user to the Pachyderm cluster using a one-time password. Returns a string that can be used for making authenticated requests.

Parameters

one_time_password [str] This is a short-lived, one-time-use password generated by Pachyderm, for the purpose of propagating authentication to new clients (e.g. from the dash to pachd.)

`authorize`(*repo*, *scope*)
Authorizes the user to a given repo/scope. Return a bool specifying if the caller has at least *scope*-level access to *repo*.

Parameters

repo [str] The repo name that the caller wants access to.

scope [int] The access level that the caller needs to perform an action. See the `Scope` enum for variants.

`deactivate_auth`()
Deactivates auth, removing all ACLs, tokens, and admins from the Pachyderm cluster and making all data publicly accessible.

`extend_auth_token`(*token*, *ttl*)
Extends an existing auth token.

Parameters

token [str] Indicates the Pachyderm token whose TTL is being extended.

ttl [int] Indicates the approximate remaining lifetime of this token, in seconds.

`extract_auth_tokens`()
This maps to an internal function that is only used for migration. Pachyderm's *extract* and *restore* functionality calls *extract_auth_tokens* and *restore_auth_tokens* to move Pachyderm tokens between clusters during migration. Currently this function is only used for Pachyderm internals, so we're avoiding support for this function in python-pachyderm client until we find a use for it (feel free to file an issue in github.com/pachyderm/pachyderm).

`get_acl`(*repo*)
Gets the ACL of a repo. Returns a `GetACLResponse` object.

Parameters

repo [str] The repo to get an ACL for.

`get_admins`()
Returns a list of strings specifying the cluster admins.

`get_auth_configuration`()
Gets the auth configuration. Returns an *AuthConfig* object.

`get_auth_token`(*subject*, *ttl=None*)
Gets an auth token for a subject. Returns an `GetAuthTokenResponse` object.

Parameters

subject [str] The returned token will allow the caller to access resources as this subject.

ttl [int, optional] Indicates the approximate remaining lifetime of this token, in seconds.

`get_cluster_role_bindings()`
Returns the current set of cluster role bindings.

`get_groups(username=None)`
Gets which groups the given *username* belongs to. Returns a list of strings.

Parameters

username [str, optional] The username.

`get_oidc_login()`
Returns the OIDC login configuration.

`get_one_time_password(subject=None, ttl=None)`
If this `Client` is authenticated as an admin, you can generate a one-time password for any given *subject*. If the caller is not an admin or the *subject* is not set, a one-time password will be returned for logged-in subject. Returns a string.

Parameters

subject [str, optional] The subject.

ttl [int, optional] Indicates the approximate remaining lifetime of this token, in seconds.

`get_scope(username, repos)`
Gets the auth scope. Returns a list of *Scope* objects.

Parameters

username [str] A string specifying the principal (some of which belong to robots rather than users, but the name is preserved for now to provide compatibility with the pachyderm dash) whose access level is queried. To query the access level of a robot user, the caller must prefix username with "robot:". If *username* has no prefix (i.e. no ":"), then it's assumed to be a github user's principal.

repos [List[str]] A list of strings specifying the objects to which `username`s access level is being queried

`get_users(group)`
Gets which users below to the *given*. Returns a list of strings.

Parameters

group [str] The group to list users for.

`modify_admins(add=None, remove=None)`
Adds and/or removes admins.

Parameters

add [List[str], optional] A list of strings specifying admins to add.

remove [List[str], optional] A list of strings specifying admins to remove.

`modify_cluster_role_binding(principal, roles=None)`
Sets the list of admin roles for a principal.

Parameters

principal [str, optional] A string specifying the principal.

roles [ClusterRoles protobuf] A *ClusterRoles* object specifying cluster-wide permissions the principal has. If unspecified, all roles are revoked for the principal.

`modify_members`(*group*, *add=None*, *remove=None*)
Adds and/or removes members of a group.

Parameters

group [str] The group to modify.

add [List[str], optional] A list of strings specifying members to add.

remove [List[str], optional] A list of strings specifying members to remove.

`restore_auth_token`(*token=None*)
This maps to an internal function that is only used for migration. Pachyderm's *extract* and *restore* functionality calls *extract_auth_tokens* and *restore_auth_tokens* to move Pachyderm tokens between clusters during migration. Currently this function is only used for Pachyderm internals, so we're avoiding support for this function in python-pachyderm client until we find a use for it (feel free to file an issue in github.com/pachyderm/pachyderm).

`revoke_auth_token`(*token*)
Revokes an auth token.

Parameters

token [str] Indicates the Pachyderm token that is being revoked.

`set_acl`(*repo*, *entries*)
Sets the ACL of a repo.

Parameters

repo [str] The repo to set an ACL on.

entries [List[ACLEntry protobuf]] A list of *ACLEntry* objects.

`set_auth_configuration`(*configuration*)
Set the auth configuration.

Parameters

config [AuthConfig protobuf] The auth configuration.

`set_groups_for_user`(*username*, *groups*)
Sets the group membership for a user.

Parameters

username [str] The username.

groups [List[str]] The groups to add *username* to.

`set_scope`(*username*, *repo*, *scope*)
Set the auth scope.

Parameters

username [str] A string specifying the principal (some of which belong to robots rather than users, but the name is preserved for now to provide compatibility with the pachyderm dash) whose access level is queried. To query the access level of a robot user, the caller must prefix username with "robot:". If 'username' has no prefix (i.e. no ":"), then it's assumed to be a github user's principal.

repo [str] A string specifying the object to which `username`s access level is being granted/revoked.

scope [int] The access level that *username* will now have. See the `Scope` enum for variants.

who_am_i()
Returns info about the user tied to this *Client*.

python_pachyderm.mixin.debug

class python_pachyderm.mixin.debug.**DebugMixin**

Methods

binary([filter])	Gets the pachd binary.
dump([filter, limit])	Gets a debug dump.
profile_cpu(duration[, filter])	Gets a CPU profile.

binary(*filter=None*)
Gets the pachd binary. Yields byte arrays.

Parameters

filter [Filter protobuf, optional] An optional *Filter* object.

dump(*filter=None*, *limit=None*)
Gets a debug dump. Yields byte arrays.

Parameters

filter [Filter protobuf, optional] An optional *Filter* object.

limit [int, optional] Limits the number of commits/jobs returned for each repo/pipeline in the dump

profile_cpu(*duration*, *filter=None*)
Gets a CPU profile. Yields byte arrays.

Parameters

duration [Duration protobuf] A `Duration` object specifying how long to run the CPU profiler.

filter [Filter protobuf, optional] An optional *Filter* object.

python_pachyderm.mixin.enterprise

class python_pachyderm.mixin.enterprise.**EnterpriseMixin**

Methods

activate_enterprise(activation_code[, expires])	Activates enterprise.
deactivate_enterprise()	Deactivates enterprise.
get_activation_code()	Returns the enterprise code used to activate Pachdyerm Enterprise in this cluster.
get_enterprise_state()	Gets the current enterprise state of the cluster.

activate_enterprise(*activation_code*, *expires=None*)
Activates enterprise. Returns a *TokenInfo* object.

Parameters

activation_code [str] Specifies a Pachyderm enterprise activation code. New users can obtain trial activation codes.

expires [Timestamp protobuf, optional] An optional `Timestamp` object indicating when this activation code will expire. This should not generally be set (it's primarily used for testing), and is only applied if it's earlier than the signed expiration time in *activation_code*.

deactivate_enterprise()
Deactivates enterprise.

get_activation_code()
Returns the enterprise code used to activate Pachdyerm Enterprise in this cluster.

get_enterprise_state()
Gets the current enterprise state of the cluster. Returns a `GetEnterpriseResponse` object.

python_pachyderm.mixin.health

class python_pachyderm.mixin.health.**HealthMixin**

Methods

health()	Returns a health check indicating if the server can handle RPCs.

health()
Returns a health check indicating if the server can handle RPCs.

python_pachyderm.mixin.pfs

class python_pachyderm.mixin.pfs.**AtomicOp**(*commit*, *path*, ***kwargs*)
Represents an operation in a `PutFile` call.

Methods

reqs()	Yields one or more protobuf `PutFileRequests`, which are then enqueued into the request's channel.

reqs()
Yields one or more protobuf `PutFileRequests`, which are then enqueued into the request's channel.

class python_pachyderm.mixin.pfs.**AtomicPutFileobjOp**(*commit*, *path*, *value*, ***kwargs*)
A `PutFile` operation to put a file from a file-like object.

