SCM Backup Documentation

Release 1.4.0

Christian Specht

May 08, 2020

Contents:

SCM Backup is a tool which makes offline backups of your cloud hosted source code repositories, by cloning them.

- Source code
- Website
- Download latest release

It's written in .NET Core, which means it's supposed to run on Windows, Linux and MacOS.

CHAPTER 1

Introduction

SCM Backup is a tool which makes offline backups of your cloud hosted source code repositories, by cloning them.

It's free and open source!

It supports backing up from multiple source code hosters and backing up multiple users/teams per source code hoster.

At the moment, the following hosters are supported:

- Bitbucket
- GitHub
- GitLab

1.1 How does it work?

SCM Backup uses the respective hoster's API to get a list of all your repositories hosted there.

Then, it uses the respective SCM (e.g. Git and/or Mercurial, which need to be installed on your machine if you have at least one repository of the given type) to clone every repository into your local backup folder - or just pull the newest changes, if it already **is** in your local backup folder.

Many hosters support wikis per repository, which are separate repositories and will be backed up as well.

CHAPTER 2

Installation

2.1 System Requirements

2.1.1 .NET Core

SCM Backup is written in .NET Core, the cross-platform version of .NET.

The available releases are framework-dependent deployments, which means that the same download should work on any Windows, Linux and MacOS machine, as long as .NET Core is installed on it.

If it's not on your machine, you can get it from the official download page. The exact version you need is **version 3.1** of the .NET Core **runtime**.

Note: So far, SCM Backup has been written and tested on Windows only. Technically, it should run on Linux and MacOS as well, but this has not been tested yet.

2.1.2 Source control software

SCM Backup doesn't come with its own versions of Git and/or Mercurial, so the respective SCM needs to be installed on your machine if you have at least one repository of the given type.

By default, SCM Backup expects all source control software to be in your path, so it just needs to execute `git`, `hg` etc. without a complete path, although it's possible to *specify the path to the executable in the config*.

Note that at runtime, SCM Backup checks the presence of all required SCMs on your system. It will stop if you have repositories needing a SCM which is not present on your system.

2.2 Download

At the moment, there are only .zip downloads.

Download the .zip file from the latest release and unzip it into a folder of your choice.

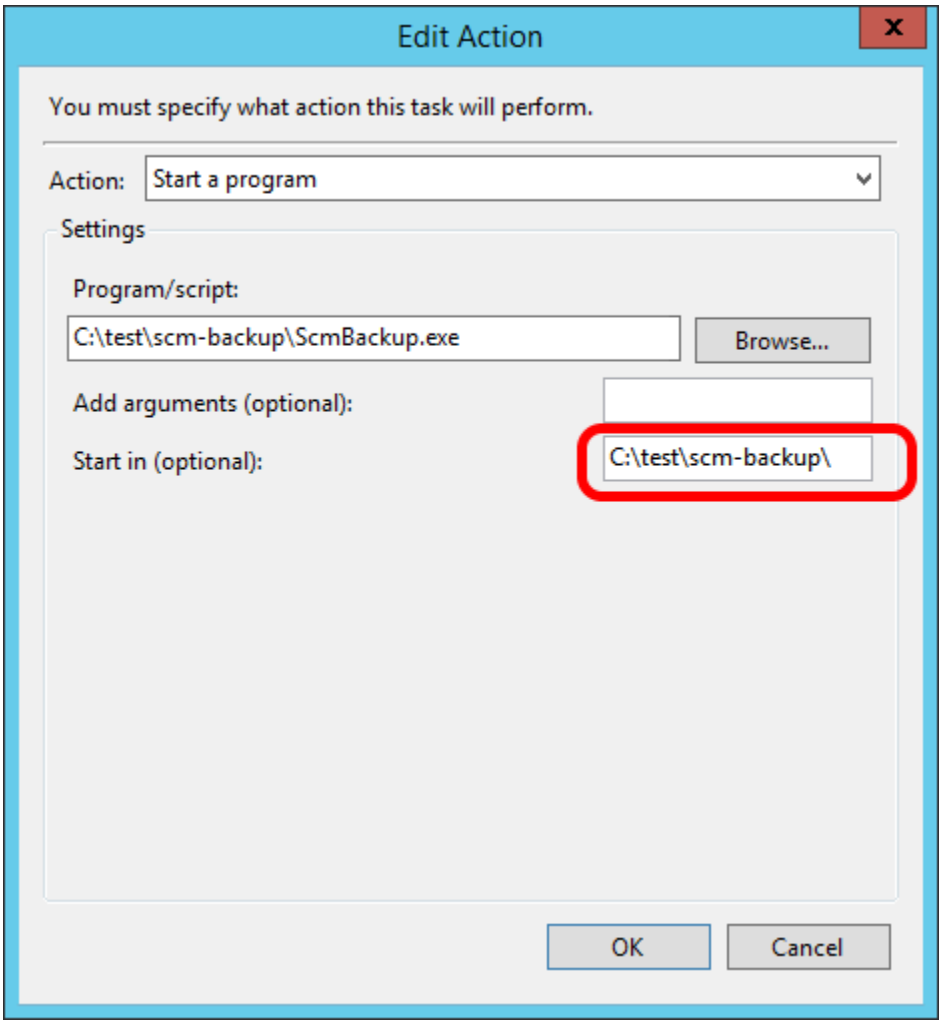
2.3 How to run

> **Warning:** You should edit the configuration file before running SCM Backup for the first time!
>
> *Read the guide* for more information.

