spurplus Documentation

Release 2.2.0

Marko Ristin

Jun 22, 2021

CONTENTS:

CHAPTER

ONE

SPURPLUS

Manage remote machines and perform file operations over SSH.

class spurplus.**Delete**(*value*)
Enumerate delete strategies when syncing.

class spurplus.**DirectoryDiff**
Represent the difference between a local and a remote directory.

All paths are given as relative. **L** designates the local machine, **R** designates the remote machine.

Variables

- **local_only_files** – files which exist on **L**, but are missing on **R**
- **identical_files** – files which are the same on **L** and **R**
- **differing_files** – files which differ between **L** and **R**
- **remote_only_files** – files which exist on **R**, but are missing on **L**
- **local_only_directories** – directories that exist only on **L**, but are missing on **R**
- **common_directories** – directories that exist both on **L** and on **R**
- **remote_only_directories** – directories that exist on **R**, but are missing on **L**

class spurplus.**SshShell**(*spur_ssh_shell*, *sftp*, *close_spur_shell=True*, *close_sftp=True*)
Wrap a spur.SshShell instance.

This wrapper adds typing and support for pathlib.Path and facilitates common tasks such as md5 sum computation and file operations.

Variables

- **hostname** (*str*) – host name of the machine
- **port** (*int*) – port of the SSH connection

as_sftp()
Get the underlying SFTP client.

Use that client if you need fine-grained SFTP functionality not available in this class.

Return type Union[SFTP, *ReconnectingSFTP*]

Returns underlying SFTP client

as_spur()
Get the underlying spur shell instance.

Use that instance if you need undocumented spur functionality.

Return type SshShell

Returns underlying spur shell

check_output(*command*, *update_env=None*, *cwd=None*, *stderr=None*, *encoding='utf-8'*, *use_pty=False*)

Run a command on the remote instance that is not allowed to fail and captures its output.

See run() for further documentation.

Return type str

Returns the captured output

chmod(*remote_path*, *mode*)

Change the permission mode of the file.

Parameters

- **remote_path** (Union[str, Path]) – to the file
- **mode** (int) – permission mode

Return type None

Returns

chown(*remote_path*, *uid*, *gid*)

Change the ownership of the file.

If you only want to change the uid or gid, please stat() the file before, and re-apply the current uid or gid, respectively.

Parameters

- **remote_path** (Union[str, Path]) – to the file
- **uid** (int) – ID of the user that owns the file
- **gid** (int) – ID of the group that owns the file

Return type None

Returns

close()

Close the underlying spur shell and SFTP (if close_spur_shell and close_sftp, respectively).

Return type None

directory_diff(*local_path*, *remote_path*)

Iterate through the local and the remote directory and computes the diff.

If one of the directories does not exist, all files are assumed "missing" in that directory.

The identity of the files is based on MD5 checksums.

Parameters

- **local_path** (Union[str, Path]) – path to the local directory
- **remote_path** (Union[str, Path]) – path to the remote directory

Return type *DirectoryDiff*

Returns difference between the directories

exists(*remote_path*)

Check whether a file exists.

Parameters **remote_path** (Union[str, Path]) – to the file

Return type bool

Returns True if the file exists on the remote machine at *remote_path*

get(*remote_path*, *local_path*, *create_directories=True*, *consistent=True*)
Get a file from the remote host.

Parameters

- **remote_path** (Union[str, Path]) – to the file
- **local_path** (Union[str, Path]) – to the file
- **create_directories** (bool) – if set, creates the parent directories of the local path with permission mode 0o777
- **consistent** (bool) – if set, copies to a temporary local file first, and then renames it.

Return type None

Returns

is_dir(*remote_path*)
Check whether the remote path is a directory.

Parameters **remote_path** (Union[str, Path]) – path to the remote file or directory

Return type bool

Returns True if the remote path is a directory

Raise FileNotFound if the remote path does not exist

is_symlink(*remote_path*)
Check whether the remote path is a symlink.

Parameters **remote_path** (Union[str, Path]) – path to the remote file or directory

Return type bool

Returns True if the remote path is a directory

Raise FileNotFound if the remote path does not exist

md5(*remote_path*)
Compute MD5 checksum of the remote file. It is assumed that md5sum command is available on the remote machine.

Parameters **remote_path** (Union[str, Path]) – to the file

Return type str

Returns MD5 sum

md5s(*remote_paths*)
Compute MD5 checksums of multiple remote files individually.

