SyncHg Documentation

Release 1.0.0

Graeme Coupar

March 28, 2013

CONTENTS

Ever had to keep two mercurial repositories on different machines in sync? Mercurials push & pull help to make this fairly easy, but if you make use of mercurial queues or the histedit extension then it can quickly become tedious. That's where synchg comes in.

Synchg intends to make syncing two mercurial repositories as simple as possible. Simply run a command, and synchg will take care of the rest.

Python 2.7 & Mercurial 2.3 are recommended, though others will probably work.

Synchg depends on these python packages:

- Plumbum
- Clint

It also requires:

- An ssh client on the path (putty on windows, openssh compatible on other platforms)
- Access to an SSH server on the remote machine(s)
- An ssh private key loaded in an ssh agent (pagaent on windows, ssh-agent on other platforms)
- That the mq extension is enabled on the remote machine(s)

CHAPTER

ONE

INSTALLATION

Synchg and it's python dependencies can be installed via pip:

```
$ pip install synchg
```

CHAPTER

TWO

USING SYNCHG

The synchg script should be run from the command line:

```
$ synchg remote_host [local_path=None]
```

Where `remote_host` is the host you wish to sync with and `local_path` is the optional path to the local mercurial repository (if missing, the current directory will be assumed)

Information on more options can be found by running:

```
$ synchg --help
```

Caution: Synchg regards remote repositories as "slaves" and will strip out any changesets it finds that are not in the local repository. You will be prompted before this happens, but the script will be unable to continue if you don't answer yes.

2.1 Configuration

On first run of synchg you will be prompted with some configuration options:

Remote source directory This is the path on the remote under which all your repositories should be found. For example, if you have repositories at `/repo/one/` and `/repo/two/` then you would set this to `/repo/`

If you want to change the configuration of synchg, then simply run `synchg -c` to run the config process again.

CHAPTER

THREE

SYNCHG API

Synchg also exposes a simple python API that can be used to integrate synchg functionality into other python projects such as build scripts.

The SyncHg API can be used easily, simply by calling the `synchg.sync.SyncRemote()` function. For example:

```
from synchg import SyncRemote

print "Enter the following details:"
host = raw_input('Remote host: ')
repo_name = raw_input('Repository name: ')
local_path = raw_input('Local path: ')
remote_root = raw_input('Remote root: ')

SyncRemote(host, repo_name, local_path, remote_root)
```

3.1 Syncing Utilities (synchg.sync)

This module provides the actual syncing functionality for SyncHg. It's functions can be called by imported and called by other libraries if they wish to make use of SyncHg functionality.

exception `synchg.sync.`**`AbortException`**
An exception that's thrown when a user chooses to abort. This should be caught and ignored at the start of the program to allow users to abort at prompts

exception `synchg.sync.`**`SyncError`**
An exception that's thrown when a non-exceptional error occurs. This exception is usually accompanied by an error message and should probably be caught and the backtrace suppressed.

`synchg.sync.`**`SyncRemote`**(*host*, *name*, *localpath*, *remote_root*)
Syncs a remote repository. This function should be called to kick off a sync

Parameters

- **host** – The hostname of the remote repository
- **name** – The name of the project that is being synced. This parameter will be appended to the remote_root to find the remote repository.
- **localpath** – A plumbum path to the local repository
- **remote_root** – The path to the parent directory of the remote repository

3.2 Repository Control (synchg.repo)

class synchg.repo.**Repo**(*machine*, *remote=None*)

This class provides an abstraction around running commands on a mercurial repository. It can be used against either a local or remote repository depending on the machine parameter to the constructor.

Parameters

- **machine** – The plumbum machine object to use (can be a local machine or remote machine)
- **remote** – The name of the remote repo to be used by push, pull and other operations.

class **ChangesetInfo**

ChangesetInfo(hash, desc)

desc

Alias for field number 1

hash

Alias for field number 0

Repo.**CleanMq**(**args*, ***kwds*)

Returns a context manager that keeps the mq repository clean for it's lifetime

Repo.**Clone**(**pargs*)

Clones the repository to a different location

Parameters

- **destination** – The destination clone path
- **createRemote** – If set a remote will be created in the local hgrc with the name the class was initialised with.

Repo.**CloneMq**(*destination*, *createRemote=True*)

Clones the mq repository to a different location

Parameters

- **destination** – The destination path to the top-level remote repository. NOT the remote mq repository
- **createRemote** – If set, a remote will be created in the local mq hgrc with the name the class was initialised with

class Repo.**CommitChangeInfo**

CommitChangeInfo(modified, unknown)

modified

Alias for field number 0

unknown

Alias for field number 1

Repo.**CommitMq**(*msg=None*)

Commits the mq repository

Parameters **msg** – An optional commit message

Repo.**InitMq**()

Initialises the mq repository

class Repo.**MqAppliedInfo**

MqAppliedInfo(applied, unapplied)

applied
Alias for field number 0

unapplied
Alias for field number 1

Repo.**PopPatch**(*patch=None*)
Pops mq patch(es)

Parameters **patch** – Name of the patch to pop to. If None, all will be popped

Repo.**PushMqToRemote**()
Pushes the mq repo to the remote at *self.remote*

Repo.**PushPatch**(*patch=None*)
Pushes mq patch(es)

Parameters **patch** – Name of the patch to push to. If None, all will be pushed

Repo.**PushToRemote**(**pargs*)
Pushes to the remote repository at *self.remote*

Repo.**RefreshMq**()
Refreshes the current mq patch

Repo.**Strip**(**pargs*)
Strips changesets from this repository

Parameters **changesets** – A list of `ChangesetInfo` representing the changesets to strip

class Repo.**SummaryInfo**
SummaryInfo(commit, mq)

commit
Alias for field number 0

mq
Alias for field number 1

Repo.**Update**(**pargs*)
Updates to a specific changeset

Parameters **changeset** – A changeset hash string, or `ChangesetInfo` representing the changeset to update to

Repo.**UpdateMq**()
Updates the mq repository to tip

Repo.**branch**
Gets the current branch This property is cached, so it may be out of date

Returns A string containing the current branch name

Repo.**config**
Gets the configuration for this repository

Returns A `RepoConfig` class

Repo.**currentRev**
Gets the current revision This property is cached, so it may be out of date

Returns A string containing the current revision hash

Repo.**incomings**
Gets the incoming changesets from *self.remote*

Returns A list containing `ChangesetInfo` that represent the current incoming changesets

`Repo.`**`lastAppliedPatch`**
Gets the last applied mq patch (if there is one)

Returns A single mq patch name (or None)

`Repo.`**`mqconfig`**
Gets the configuration for the mq repository

:returns A `RepoConfig` class

`Repo.`**`outgoings`**
Gets the outgoing changesets to *self.remote*

Returns A list containing `ChangesetInfo` that represent the current outgoing changesets

`Repo.`**`summary`**
Gets info from hg summary

Returns A `SummaryInfo` containing `CommitChangeInfo` & `MqAppliedInfo`

PYTHON MODULE INDEX

S