Methods

reqs()	Yields one or more protobuf `PutFileRequests`, which are then enqueued into the request's channel.

reqs()
Yields one or more protobuf `PutFileRequests`, which are then enqueued into the request's channel.

class python_pachyderm.mixin.pfs.**AtomicPutFilepathOp**(*commit*, *pfs_path*, *local_path*, ***kwargs*)
A `PutFile` operation to put a file locally stored at a given path. This file is opened on-demand, which helps with minimizing the number of open files.

Methods

reqs()	Yields one or more protobuf `PutFileRequests`, which are then enqueued into the request's channel.

reqs()
Yields one or more protobuf `PutFileRequests`, which are then enqueued into the request's channel.

class python_pachyderm.mixin.pfs.**PFSFile**(*res*)
The contents of a file stored in PFS.

Examples

You can treat these as either file-like objects, like so:

```
>>> source_file = client.get_file("montage/master", "/montage.png")
>>> with open("montage.png", "wb") as dest_file:
>>>     shutil.copyfileobj(source_file, dest_file)
```

Or as an iterator of bytes, like so:

```
>>> source_file = client.get_file("montage/master", "/montage.png")
>>> with open("montage.png", "wb") as dest_file:
>>>     for chunk in source_file:
>>>         dest_file.write(chunk)
```

Methods

close()	Closes the *PFSFile*
read([size])	Reads from the *PFSFile* buffer.

close()
Closes the *PFSFile*

read(*size=- 1*)
Reads from the *PFSFile* buffer.

Parameters

size [int, optional] The number of bytes to read from the buffer.

class python_pachyderm.mixin.pfs.**PFSMixin**

Methods

commit(repo_name[, branch, parent, description])	A context manager for running operations within a commit.
copy_file(source_commit, source_path, ...[, ...])	Efficiently copies files already in PFS.
create_branch(repo_name, branch_name[, ...])	Creates a new branch.
create_repo(repo_name[, description, update])	Creates a new `Repo` object in PFS with the given name. Repos are the top level data object in PFS and should be used to store data of a similar type. For example rather than having a single `Repo` for an entire project you might have separate ``Repo``s for logs, metrics, database dumps etc.
create_tmp_file_set()	Creates a temporary fileset (used internally).
delete_all_repos([force])	Deletes all repos.
delete_branch(repo_name, branch_name[, force])	Deletes a branch, but leaves the commits themselves intact.
delete_commit(commit)	Deletes a commit.
delete_file(commit, path)	Deletes a file from a Commit.
delete_repo(repo_name[, force, ...])	Deletes a repo and reclaims the storage space it was using.
diff_file(new_commit, new_path[, ...])	Diffs two files.
finish_commit(commit[, description, ...])	Ends the process of committing data to a Repo and persists the Commit.
flush_commit(commits[, repos])	Blocks until all of the commits which have a set of commits as provenance have finished.
fsck([fix])	Performs a file system consistency check for PFS.
get_file(commit, path[, offset_bytes, ...])	Returns a *PFSFile* object, containing the contents of a file stored in PFS.
glob_file(commit, pattern)	Lists files that match a glob pattern.
inspect_branch(repo_name, branch_name)	Inspects a branch.
inspect_commit(commit[, block_state])	Inspects a commit.
inspect_file(commit, path)	Inspects a file.
inspect_repo(repo_name)	Returns info about a specific repo.
list_branch(repo_name[, reverse])	Lists the active branch objects on a repo.
list_commit(repo_name[, to_commit, ...])	Lists commits.
list_file(commit, path[, history, ...])	
list_repo()	Returns info about all repos, as a list of `RepoInfo` objects.
put_file_bytes(commit, path, value[, ...])	Uploads a PFS file from a file-like object, bytestring, or iterator of bytestrings.
put_file_client()	A context manager that gives a *PutFileClient*.
put_file_url(commit, path, url[, delimiter, ...])	Puts a file using the content found at a URL.
renew_tmp_file_set(fileset_id, ttl_seconds)	Renews a temporary fileset (used internally).
start_commit(repo_name[, branch, parent, ...])	Begins the process of committing data to a Repo.
subscribe_commit(repo_name, branch[, ...])	Yields `CommitInfo` objects as commits occur.
walk_file(commit, path)	Walks over all descendant files in a directory.

`commit`(*repo_name*, *branch=None*, *parent=None*, *description=None*)
A context manager for running operations within a commit.

Parameters

repo_name [str] The name of the repo.

branch [str, optional] The branch name. This is a more convenient way to build linear chains of commits. When a commit is started with a non-empty branch the value of branch becomes an alias for the created Commit. This enables a more intuitive access pattern. When the commit is started on a branch the previous head of the branch is used as the parent of the commit.

parent [Union[tuple, str, Commit probotuf], optional] An optional `Commit` object specifying the parent commit. Upon creation the new commit will appear identical to the parent commit, data can safely be added to the new commit without affecting the contents of the parent commit.

description [str, optional] Description of the commit.

`copy_file`(*source_commit*, *source_path*, *dest_commit*, *dest_path*, *overwrite=None*)
Efficiently copies files already in PFS. Note that the destination repo cannot be an output repo, or the copy operation will (as of 1.9.0) silently fail.

Parameters

source_commit [Union[tuple, str, Commit protobuf]] Represents the commit with the source file.

source_path [str] The path of the source file.

dest_commit [Union[tuple, str, Commit protobuf]] Represents the commit for the destination file.

dest_path [str] The path of the destination file.

overwrite [bool, optional] Whether to overwrite the destination file if it already exists.

`create_branch`(*repo_name*, *branch_name*, *commit=None*, *provenance=None*, *trigger=None*)
Creates a new branch.

Parameters

repo_name [str] The name of the repo.

branch_name [str] The new branch name.

commit [Union[tuple, str, Commit protobuf], optional] Represents the head commit of the new branch.

provenance [List[Branch protobuf], optional] An optional iterable of *Branch* objects representing the branch provenance.

trigger [Trigger protobuf, optional] An optional *Trigger* object controlling when the head of *branch_name* is moved.

`create_repo`(*repo_name*, *description=None*, *update=None*)
Creates a new `Repo` object in PFS with the given name. Repos are the top level data object in PFS and should be used to store data of a similar type. For example rather than having a single `Repo` for an entire project you might have separate ``Repo``s for logs, metrics, database dumps etc.

Parameters

repo_name [str] Name of the repo.

description [str, optional] Description of the repo.

update [bool, optional] Whether to update if the repo already exists.

`create_tmp_file_set()`
Creates a temporary fileset (used internally). Currently, temp-fileset-related APIs are only used for Pachyderm internals (job merging), so we're avoiding support for these functions until we find a use for them (feel free to file an issue in github.com/pachyderm/pachyderm)

`delete_all_repos(force=None)`
Deletes all repos.

Parameters

force [bool, optional] If set to true, the repo will be removed regardless of errors. This argument should be used with care.

`delete_branch(repo_name, branch_name, force=None)`
Deletes a branch, but leaves the commits themselves intact. In other words, those commits can still be accessed via commit IDs and other branches they happen to be on.

Parameters

repo_name [str] The repo name.

branch_name [str] The name of the branch to delete.

force [bool, optional] Whether to force the branch deletion.

`delete_commit(commit)`
Deletes a commit.

Parameters

commit [Union[tuple, str, Commit protobuf]] The commit to delete.

`delete_file(commit, path)`
Deletes a file from a Commit. DeleteFile leaves a tombstone in the Commit, assuming the file isn't written to later attempting to get the file from the finished commit will result in not found error. The file will of course remain intact in the Commit's parent.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path to the file.

`delete_repo(repo_name, force=None, split_transaction=None)`
Deletes a repo and reclaims the storage space it was using.

Parameters

repo_name [str] The name of the repo.

force [bool, optional] If set to true, the repo will be removed regardless of errors. This argument should be used with care.

split_transaction [bool, optional] Controls whether Pachyderm attempts to delete the entire repo in a single database transaction. Setting this to `True` can work around certain Pachyderm errors, but, if set, the `delete_repo()` call may need to be retried.