It is assumed that md5sum command is available on the remote machine.

Parameters **remote_paths** (Sequence[Union[str, Path]]) – to the files

Return type List[Optional[str]]

Returns MD5 sum for each remote file separately; if a file does not exist, its checksum is set to None.

mirror_local_permissions(*relative_paths*, *local_path*, *remote_path*)
Set the permissions of the remote files to be the same as the permissions of the local files.

The files are given as relative paths and are expected to exist both beneath `local_path` and beneath `remote_path`.

Parameters

- **relative_paths** (Sequence[Union[str, Path]]) – relative paths of files whose permissions are changed
- **local_path** (Union[str, Path]) – path to the local directory
- **remote_path** (Union[str, Path]) – path to the remote directory

Return type None

Returns

mkdir(*remote_path*, *mode=511*, *parents=False*, *exist_ok=False*)
Create the remote directory.

Parameters

- **remote_path** (Union[str, Path]) – to the directory
- **mode** (int) – directory permission mode
- **parents** (bool) – if set, creates the parent directories
- **exist_ok** (bool) – if set, ignores an existing directory.

Return type None

Returns

put(*local_path*, *remote_path*, *create_directories=True*, *consistent=True*)
Put a file on the remote host.

Mind that if you set consistent to True, the file will be copied to a temporary file and then POSIX rename function will be used to rename it. The ownership and the permissions of the original 'remote_path' are preserved. However, if the original 'remote_path' has read-only permissions and you still have write permissions to the directory, the 'remote_path' will be overwritten nevertheless due to the logic of POSIX rename.

Parameters

- **local_path** (Union[str, Path]) – to the file
- **remote_path** (Union[str, Path]) – to the file
- **create_directories** (bool) – if set, creates the parent directory of the remote path with mode 0o777
- **consistent** (bool) – if set, copies to a temporary remote file first, and then renames it.

Return type None

Returns

read_bytes(*remote_path*)
Read the binary data from a remote file.

First the remote file is copied to a temporary local file making sure that the connection is reestablished if needed. Next the data is read.

Parameters **remote_path** (Union[str, Path]) – to the file

Return type bytes

Returns binary content of the file

read_text(*remote_path*, *encoding='utf-8'*)
Read the text content of a remote file.

Parameters

- **remote_path** (Union[str, Path]) – to the file
- **encoding** (str) – of the text file

Return type str

Returns binary content of the file

remove(*remote_path*, *recursive=False*)
Remove a file or a directory.

Parameters

- **remote_path** (Union[str, Path]) – to a file or a directory
- **recursive** (bool) – if set, removes the directory recursively. This parameter has no effect if remote_path is not a directory.

Return type None

Returns

run(*command*, *cwd=None*, *update_env=None*, *allow_error=False*, *stdout=None*, *stderr=None*, *encoding='utf-8'*, *use_pty=False*)
Run a command on the remote instance and waits for it to complete.

From https://github.com/mwilliamson/spur.py/blob/0.3.20/README.rst:

Parameters

- **command** (Sequence[str]) – to be executed
- **cwd** (Union[str, Path, None]) – change the current directory to this value before executing the command.
- **update_env** (Optional[Mapping[str, str]]) – environment variables to be set before running the command.

  If there's an existing environment variable with the same name, it will be overwritten. Otherwise, it is unchanged.
- **allow_error** (bool) – If False, an exception is raised if the return code of the command is anything but 0. If True, a result is returned irrespective of return code.
- **stdout** (Optional[TextIO]) – if not None, anything the command prints to standard output during its execution will also be written to stdout using stdout.write.
- **stderr** (Optional[TextIO]) – if not None, anything the command prints to standard error during its execution will also be written to stderr using stderr.write.
- **encoding** (str) – if set, this is used to decode any output. By default, any output is treated as raw bytes. If set, the raw bytes are decoded before writing to the passed stdout and stderr arguments (if set) and before setting the output attributes on the result.
- **use_pty** (bool) – (undocumented in spur 0.3.20) If set, requests a pseudo-terminal from the server.

Return type ExecutionResult

Returns execution result

Raise spur.results.RunProcessError on an error if allow_error=False

spawn(*command*, *update_env=None*, *store_pid=False*, *cwd=None*, *stdout=None*, *stderr=None*, *encoding='utf-8'*, *use_pty=False*, *allow_error=False*)

Spawn a remote process.

