

---

# **SyncHg Documentation**

*Release 0.9.5*

**Graeme Coupar**

November 01, 2012



# 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:

- Access to an SSH server on the remote machine(s)
- That the [mq](#) extension is enabled on the remote machine(s)



# INSTALLATION

Synchg and its python dependencies can be installed via pip:

```
$ pip install synchg
```



# USING SYNCHG

Before using `synchg` on a repository you should ensure that your environment is set up correctly. If you intend to use `mq` patches with `synchg`, then you should run `hg init --mq` on each local repository before you attempt to use it with `synchg`.

It's recommended that you use `synchg` to make the initial clone to your remote machine. This way it can take steps to add necessary settings to the local repository. However, if you wish to use `synchg` with an existing clone of your repository, then read the section below entitled [Using With Existing Clones](#).

## 2.1 Running The Script

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.

## 2.2 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.

## 2.3 Using With Existing Clones

Though it's recommended that you allow synchg to perform the initial clone of a repository, it is possible to use it with existing clones. You simply need to make sure that the remote repository is listed as a remote in the `.hgrc` for your local repository. The remote should be named using the hostname of the remote machine.

If you intend to use mq patches, this will also need to be done with the mq repository.

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

## 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, \*\*kws)

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
- **remoteName** – If set a remote will be created with this name

**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

**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.**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.**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

`synchg.repo`, ??  
`synchg.sync`, ??