`diff_file(new_commit, new_path, old_commit=None, old_path=None, shallow=None)`
Diffs two files. If *old_commit* or *old_path* are not specified, the same path in the parent of the file specified by *new_commit* and *new_path* will be used.

Parameters

new_commit [Union[tuple, str, Commit protobuf]] Represents the commit for the new file.

new_path [str] The path of the new file.

old_commit [Union[tuple, str, Commit protobuf]] Represents the commit for the old file.

old_path [str] The path of the old file.

shallow [bool, optional] Whether to do a shallow diff.

`finish_commit`(*commit*, *description=None*, *input_tree_object_hash=None*, *tree_object_hashes=None*, *datum_object_hash=None*, *size_bytes=None*, *empty=None*)

Ends the process of committing data to a Repo and persists the Commit. Once a Commit is finished the data becomes immutable and future attempts to write to it with PutFile will error.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

description [str, optional] Description of this commit.

input_tree_object_hash [str, optional] Specifies an input tree object hash.

tree_object_hashes [List[str], optional] A list of zero or more strings specifying object hashes for the output trees.

datum_object_hash [str, optional] Specifies an object hash.

size_bytes [int, optional] An optional int.

empty [bool, optional] If set, the commit will be closed (its *finished* field will be set to the current time) but its *tree* will be left None.

`flush_commit`(*commits*, *repos=None*)

Blocks until all of the commits which have a set of commits as provenance have finished. For commits to be considered they must have all of the specified commits as provenance. This in effect waits for all of the jobs that are triggered by a set of commits to complete. It returns an error if any of the commits it's waiting on are cancelled due to one of the jobs encountering an error during runtime. Note that it's never necessary to call FlushCommit to run jobs, they'll run no matter what, FlushCommit just allows you to wait for them to complete and see their output once they do. This returns an iterator of CommitInfo objects.

Yields `CommitInfo` objects.

Parameters

commits [List[Union[tuple, str, Commit protobuf]]] The commits to flush.

repos [List[str], optional] An optional list of strings specifying repo names. If specified, only commits within these repos will be flushed.

`fsck`(*fix=None*)

Performs a file system consistency check for PFS.

`get_file`(*commit*, *path*, *offset_bytes=None*, *size_bytes=None*)

Returns a *PFSFile* object, containing the contents of a file stored in PFS.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path of the file.

offset_bytes [int, optional] Specifies the number of bytes that should be skipped in the beginning of the file.

size_bytes [int, optional] Limits the total amount of data returned, note you will get fewer bytes than *size_bytes* if you pass a value larger than the size of the file. If 0, then all of the data will be returned.

`glob_file`(*commit*, *pattern*)
Lists files that match a glob pattern. Yields `FileInfo` objects.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

pattern [str] The glob pattern.

`inspect_branch`(*repo_name*, *branch_name*)
Inspects a branch. Returns a `BranchInfo` object.

Parameters

repo_name [str] The repo name.

branch_name [str] The branch name.

`inspect_commit`(*commit*, *block_state=None*)
Inspects a commit. Returns a `CommitInfo` object.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

block_state [int, optional] Causes this method to block until the commit is in the desired commit state. See the `CommitState` enum.

`inspect_file`(*commit*, *path*)
Inspects a file. Returns a `FileInfo` object.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path to the file.

`inspect_repo`(*repo_name*)
Returns info about a specific repo. Returns a `RepoInfo` object.

Parameters

repo_name [str] Name of the repo.

`list_branch`(*repo_name*, *reverse=None*)
Lists the active branch objects on a repo. Returns a list of `BranchInfo` objects.

Parameters

repo_name [str] The repo name.

reverse [bool, optional] If true, returns branches oldest to newest.

`list_commit`(*repo_name*, *to_commit=None*, *from_commit=None*, *number=None*, *reverse=None*)
Lists commits. Yields `CommitInfo` objects.

Parameters

repo_name [str] If only *repo_name* is given, all commits in the repo are returned.

to_commit [Union[tuple, str, Commit protobuf], optional] Only the ancestors of *to*, including *to* itself, are considered.

from_commit [Union[tuple, str, Commit protobuf], optional] Only the descendants of *from*, including *from* itself, are considered.

number [int, optional] Determines how many commits are returned. If *number* is 0, all commits that match the aforementioned criteria are returned.

reverse [bool, optional] If true, returns commits oldest to newest.

`list_file`(*commit*, *path*, *history=None*, *include_contents=None*)
Lists the files in a directory.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path to the directory.

history [int, optional] Indicates how many historical versions you want returned. Semantics are:

- 0: Return the files as they are in *commit*
- 1: Return above and the files as they are in the last commit they were modified in.
- 2: etc.
- -1: Return all historical versions.

include_contents [bool, optional] If *True*, file contents are included.

`list_repo`()
Returns info about all repos, as a list of `RepoInfo` objects.

`put_file_bytes`(*commit*, *path*, *value*, *delimiter=None*, *target_file_datums=None*, *target_file_bytes=None*, *overwrite_index=None*, *header_records=None*)
Uploads a PFS file from a file-like object, bytestring, or iterator of bytestrings.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path in the repo the file(s) will be written to.

value [Union[bytes, BinaryIO]] The file contents as bytes, represented as a file-like object, bytestring, or iterator of bytestrings.

delimiter [int, optional] Causes data to be broken up into separate files by the delimiter e.g. if you used `Delimiter.CSV.value`, a separate PFS file will be created for each row in the input CSV file, rather than one large CSV file.

target_file_datums [int, optional] Specifies the target number of datums in each written file. It may be lower if data does not split evenly, but will never be higher, unless the value is 0.

target_file_bytes [int, optional] Specifies the target number of bytes in each written file, file may have more or fewer bytes than the target.

overwrite_index [int, optional] This is the object index where the write starts from. All existing objects starting from the index are deleted.

header_records [int, optional] An optional int for splitting data when *delimiter* is not `NONE` (or `SQL`). It specifies the number of records that are converted to a header and applied to all file shards.

`put_file_client`()
A context manager that gives a *PutFileClient*. When the context manager exits, any operations enqueued from the *PutFileClient* are executed in a single, atomic `PutFile` call.

`put_file_url`(*commit*, *path*, *url*, *delimiter=None*, *recursive=None*, *target_file_datums=None*, *target_file_bytes=None*, *overwrite_index=None*, *header_records=None*)

Puts a file using the content found at a URL. The URL is sent to the server which performs the request.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path in the repo the file will be written to.

url [str] The url of the file to put.

delimiter [int, optional] Causes data to be broken up into separate files by the delimiter e.g. if you used `Delimiter.CSV.value`, a separate PFS file will be created for each row in the input CSV file, rather than one large CSV file.

recursive [bool, optional] Allow for recursive scraping of some types URLs, for example on s3:// URLs.

target_file_datums [int, optional] Specifies the target number of datums in each written file. It may be lower if data does not split evenly, but will never be higher, unless the value is 0.

target_file_bytes [int, optional] Specifies the target number of bytes in each written file, file may have more or fewer bytes than the target.

overwrite_index [int, optional] This is the object index where the write starts from. All existing objects starting from the index are deleted.

header_records [int, optional] An optional int for splitting data when *delimiter* is not `NONE` (or `SQL`). It specifies the number of records that are converted to a header and applied to all file shards.

`renew_tmp_file_set`(*fileset_id*, *ttl_seconds*)

Renews a temporary fileset (used internally). Currently, temp-fileset-related APIs are only used for Pachyderm internals (job merging), so we're avoiding support for these functions until we find a use for them (feel free to file an issue in github.com/pachyderm/pachyderm)

Parameters

fileset_id [str] The fileset ID.

ttl_seconds [int] The number of seconds to keep alive the temporary fileset.

`start_commit`(*repo_name*, *branch=None*, *parent=None*, *description=None*, *provenance=None*)

Begins the process of committing data to a Repo. Once started you can write to the Commit with PutFile and when all the data has been written you must finish the Commit with FinishCommit. NOTE, data is not persisted until FinishCommit is called. A Commit object is returned.