From https://github.com/mwilliamson/spur.py/blob/0.3.20/README.rst:

Parameters

- **command** (Sequence[str]) – to be executed
- **cwd** (Union[str, Path, None]) – change the current directory to this value before executing the command.
- **update_env** (Optional[Mapping[str, str]]) – environment variables to be set before running the command.

  If there's an existing environment variable with the same name, it will be overwritten. Otherwise, it is unchanged.
- **store_pid** (bool) – If set to True, store the process id of the spawned process as the attribute pid on the returned process object.
- **allow_error** (bool) – If False, an exception is raised if the return code of the command is anything but 0. If True, a result is returned irrespective of return code.
- **stdout** (Optional[TextIO]) – If not None, anything the command prints to standard output during its execution will also be written to stdout using stdout.write.
- **stderr** (Optional[TextIO]) – If not None, anything the command prints to standard error during its execution will also be written to stderr using stderr.write.
- **encoding** (str) – If set, this is used to decode any output. By default, any output is treated as raw bytes. If set, the raw bytes are decoded before writing to the passed stdout and stderr arguments (if set) and before setting the output attributes on the result.
- **use_pty** (bool) – (undocumented in spur 0.3.20) If set, requests a pseudo-terminal from the server.

Return type SshProcess

Returns spawned process

Raise spur.results.RunProcessError on an error if allow_error=False

stat(*remote_path*)

Stat the given remote path.

Parameters **remote_path** (Union[str, Path]) – to the file

Return type Optional[SFTPAttributes]

Returns stats of the file; None if the file does not exist

symlink(*source*, *destination*)

Create a symbolic link to the source remote path at destination.

Parameters

- **source** (Union[str, Path]) – remote path to the source
- **destination** (Union[str, Path]) – remote path where to store the symbolic link

Return type None

Returns

sync_to_remote(*local_path*, *remote_path*, *consistent=True*, *delete=None*, *preserve_permissions=False*)

Sync all the files beneath the `local_path` to `remote_path`.

Both local path and remote path are directories. If the `remote_path` does not exist, it is created. The files are compared with MD5 first and only the files whose MD5s mismatch are copied.

Mind that the directory lists and the mapping (path -> MD5) needs to fit in memory for both the local path and the remote path.

Parameters

- **local_path** (Union[str, Path]) – path to the local directory
- **remote_path** (Union[str, Path]) – path to the remote directory
- **consistent** (bool) – if set, writes to a temporary remote file first on each copy, and then renames it.
- **delete** (Optional[*Delete*]) – if set, files and directories missing in `local_path` and existing in `remote_path` are deleted.
- **preserve_permissions** (bool) – if set, the remote files and directories are chmod'ed to reflect the local files and directories, respectively.

Return type None

Returns

whoami()

Execute the *whoami* command and return the user name.

Return type str

write_bytes(*remote_path*, *data*, *create_directories=True*, *consistent=True*)

Write the binary data to a remote file.

First, the data is written to a temporary local file. Next the local file is transferred to the remote path making sure that the connection is reestablished if needed.

Parameters

- **remote_path** (Union[str, Path]) – to the file
- **data** (bytes) – to be written
- **create_directories** (bool) – if set, creates the parent directory of the remote path with mode 0o777
- **consistent** (bool) – if set, writes to a temporary remote file first, and then renames it.

Return type None

Returns

write_text(*remote_path*, *text*, *encoding='utf-8'*, *create_directories=True*, *consistent=True*)

Write the binary content to the remote host.

Parameters

- **remote_path** (Union[str, Path]) – to the file
- **text** (str) – to be written

- **encoding** (str) – to encode the text
- **create_directories** (bool) – if set, creates the parent directory of the remote path with mode 0o777
- **consistent** (bool) – if set, writes to a temporary remote file first, and then renames it.

Return type None

Returns

class spurplus.**TemporaryDirectory**(*shell*, *prefix=None*, *suffix=None*, *tmpdir=None*)

Represent a remote temporary directory.

spurplus.**chunk_arguments**(*args*, *arg_max=16384*, *argc_max=1024*)

Split a long list of command-line arguments into chunks.

This is needed in order not to overflow the maximum length of the command-line arguments.