Parameters

repo_name [str] The name of the repo.

branch [str, optional] The branch name. This is a more convenient way to build linear chains of commits. When a commit is started with a non-empty branch the value of branch becomes an alias for the created Commit. This enables a more intuitive access pattern. When the commit is started on a branch the previous head of the branch is used as the parent of the commit.

parent [Union[tuple, str, Commit probotuf], optional] An optional `Commit` object specifying the parent commit. Upon creation the new commit will appear identical to the parent commit, data can safely be added to the new commit without affecting the contents of the parent commit.

description [str, optional] Description of the commit.

provenance [List[CommitProvenance protobuf], optional] An optional iterable of *CommitProvenance* objects specifying the commit provenance.

subscribe_commit(*repo_name*, *branch*, *from_commit_id=None*, *state=None*, *prov=None*)
Yields `CommitInfo` objects as commits occur.

Parameters

repo_name [str] The name of the repo.

branch [str] The branch to subscribe to.

from_commit_id [str, optional] A commit ID. Only commits created since this commit are returned.

state [int, optional] The commit state to filter on. See the `CommitState` enum.

prov [CommitProvenance protobuf, optional] An optional `CommitProvenance` object.

walk_file(*commit*, *path*)
Walks over all descendant files in a directory. Returns a generator of `FileInfo` objects.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path to the directory.

class python_pachyderm.mixin.pfs.**PutFileClient**
PutFileClient puts or deletes PFS files atomically.

Methods

delete_file(commit, path)	Deletes a file.
put_file_from_bytes(commit, path, value[, ...])	Uploads a PFS file from a bytestring.
put_file_from_fileobj(commit, path, value[, ...])	Uploads a PFS file from a file-like object.
put_file_from_filepath(commit, pfs_path, ...)	Uploads a PFS file from a local path at a specified path.
put_file_from_url(commit, path, url[, ...])	Puts a file using the content found at a URL.

delete_file(*commit*, *path*)
Deletes a file.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path to the file.

put_file_from_bytes(*commit*, *path*, *value*, *delimiter=None*, *target_file_datums=None*, *target_file_bytes=None*, *overwrite_index=None*, *header_records=None*)
Uploads a PFS file from a bytestring.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path in the repo to upload the file to will be written to.

value [bytes] The file contents as a bytestring.

delimiter [int, optional] Causes data to be broken up into separate files by the delimiter e.g. if you used `Delimiter.CSV.value`, a separate PFS file will be created for each row in the input CSV file, rather than one large CSV file.

target_file_datums [int, optional] Specifies the target number of datums in each written file. It may be lower if data does not split evenly, but will never be higher, unless the value is 0.

target_file_bytes [int, optional] Specifies the target number of bytes in each written file, file may have more or fewer bytes than the target.

overwrite_index [int, optional] This is the object index where the write starts from. All existing objects starting from the index are deleted.

header_records [int, optional] An optional int for splitting data when *delimiter* is not `NONE` (or `SQL`). It specifies the number of records that are converted to a header and applied to all file shards.

`put_file_from_fileobj`(*commit*, *path*, *value*, *delimiter=None*, *target_file_datums=None*, *target_file_bytes=None*, *overwrite_index=None*, *header_records=None*)

Uploads a PFS file from a file-like object.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path in the repo to upload the file to will be written to.

value [BinaryIO] The file-like object.

delimiter [int, optional] Causes data to be broken up into separate files by the delimiter e.g. if you used `Delimiter.CSV.value`, a separate PFS file will be created for each row in the input CSV file, rather than one large CSV file.

target_file_datums [int, optional] Specifies the target number of datums in each written file. It may be lower if data does not split evenly, but will never be higher, unless the value is 0.

target_file_bytes [int, optional] Specifies the target number of bytes in each written file, file may have more or fewer bytes than the target.

overwrite_index [int, optional] This is the object index where the write starts from. All existing objects starting from the index are deleted.

header_records [int, optional] An optional int for splitting data when *delimiter* is not `NONE` (or `SQL`). It specifies the number of records that are converted to a header and applied to all file shards.

`put_file_from_filepath`(*commit*, *pfs_path*, *local_path*, *delimiter=None*, *target_file_datums=None*, *target_file_bytes=None*, *overwrite_index=None*, *header_records=None*)

Uploads a PFS file from a local path at a specified path. This will lazily open files, which will prevent too many files from being opened, or too much memory being consumed, when atomically putting many files.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

pfs_path [str] The path in the repo to upload the file to will be written to.

local_path [str] The local file path.

delimiter [int, optional] Causes data to be broken up into separate files by the delimiter e.g. if you used `Delimiter.CSV.value`, a separate PFS file will be created for each row in the input CSV file, rather than one large CSV file.

target_file_datums [int, optional] Specifies the target number of datums in each written file. It may be lower if data does not split evenly, but will never be higher, unless the value is 0.

target_file_bytes [int, optional] Specifies the target number of bytes in each written file, file may have more or fewer bytes than the target.

overwrite_index [int, optional] This is the object index where the write starts from. All existing objects starting from the index are deleted.

header_records [int, optional] An optional int for splitting data when *delimiter* is not `NONE` (or `SQL`). It specifies the number of records that are converted to a header and applied to all file shards.

put_file_from_url(*commit*, *path*, *url*, *delimiter=None*, *recursive=None*, *target_file_datums=None*, *target_file_bytes=None*, *overwrite_index=None*, *header_records=None*)

Puts a file using the content found at a URL. The URL is sent to the server which performs the request.

Parameters

commit [Union[tuple, str, Commit protobuf]] Represents the commit.

path [str] The path in the repo the file will be written to.

url [str] The url of the file to put.

delimiter [int, optional] Causes data to be broken up into separate files by the delimiter e.g. if you used `Delimiter.CSV.value`, a separate PFS file will be created for each row in the input CSV file, rather than one large CSV file.

recursive [bool, optional] Allow for recursive scraping of some types URLs, for example on s3:// URLs.

target_file_datums [int, optional] Specifies the target number of datums in each written file. It may be lower if data does not split evenly, but will never be higher, unless the value is 0.

target_file_bytes [int, optional] Specifies the target number of bytes in each written file, file may have more or fewer bytes than the target.

overwrite_index [int, optional] This is the object index where the write starts from. All existing objects starting from the index are deleted.

header_records [int, optional] An optional int for splitting data when *delimiter* is not `NONE` (or `SQL`). It specifies the number of records that are converted to a header and applied to all file shards.

python_pachyderm.mixin.pfs.**put_file_from_fileobj_reqs**(*fileish*, ***kwargs*)

python_pachyderm.mixin.pfs.**put_file_from_iterable_reqs**(*value*, ***kwargs*)

python_pachyderm.mixin.pps

class python_pachyderm.mixin.pps.**PPSMixin**

Methods

create_pipeline(pipeline_name, transform[, ...])	Creates a pipeline.
create_pipeline_from_request(req)	Creates a pipeline from a `CreatePipelineRequest` object.
create_secret(secret_name, data[, labels, ...])	Creates a new secret.
create_tf_job_pipeline(pipeline_name, tf_job)	Creates a pipeline.
delete_all()	Deletes everything in Pachyderm.
delete_all_pipelines([force])	Deletes all pipelines.
delete_job(job_id)	Deletes a job by its ID.
delete_pipeline(pipeline_name[, force, ...])	Deletes a pipeline.
delete_secret(secret_name)	Deletes a secret.
flush_job(commits[, pipeline_names])	Blocks until all of the jobs which have a set of commits as provenance have finished.
garbage_collect([memory_bytes])	Runs garbage collection.
get_job_logs(job_id[, data_filters, datum, ...])	Gets logs for a job.
get_pipeline_logs(pipeline_name[, ...])	Gets logs for a pipeline.
inspect_datum(job_id, datum_id)	Inspects a datum.
inspect_job(job_id[, block_state, ...])	Inspects a job with a given ID.
inspect_pipeline(pipeline_name[, history])	
inspect_secret(secret_name)	Inspects a secret.
list_datum([job_id, page_size, page, input, ...])	Lists datums.
list_job([pipeline_name, input_commit, ...])	
list_pipeline([history, allow_incomplete, ...])	
list_secret()	Lists secrets.
restart_datum(job_id[, data_filters])	Restarts a datum.
run_cron(pipeline_name)	Explicitly triggers a pipeline with one or more cron inputs to run now.
run_pipeline(pipeline_name[, provenance, job_id])	Runs a pipeline.
start_pipeline(pipeline_name)	Starts a pipeline.
stop_job(job_id)	Stops a job by its ID.
stop_pipeline(pipeline_name)	Stops a pipeline.

create_pipeline(*pipeline_name*, *transform*, *parallelism_spec=None*, *hashtree_spec=None*, *egress=None*, *update=None*, *output_branch=None*, *resource_requests=None*, *resource_limits=None*, *input=None*, *description=None*, *cache_size=None*, *enable_stats=None*, *reprocess=None*, *max_queue_size=None*, *service=None*, *chunk_spec=None*, *datum_timeout=None*, *job_timeout=None*, *salt=None*, *standby=None*, *datum_tries=None*, *scheduling_spec=None*, *pod_patch=None*, *spout=None*, *spec_commit=None*, *metadata=None*, *s3_out=None*, *sidecar_resource_limits=None*, *reprocess_spec=None*, *autoscaling=None*)

Creates a pipeline. For more info, please refer to the pipeline spec document: http://docs.pachyderm.io/en/latest/reference/pipeline_spec.html

Parameters

pipeline_name [str] The pipeline name.

transform [Transform protobuf] A `Transform` object.

parallelism_spec [ParallelismSpec protobuf, optional] An optional `ParallelismSpec` object.

hashtree_spec [HashtreeSpec protobuf, optional] An optional `HashtreeSpec` object.

egress [Egress protobuf, optional] An optional `Egress` object.

update [bool, optional] Whether this should behave as an upsert.

output_branch [str, optional] The branch to output results on.

resource_requests [ResourceSpec protobuf, optional] An optional `ResourceSpec` object.

resource_limits [ResourceSpec protobuf, optional] An optional `ResourceSpec` object.

input [Input protobuf, optional] An optional `Input` object.

description [str, optional] Description of the pipeline.

cache_size [str, optional] An optional string.

enable_stats [bool, optional] An optional bool.

reprocess [bool, optional] If true, Pachyderm forces the pipeline to reprocess all datums. It only has meaning if *update* is `True`.

max_queue_size [int, optional] An optional int.

service [Service protobuf, optional] An optional `Service` object.

chunk_spec [ChunkSpec protobuf, optional] An optional `ChunkSpec` object.

datum_timeout [Duration protobuf, optional] An optional `Duration` object.

job_timeout [Duration protobuf, optional] An optional `Duration` object.

salt [str, optional] An optional string.

standby [bool, optional] An optional bool.

datum_tries [int, optional] An optional int.

scheduling_spec [SchedulingSpec protobuf, optional] An optional `SchedulingSpec` object.

pod_patch [str, optional] An optional string.

spout [Spout protobuf, optional] An optional `Spout` object.

spec_commit [Commit protobuf, optional] An optional `Commit` object.

metadata [Metadata protobuf, optional] An optional `Metadata` object.

s3_out [bool, optional] Unused.

sidecar_resource_limits [ResourceSpec protobuf, optional] An optional `ResourceSpec` setting resource limits for the pipeline sidecar.

`create_pipeline_from_request`(*req*)

Creates a pipeline from a `CreatePipelineRequest` object. Usually this would be used in conjunction with `util.parse_json_pipeline_spec()` or `util.parse_dict_pipeline_spec()`. If you're in pure python and not working with a pipeline spec file, the sibling method `create_pipeline()` is more ergonomic.

Parameters

req [CreatePipelineRequest protobuf] A *CreatePipelineRequest* object.

`create_secret`(*secret_name*, *data*, *labels=None*, *annotations=None*)
Creates a new secret.

Parameters

secret_name [str] The name of the secret to create.

data [Dict[str, Union[str, bytes]]] The data to store in the secret. Each key must consist of alphanumeric characters -, _ or .

labels [Dict[str, str], optional] Kubernetes labels to attach to the secret.

annotations [Dict[str, str], optional] Kubernetes annotations to attach to the secret.

`create_tf_job_pipeline`(*pipeline_name*, *tf_job*, *parallelism_spec=None*, *hashtree_spec=None*, *egress=None*, *update=None*, *output_branch=None*, *scale_down_threshold=None*, *resource_requests=None*, *resource_limits=None*, *input=None*, *description=None*, *cache_size=None*, *enable_stats=None*, *reprocess=None*, *max_queue_size=None*, *service=None*, *chunk_spec=None*, *datum_timeout=None*, *job_timeout=None*, *salt=None*, *standby=None*, *datum_tries=None*, *scheduling_spec=None*, *pod_patch=None*, *spout=None*, *spec_commit=None*)
Creates a pipeline. For more info, please refer to the pipeline spec document: http://docs.pachyderm.io/en/latest/reference/pipeline_spec.html

Parameters

pipeline_name [str] The pipeline name.

tf_job [TFJob protobuf] Pachyderm uses this to create TFJobs when running in a Kubernetes cluster on which kubeflow has been installed.

parallelism_spec [ParallelismSpec protobuf, optional] An optional `ParallelismSpec` object.

hashtree_spec [HashtreeSpec protobuf, optional] An optional `HashtreeSpec` object.

egress [Egress protobuf, optional] An optional `Egress` object.

update [bool, optional] Whether this should behave as an upsert.

output_branch [str, optional] The branch to output results on.

scale_down_threshold [Duration protobuf, optional] An optional *Duration* object.

resource_requests [ResourceSpec protobuf, optional] An optional `ResourceSpec` object.

resource_limits [ResourceSpec protobuf, optional] An optional `ResourceSpec` object.

input [Input protobuf, optional] An optional `Input` object.

description [str, optional] Description of the pipeline.

cache_size [str, optional] An optional string.

enable_stats [bool, optional] An optional bool.

reprocess [bool, optional] If true, Pachyderm forces the pipeline to reprocess all datums. It only has meaning if *update* is `True`.

max_queue_size [int, optional] An optional int.

service [Service protobuf, optional] An optional `Service` object.

chunk_spec [ChunkSpec protobuf, optional] An optional `ChunkSpec` object.

datum_timeout [Duration protobuf, optional] An optional `Duration` object.

job_timeout [Duration protobuf, optional] An optional `Duration` object.

salt [str, optional] An optional string.

standby [bool, optional] An optional bool.

datum_tries [int, optional] An optional int.

scheduling_spec [SchedulingSpec protobuf, optional] An optional `SchedulingSpec` object.

pod_patch [str, optional] An optional string.

spout [Spout protobuf, optional] An optional `Spout` object.

spec_commit [Commit protobuf, optional] An optional `Commit` object.

`delete_all`()
Deletes everything in Pachyderm.

`delete_all_pipelines`(*force=None*)
Deletes all pipelines.

Parameters

force [bool, optional] Whether to force delete.

`delete_job`(*job_id*)
Deletes a job by its ID.

Parameters

job_id [str] The ID of the job to delete.

`delete_pipeline`(*pipeline_name*, *force=None*, *keep_repo=None*, *split_transaction=None*)
Deletes a pipeline.

Parameters

pipeline_name [str] The pipeline name.

force [bool, optional] Whether to force delete.

keep_repo [bool, optional] Whether to keep the output repo.

split_transaction [bool, optional] Whether Pachyderm attempts to delete the pipeline in a single database transaction. Setting this to `True` can work around certain Pachyderm errors, but, if set, the ``delete_repo()` call may need to be retried.

`delete_secret`(*secret_name*)
Deletes a secret.

Parameters

secret_name [str] The name of the secret to delete.

`flush_job`(*commits*, *pipeline_names=None*)
Blocks until all of the jobs which have a set of commits as provenance have finished. Yields `JobInfo` objects.

Parameters

commits [List[Union[tuple, str, Commit protobuf]]] A list representing the commits to flush.

pipeline_names [List[str], optional] A list of strings specifying pipeline names. If specified, only jobs within these pipelines will be flushed.

`garbage_collect`(*memory_bytes=None*)
Runs garbage collection.

Parameters

memory_bytes [int, optional] How much memory to use in computing which objects are alive. A larger number will result in more precise garbage collection (at the cost of more memory usage).

`get_job_logs`(*job_id*, *data_filters=None*, *datum=None*, *follow=None*, *tail=None*, *use_loki_backend=None*, *since=None*)
Gets logs for a job. Yields *LogMessage* objects.