Parameters

- **args** (Sequence[str]) – command-line arguments
- **arg_max** (int) – maximum length of the command-line arguments (i.e. the result of `getconf ARG_MAX`)
- **argc_max** – maximum number of command-line arguments

Return type List[List[str]]

Returns chunked command-line arguments

spurplus.**connect_with_retries**(*hostname*, *username=None*, *password=None*, *port=None*, *private_key_file=None*, *connect_timeout=None*, *missing_host_key=None*, *shell_type=None*, *look_for_private_keys=True*, *load_system_host_keys=True*, *sock=None*, *retries=12*, *retry_period=5*)

Try to connect to the instance and retry on failure.

Reconnect *retries* number of times and wait for *retry_period* seconds between the retries.

For all the arguments except *retries* and *retry_period*, the documentation was copy/pasted from https://github.com/mwilliamson/spur.py/blob/0.3.20/README.rst:

You need to specify some combination of a username, password and private key to authenticate.

Parameters

- **hostname** (str) – of the instance to connect
- **username** (Optional[str]) – for authentication
- **password** (Optional[str]) – for authentication
- **port** (Optional[int]) – for connection, default is 22
- **private_key_file** (Union[str, Path, None]) – path to the private key file
- **connect_timeout** (Optional[int]) – a timeout in seconds for establishing an SSH connection. Defaults to 60 (one minute).
- **missing_host_key** (Optional[MissingHostKey]) – by default, an error is raised when a host key is missing.

  One of the following values can be used to change the behaviour when a host key is missing: * spur.ssh.MissingHostKey.raise_error – raise an error * spur.ssh.MissingHostKey.warn –

accept the host key and log a warning * spur.ssh.MissingHostKey.accept – accept the host key

- **shell_type** (Optional[ShShellType]) – the type of shell used by the host. Defaults to spur.ssh.ShellTypes.sh, which should be appropriate for most Linux distributions. If the host uses a different shell, such as simpler shells often found on embedded systems, try changing shell_type to a more appropriate value, such as spur.ssh.ShellTypes.minimal. The following shell types are currently supported:
  - spur.ssh.ShellTypes.sh – the Bourne shell. Supports all features.
  - spur.ssh.ShellTypes.minimal – a minimal shell. Several features are unsupported:
  - Non-existent commands will not raise spur.NoSuchCommandError.
  - The following arguments to spawn and run are unsupported unless set to their default values: cwd, update_env, and store_pid.
- **look_for_private_keys** (Optional[bool]) – by default, Spur will search for discoverable private key files in ~/.ssh/. Set to False to disable this behaviour.
- **load_system_host_keys** (Optional[bool]) – by default, Spur will attempt to read host keys from the user's known hosts file, as used by OpenSSH, and no exception will be raised if the file can't be read. Set to False to disable this behaviour.
- **sock** (Optional[socket]) – an open socket or socket-like object to use for communication to the target host.
- **retries** (int) – (spurplus) number of re-tries if the connection could not be established
- **retry_period** (int) – (spurplus) how many seconds to wait between the retries

Return type *SshShell*

Returns established SshShell

CHAPTER

TWO

SPURPLUS.SFTP

Wrap paramiko.SFTP.

class spurplus.sftp.**ReconnectingSFTP**(*sftp_opener*, *max_retries=10*, *retry_period=0.1*)

Open automatically a new paramiko.SFTP on connection failure.

chmod(*path*, *mode*)

See paramiko.SFTP documentation.

chown(*path*, *uid*, *gid*)

See paramiko.SFTP documentation.

close()

Close the the underlying paramiko SFTP client.

Return type None

get(*remotepath*, *localpath*, *callback=None*)

See paramiko.SFTP documentation.

listdir(*path='.'*)

See paramiko.SFTP documentation.

listdir_attr(*path='.'*)

See paramiko.SFTP documentation.

lstat(*path*)

See paramiko.SFTP documentation.

mkdir(*path*, *mode=511*)

See paramiko.SFTP documentation.

posix_rename(*oldpath*, *newpath*)

See paramiko.SFTP documentation.

put(*localpath*, *remotepath*, *callback=None*, *confirm=True*)

See paramiko.SFTP documentation.

remove(*path*)

See paramiko.SFTP documentation.

rmdir(*path*)

See paramiko.SFTP documentation.

stat(*path*)

See paramiko.SFTP documentation.

symlink(*source*, *dest*)

See paramiko.SFTP documentation.

unlink(*path*)
See paramiko.SFTP documentation.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

U

W