Parameters

job_id [str] The ID of the job.

data_filters [List[str], optional] A list of the names of input files from which we want processing logs. This may contain multiple files, in case *pipeline_name* contains multiple inputs. Each filter may be an absolute path of a file within a repo, or it may be a hash for that file (to search for files at specific versions).

datum [Datum protobuf, optional] Filters log lines for the specified datum.

follow [bool, optional] If true, continue to follow new logs as they appear.

tail [int, optional] If nonzero, the number of lines from the end of the logs to return. Note: tail applies per container, so you will get *tail* * <number of pods> total lines back.

use_loki_backend [bool, optional] If true, use loki as a backend, rather than Kubernetes, for fetching logs. Requires a loki-enabled cluster.

since [Duration protobuf, optional] Specifies how far in the past to return logs from.

`get_pipeline_logs`(*pipeline_name*, *data_filters=None*, *master=None*, *datum=None*, *follow=None*, *tail=None*, *use_loki_backend=None*, *since=None*)
Gets logs for a pipeline. Yields `LogMessage` objects.

Parameters

pipeline_name [str] The name of the pipeline.

data_filters [List[str], optional] A list of the names of input files from which we want processing logs. This may contain multiple files, in case *pipeline_name* contains multiple inputs. Each filter may be an absolute path of a file within a repo, or it may be a hash for that file (to search for files at specific versions).

master [bool, optional] If true, includes logs from the master

datum [Datum protobuf, optional] Filters log lines for the specified datum.

follow [bool, optional] If true, continue to follow new logs as they appear.

tail [int, optional] If nonzero, the number of lines from the end of the logs to return. Note: tail applies per container, so you will get *tail* * <number of pods> total lines back.

use_loki_backend [bool, optional] If true, use loki as a backend, rather than Kubernetes, for fetching logs. Requires a loki-enabled cluster.

since [Duration protobuf, optional] Specifies how far in the past to return logs from.

`inspect_datum`(*job_id*, *datum_id*)
Inspects a datum. Returns a `DatumInfo` object.

Parameters

job_id [str] The ID of the job.

datum_id [str] The ID of the datum.

`inspect_job`(*job_id*, *block_state=None*, *output_commit=None*, *full=None*)
Inspects a job with a given ID. Returns a `JobInfo`.

Parameters

job_id [str] The ID of the job to inspect.

block_state [bool, optional] If true, block until the job completes.

output_commit [Union[tuple, str, Commit protobuf], optional] Represents an output commit to filter on.

full [bool, optional] If true, include worker status.

`inspect_pipeline`(*pipeline_name*, *history=None*)
Inspects a pipeline. Returns a `PipelineInfo` object.

Parameters

pipeline_name [str] The pipeline name.

history [int, optional] Indicates to return historical versions of pipelines. Semantics are:

- 0: Return current version of pipelines.
- 1: Return the above and pipelines from the next most recent version.
- 2: etc.
- -1: Return pipelines from all historical versions.

`inspect_secret`(*secret_name*)
Inspects a secret.

Parameters

secret_name [str] The name of the secret to inspect.

`list_datum`(*job_id=None*, *page_size=None*, *page=None*, *input=None*, *status_only=None*)
Lists datums. Yields `ListDatumStreamResponse` objects.

Parameters

job_id [str, optional] The ID of a job. Exactly one of *job_id* (real) or *input* (hypothetical) must be set.

page_size [int, optional] The size of the page.

page [int, optional] The page number.

input [Input protobuf, optional] If set in lieu of *job_id*, `list_datum()` returns the datums that would be given to a hypothetical job that used *input* as its input spec. Exactly one of *job_id* (real) or *input* (hypothetical) must be set.

`list_job`(*pipeline_name=None*, *input_commit=None*, *output_commit=None*, *history=None*, *full=None*, *jqFilter=None*)
Lists jobs. Yields `JobInfo` objects.

Parameters

pipeline_name [str, optional] A pipeline name to filter on.

input_commit [List[Union[tuple, str, Commit protobuf]], optional] An optional list representing input commits to filter on.

output_commit [Union[tuple, str, Commit protobuf], optional] Represents an output commit to filter on.

history [int, optional] Indicates to return jobs from historical versions of pipelines. Semantics are:

- 0: Return jobs from the current version of the pipeline or pipelines.
- 1: Return the above and jobs from the next most recent version
- 2: etc.
- -1: Return jobs from all historical versions.

full [bool, optional] Whether the result should include all pipeline details in each `JobInfo`, or limited information including name and status, but excluding information in the pipeline spec. Leaving this `None` (or `False`) can make the call significantly faster in clusters with a large number of pipelines and jobs. Note that if *input_commit* is set, this field is coerced to `True`.

jqFilter [str, optional] A `jq` filter that can restrict the list of jobs returned.

`list_pipeline`(*history=None*, *allow_incomplete=None*, *jqFilter=None*)
Lists pipelines. Returns a *PipelineInfos* object.

Parameters

history [int, optional] Indicates to return historical versions of pipelines. Semantics are:

- 0: Return current version of pipelines.
- 1: Return the above and pipelines from the next most recent version.
- 2: etc.
- -1: Return pipelines from all historical versions.

allow_incomplete [bool, optional] If True, causes `list_pipeline()` to return `PipelineInfos` with incomplete data where the pipeline spec cannot be retrieved. Incomplete `PipelineInfos` will have a `None` *Transform* field, but will have the fields present in `EtcdPipelineInfo`.

jqFilter [str, optional] A `jq` filter that can restrict the list of pipelines returned.

`list_secret`()
Lists secrets. Returns a list of `SecretInfo` objects.

`restart_datum`(*job_id*, *data_filters=None*)
Restarts a datum.

Parameters

job_id [str] The ID of the job.

data_filters [List[str], optional] An optional iterable of strings.

`run_cron`(*pipeline_name*)
Explicitly triggers a pipeline with one or more cron inputs to run now.

Parameters

pipeline_name [str] The pipeline name.

`run_pipeline`(*pipeline_name*, *provenance=None*, *job_id=None*)
Runs a pipeline.

Parameters

pipeline_name [str] The pipeline name.

provenance [List[CommitProvenance protobuf], optional] A list representing the pipeline execution provenance.

job_id [str, optional] A specific job ID to run.

`start_pipeline`(*pipeline_name*)
Starts a pipeline.

Parameters

pipeline_name [str] The pipeline name.

`stop_job`(*job_id*)
Stops a job by its ID.

Parameters

job_id [str] The ID of the job to stop.

`stop_pipeline`(*pipeline_name*)
Stops a pipeline.

Parameters

pipeline_name [str] The pipeline name.

`python_pachyderm.mixin.pps.pipeline_inputs`(*root*)

python_pachyderm.mixin.transaction

`class python_pachyderm.mixin.transaction.TransactionMixin`

Methods

batch_transaction(requests)	Executes a batch transaction.
delete_all_transactions()	Deletes all transactions.
delete_transaction(transaction)	Deletes a given transaction.
finish_transaction(transaction)	Finishes a given transaction.
inspect_transaction(transaction)	Inspects a given transaction.
list_transaction()	Lists transactions.
start_transaction()	Starts a transaction.
transaction()	A context manager for running operations within a transaction.

`batch_transaction`(*requests*)
Executes a batch transaction.

Parameters

requests [List[TransactionRequest protobuf]] A list of *TransactionRequest* objects.

`delete_all_transactions`()
Deletes all transactions.

`delete_transaction`(*transaction*)
Deletes a given transaction.

Parameters

transaction [Union[str, Transaction protobuf]] Transaction ID or `Transaction` object.

finish_transaction(*transaction*)
Finishes a given transaction.

Parameters

transaction [Union[str, Transaction protobuf]] Transaction ID or `Transaction` object.

inspect_transaction(*transaction*)
Inspects a given transaction.

Parameters

transaction [Union[str, Transaction protobuf]] Transaction ID or `Transaction` object.

list_transaction()
Lists transactions.

start_transaction()
Starts a transaction.

transaction()
A context manager for running operations within a transaction. When the context manager completes, the transaction will be deleted if an error occurred, or otherwise finished.

python_pachyderm.mixin.transaction.**transaction_from**(*transaction*)

python_pachyderm.mixin.util

python_pachyderm.mixin.util.**commit_from**(*src*, *allow_just_repo=False*)

python_pachyderm.mixin.version

class python_pachyderm.mixin.version.**VersionMixin**

Methods

get_remote_version()	Gets version of Pachyderm server.

get_remote_version()
Gets version of Pachyderm server.

1.1.2 Client

class python_pachyderm.client.**Client**(*host=None*, *port=None*, *auth_token=None*, *root_certs=None*, *transaction_id=None*, *tls=None*)

Bases: *python_pachyderm.mixin.admin.AdminMixin*, *python_pachyderm.mixin.auth.AuthMixin*, *python_pachyderm.mixin.debug.DebugMixin*, *python_pachyderm.mixin.enterprise.EnterpriseMixin*, *python_pachyderm.mixin.health.HealthMixin*, *python_pachyderm.mixin.pfs.PFSMixin*, *python_pachyderm.mixin.pps.PPSMixin*, *python_pachyderm.mixin.transaction.TransactionMixin*, *python_pachyderm.mixin.version.VersionMixin*, `object`

Attributes

auth_token

transaction_id

Methods

`activate_auth`(subject[, github_token, ...])	Activates auth, creating an initial set of admins.
`activate_enterprise`(activation_code[, expires])	Activates enterprise.
`authenticate_github`(github_token)	Authenticates a GitHub user to the Pachyderm cluster.
`authenticate_id_token`(id_token)	Authenticates a user to the Pachyderm cluster using an ID token issued by the OIDC provider.
`authenticate_oidc`(oidc_state)	Authenticates a user to the Pachyderm cluster via OIDC.
`authenticate_one_time_password`(one_time_password)	Authenticates a user to the Pachyderm cluster using a one-time password.
`authorize`(repo, scope)	Authorizes the user to a given repo/scope.
`batch_transaction`(requests)	Executes a batch transaction.
`binary`([filter])	Gets the pachd binary.
`commit`(repo_name[, branch, parent, description])	A context manager for running operations within a commit.
`copy_file`(source_commit, source_path, ...[, ...])	Efficiently copies files already in PFS.
`create_branch`(repo_name, branch_name[, ...])	Creates a new branch.
`create_pipeline`(pipeline_name, transform[, ...])	Creates a pipeline.
`create_pipeline_from_request`(req)	Creates a pipeline from a `CreatePipelineRequest` object.
`create_repo`(repo_name[, description, update])	Creates a new `Repo` object in PFS with the given name. Repos are the top level data object in PFS and should be used to store data of a similar type. For example rather than having a single `Repo` for an entire project you might have separate ``Repo``s for logs, metrics, database dumps etc.
`create_secret`(secret_name, data[, labels, ...])	Creates a new secret.
`create_tf_job_pipeline`(pipeline_name, tf_job)	Creates a pipeline.
`create_tmp_file_set`()	Creates a temporary fileset (used internally).
`deactivate_auth`()	Deactivates auth, removing all ACLs, tokens, and admins from the Pachyderm cluster and making all data publicly accessible.
`deactivate_enterprise`()	Deactivates enterprise.
`delete_all`()	Deletes everything in Pachyderm.

continues on next page

Table 15 – continued from previous page

delete_all_pipelines([force])	Deletes all pipelines.
delete_all_repos([force])	Deletes all repos.
delete_all_transactions()	Deletes all transactions.
delete_branch(repo_name, branch_name[, force])	Deletes a branch, but leaves the commits themselves intact.
delete_commit(commit)	Deletes a commit.
delete_file(commit, path)	Deletes a file from a Commit.
delete_job(job_id)	Deletes a job by its ID.
delete_pipeline(pipeline_name[, force, ...])	Deletes a pipeline.
delete_repo(repo_name[, force, ...])	Deletes a repo and reclaims the storage space it was using.
delete_secret(secret_name)	Deletes a secret.
delete_transaction(transaction)	Deletes a given transaction.
diff_file(new_commit, new_path[, ...])	Diffs two files.
dump([filter, limit])	Gets a debug dump.
extend_auth_token(token, ttl)	Extends an existing auth token.
extract([url, no_objects, no_repos, ...])	Extracts cluster data for backup.
extract_auth_tokens()	This maps to an internal function that is only used for migration.
extract_pipeline(pipeline_name)	Extracts a pipeline for backup.
finish_commit(commit[, description, ...])	Ends the process of committing data to a Repo and persists the Commit.
finish_transaction(transaction)	Finishes a given transaction.
flush_commit(commits[, repos])	Blocks until all of the commits which have a set of commits as provenance have finished.
flush_job(commits[, pipeline_names])	Blocks until all of the jobs which have a set of commits as provenance have finished.
fsck([fix])	Performs a file system consistency check for PFS.
garbage_collect([memory_bytes])	Runs garbage collection.
get_acl(repo)	Gets the ACL of a repo.
get_activation_code()	Returns the enterprise code used to activate Pachdyerm Enterprise in this cluster.
get_admins()	Returns a list of strings specifying the cluster admins.
get_auth_configuration()	Gets the auth configuration.
get_auth_token(subject[, ttl])	Gets an auth token for a subject.
get_cluster_role_bindings()	Returns the current set of cluster role bindings.
get_enterprise_state()	Gets the current enterprise state of the cluster.
get_file(commit, path[, offset_bytes, ...])	Returns a *PFSFile* object, containing the contents of a file stored in PFS.
get_groups([username])	Gets which groups the given *username* belongs to.
get_job_logs(job_id[, data_filters, datum, ...])	Gets logs for a job.
get_oidc_login()	Returns the OIDC login configuration.
get_one_time_password([subject, ttl])	If this *Client* is authenticated as an admin, you can generate a one-time password for any given *subject*.
get_pipeline_logs(pipeline_name[, ...])	Gets logs for a pipeline.
get_remote_version()	Gets version of Pachyderm server.
get_scope(username, repos)	Gets the auth scope.
get_users(group)	Gets which users below to the *given*.
glob_file(commit, pattern)	Lists files that match a glob pattern.
health()	Returns a health check indicating if the server can handle RPCs.

continues on next page

Table 15 – continued from previous page

inspect_branch(repo_name, branch_name)	Inspects a branch.
inspect_cluster()	Inspects a cluster.
inspect_commit(commit[, block_state])	Inspects a commit.
inspect_datum(job_id, datum_id)	Inspects a datum.
inspect_file(commit, path)	Inspects a file.
inspect_job(job_id[, block_state, ...])	Inspects a job with a given ID.
inspect_pipeline(pipeline_name[, history])	
inspect_repo(repo_name)	Returns info about a specific repo.
inspect_secret(secret_name)	Inspects a secret.
inspect_transaction(transaction)	Inspects a given transaction.
list_branch(repo_name[, reverse])	Lists the active branch objects on a repo.
list_commit(repo_name[, to_commit, ...])	Lists commits.
list_datum([job_id, page_size, page, input, ...])	Lists datums.
list_file(commit, path[, history, ...])	
list_job([pipeline_name, input_commit, ...])	
list_pipeline([history, allow_incomplete, ...])	
list_repo()	Returns info about all repos, as a list of RepoInfo objects.
list_secret()	Lists secrets.
list_transaction()	Lists transactions.
modify_admins([add, remove])	Adds and/or removes admins.
modify_cluster_role_binding(principal[, roles])	Sets the list of admin roles for a principal.
modify_members(group[, add, remove])	Adds and/or removes members of a group.
new_from_config([config_file])	Creates a Pachyderm client from a config file, which can either be passed in as a file-like object, or if unset, checks the PACH_CONFIG env var for a path.
new_from_pachd_address(pachd_address[, ...])	Creates a Pachyderm client from a given pachd address.
new_in_cluster([auth_token, transaction_id])	Creates a Pachyderm client that operates within a Pachyderm cluster.
profile_cpu(duration[, filter])	Gets a CPU profile.
put_file_bytes(commit, path, value[, ...])	Uploads a PFS file from a file-like object, bytestring, or iterator of bytestrings.
put_file_client()	A context manager that gives a *PutFileClient*.
put_file_url(commit, path, url[, delimiter, ...])	Puts a file using the content found at a URL.
renew_tmp_file_set(fileset_id, ttl_seconds)	Renews a temporary fileset (used internally).
restart_datum(job_id[, data_filters])	Restarts a datum.
restore(requests)	Restores a cluster.
restore_auth_token([token])	This maps to an internal function that is only used for migration.
revoke_auth_token(token)	Revokes an auth token.
run_cron(pipeline_name)	Explicitly triggers a pipeline with one or more cron inputs to run now.
run_pipeline(pipeline_name[, provenance, job_id])	Runs a pipeline.
set_acl(repo, entries)	Sets the ACL of a repo.

continues on next page

Table 15 – continued from previous page

set_auth_configuration(configuration)	Set the auth configuration.
set_groups_for_user(username, groups)	Sets the group membership for a user.
set_scope(username, repo, scope)	Set the auth scope.
start_commit(repo_name[, branch, parent, ...])	Begins the process of committing data to a Repo.
start_pipeline(pipeline_name)	Starts a pipeline.
start_transaction()	Starts a transaction.
stop_job(job_id)	Stops a job by its ID.
stop_pipeline(pipeline_name)	Stops a pipeline.
subscribe_commit(repo_name, branch[, ...])	Yields CommitInfo objects as commits occur.
transaction()	A context manager for running operations within a transaction.
walk_file(commit, path)	Walks over all descendant files in a directory.
who_am_i()	Returns info about the user tied to this *Client*.

__init__(*host=None*, *port=None*, *auth_token=None*, *root_certs=None*, *transaction_id=None*, *tls=None*)

Creates a Pachyderm client.

Parameters

host [str, optional] The pachd host. Default is 'localhost', which is used with `pachctl port-forward`.

port [int, optional] The port to connect to. Default is 30650.

auth_token [str, optional] The authentication token. Used if authentication is enabled on the cluster.

root_certs [bytes, optional] The PEM-encoded root certificates as byte string.

transaction_id [str, optional] The ID of the transaction to run operations on.

tls [bool, optional] Whether TLS should be used. If *root_certs* are specified, they are used. Otherwise, we use the certs provided by certifi.

property auth_token

classmethod new_from_config(*config_file=None*)

Creates a Pachyderm client from a config file, which can either be passed in as a file-like object, or if unset, checks the PACH_CONFIG env var for a path. If that's also unset, it defaults to loading from '~/.pachyderm/config.json'.

Parameters

config_file [TextIO, optional] A file-like object containing the config json file. If unspecified, we load the config from the default location ('~/.pachyderm/config.json').

Returns

Client A python_pachyderm client instance.

classmethod new_from_pachd_address(*pachd_address*, *auth_token=None*, *root_certs=None*, *transaction_id=None*)

Creates a Pachyderm client from a given pachd address.

Parameters

pachd_address [str] The address of pachd server

auth_token [str, optional] The authentication token. Used if authentication is enabled on the cluster.

root_certs [bytes, optional] The PEM-encoded root certificates as byte string. If unspecified, this will load default certs from certifi.

transaction_id [str, optional] The ID of the transaction to run operations on.

Returns

Client A python_pachyderm client instance.

classmethod new_in_cluster(*auth_token=None*, *transaction_id=None*)
Creates a Pachyderm client that operates within a Pachyderm cluster.

Parameters

auth_token [str, optional] The authentication token. Used if authentication is enabled on the cluster.

transaction_id [str, optional] The ID of the transaction to run operations on.

Returns

Client A python_pachyderm client instance.

property transaction_id

1.1.3 Spout

class python_pachyderm.spout.**SpoutCommit**(*pipe*, *marker_filename=None*)
Represents a commit on a spout, permitting the addition of files.

Methods

close()	Closes the commit
put_file_from_bytes(path, bytes)	Adds a file to the spout from a bytestring.
put_file_from_fileobj(path, size, fileobj)	Adds a file to the spout from a file-like object.
put_marker_from_bytes(bytes)	Adds to the marker from a bytestring.
put_marker_from_fileobj(size, fileobj)	Writes to the marker file from a file-like object.

__init__(*pipe*, *marker_filename=None*)

close()
Closes the commit

put_file_from_bytes(*path*, *bytes*)
Adds a file to the spout from a bytestring.

Parameters

path [str] The path to the file in the spout.

bytes [bytes] The bytestring representing the file contents.

put_file_from_fileobj(*path*, *size*, *fileobj*)
Adds a file to the spout from a file-like object.

Parameters

path [str] The path to the file in the spout.

size [int] The size of the file.

fileobj [BinaryIO] The file-like object to add.

put_marker_from_bytes(*bytes*)
Adds to the marker from a bytestring.

Parameters

bytes [bytes] The bytestring representing the file contents.

put_marker_from_fileobj(*size*, *fileobj*)
Writes to the marker file from a file-like object.

Parameters

size [int] The size of the file.

fileobj [BinaryIO] The file-like object to add.

class python_pachyderm.spout.**SpoutManager**(*marker_filename=None*, *pfs_directory='/pfs'*)
A convenience context manager for creating spouts.

Examples

```
>>> spout = SpoutManager()
>>> while True:
>>>     with spout.commit() as commit:
>>>         commit.put_file_from_bytes("foo", b"#")
>>>     time.sleep(1.0)
```

Methods

close()	Closes the *SpoutManager*
commit()	Opens a commit on the spout.
marker()	Gets the marker file as a context manager.

__init__(*marker_filename=None*, *pfs_directory='/pfs'*)
Creates a new spout manager.

Parameters

marker_filename [str, optional] The name of the file for storing markers. If unspecified, marker-related operations will fail.

pfs_directory [str, optional] The directory for PFS content. Usually this shouldn't be explicitly specified, unless the spout manager is being tested outside of a real Pachyderm pipeline.

close()
Closes the *SpoutManager*

commit()
Opens a commit on the spout. When the context manager exits, any added files will be committed.

marker()
Gets the marker file as a context manager.

1.1.4 Util Helper

python_pachyderm.util.**create_python_pipeline**(*client*, *path*, *input=None*, *pipeline_name=None*, *image_pull_secrets=None*, *debug=None*, *env=None*, *secrets=None*, *image=None*, *update=False*, ***pipeline_kwargs*)

Utility function for creating (or updating) a pipeline specially built for executing python code that is stored locally at *path*.

A normal pipeline creation process requires you to first build and push a container image with the source and dependencies baked in. As an alternative process, this function circumvents container image creation by using build step-enabled pipelines. See the pachyderm core docs for more info.

If *path* references a directory, it should have:

- A `main.py`, as the pipeline entry-point.
- An optional `requirements.txt` that specifies pip requirements.

Parameters

client [Client] The *Client* instance to use.

path [str] The directory containing the python pipeline source, or an individual python file.

input [Input protobuf, optional] An `Input` object specifying the pipeline input.

pipeline_name [str, optional] A string specifying the pipeline name. Defaults to using the last directory name in *path*.

image_pull_secrets [List[str], optional] A list of strings specifying the pipeline transform's image pull secrets, which are used for pulling images from a private registry. Defaults to *None*, in which case the public docker registry will be used. See the pipeline spec document for more details.

debug [bool, optional] Specifies whether debug logging should be enabled for the pipeline. Defaults to *False*.

env [Dict[str, str], optional] A mapping of string keys to string values specifying custom environment variables.

secrets [List[Secret protobufs], optional] A list of *Secret* objects for secret environment variables.

image [str, optional] A string specifying the docker image to use for the pipeline. Defaults to using pachyderm's official python language builder.

update [bool, optional] Whether to act as an upsert.

****pipeline_kwargs** [dict] Keyword arguments to forward to *create_pipeline*.

python_pachyderm.util.**parse_dict_pipeline_spec**(*d*)

Parses a dict of serialized JSON into a *CreatePipelineRequest* protobuf.

python_pachyderm.util.**parse_json_pipeline_spec**(*j*)

Parses a string of JSON into a *CreatePipelineRequest* protobuf.

python_pachyderm.util.**put_files**(*client*, *source_path*, *commit*, *dest_path*, ***kwargs*)

Utility function for inserting files from the local *source_path* to Pachyderm. Roughly equivalent to `pachctl put file [-r]`.

Parameters

client [Client] The *Client* instance to use.

source_path [str] The file/directory to recursively insert content from.

commit [Union[tuple, str, Commit protobuf]] The `Commit` object to use for inserting files.

dest_path [str] The destination path in PFS.

****kwargs** [dict] Keyword arguments to forward. See `PutFileClient.put_file_from_fileobj()` for details.

CHAPTER

TWO

LINKS

- python_pachyderm repo
- pachyderm repo

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

R

S

T

V

W