Run the actual application by executing `ScmBackup.exe`

2.3.1 Windows Task Scheduler

To run SCM Backup via Windows Task Scheduler, you need to specify the path to the exe **and** the directory (in "Start in") in the "Edit Action" screen:



Omitting the directory can cause problems.

CHAPTER 3

Configuration

SCM Backup is configured in YAML, by editing the config file `settings.yml`.

Note: SCM Backup automatically makes a backup of its own configuration.

On each run, the following files are copied to the *backup folder*, into a subfolder named `_config`:

- `settings.yml`
- The *logger's* config file

3.1 General Options

3.1.1 localFolder

The folder (on the machine where SCM Backup runs) where all the backups will be stored.

The folder must already exist, SCM Backup won't create it.

Example:

```
localFolder: 'c:\scm-backup'
```

3.1.2 waitSecondsOnError

When an error occurs, SCM Backup will wait that many seconds before exiting the application.

Example:

```
waitSecondsOnError: 5
```

3.1.3 scms

SCM Backup uses the source control software already installed on your system. By default, it assumes that the required SCMs are installed in your path.

If this isn't the case, or if you have multiple versions of the same SCM on your system and want SCM Backup to use a specific one, you can specify the complete path to the executable in the config file.

Example:

```
scms:
  - name: git
    path: 'c:\git\git.exe'
```

3.1.4 email

Settings for *sending log information via email*.

By default, the whole section is commented out via `#`. To enable it, remove the comments so it looks like this:

```
email:
  from: from@example.com
  to: to@example.com
  server: smtp.example.com
  port: 0
  useSsl: false
  userName: testuser
  password: not-the-real-password
```

Fill all settings with the proper values for your server.

SCM Backup will try sending emails when an un-commented `email` section exists in the configuration.

3.2 Sources

SCM Backup is able to backup from multiple source code hosters, and multiple accounts per hoster.

For example, your GitHub user may be a member of an organization, and you may want to backup all repositories of your user, **and** all repositories of that organization.

In SCM Backup terms, these would be two different **sources**: your GitHub user would be one source, and the organization would be a second one.

You can define as many sources as you want in the config file, in this format:

```
sources:

  - title: some_title
    hoster: github
    type: user
    name: your_user_name

  - title: another_title
    hoster: github
    type: org
    name: your_org_name
```

Each source must have at least those four properties:

`title`

Must be unique in the whole config file.

For each source, SCM Backup will create a sub-folder named like the source's title in the *main backup folder*.

`hoster`

The source code hoster from which you want to backup. See the sub-pages for valid values for each hoster.

`type`

Either `user` or `org`, depending if you want to backup an user or a organization.

`name`

The name of the user/organization you want to backup.

Warning: With these settings, SCM Backup will backup public repositories only. For private repositories, additional properties must be set see *Authentication*

See the respective sub-page for detailed documentation per hoster:

3.2.1 Bitbucket

Configuration settings for backing up repositories from Bitbucket.

Warning: Known limitations:

- Issues are not backed up
- **SCM Backup supports Bitbucket Cloud (see Hosting Options) only, the self-hosted options are not supported!** see *Why doesn't SCM Backup support backing up local installations of [hoster X]?*

Sources

For the basics, please read the *Sources* section first.

For Bitbucket, the `hoster` entry in the config file needs to look like this:

```
hoster: bitbucket
```

SCM Backup will always backup a Bitbucket Workspace.

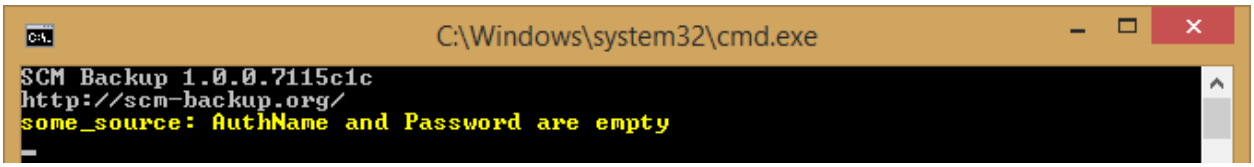
Bitbucket repository URLs look like this: `https://bitbucket.org/WORKSPACEID/REPO/...` so the URL part directly behind `https://bitbucket.org` is the workspace ID.

- To backup the repos of any workspace, the `name` entry in the config needs to be set to the workspace ID
- The `type` property of the source doesn't matter anymore *(because for what SCM Backup does, there's no difference between users and workspaces)*, so SCM Backup will accept either `user` or `org`

Authentication

Without authentication, SCM Backup can only backup your public repositories.

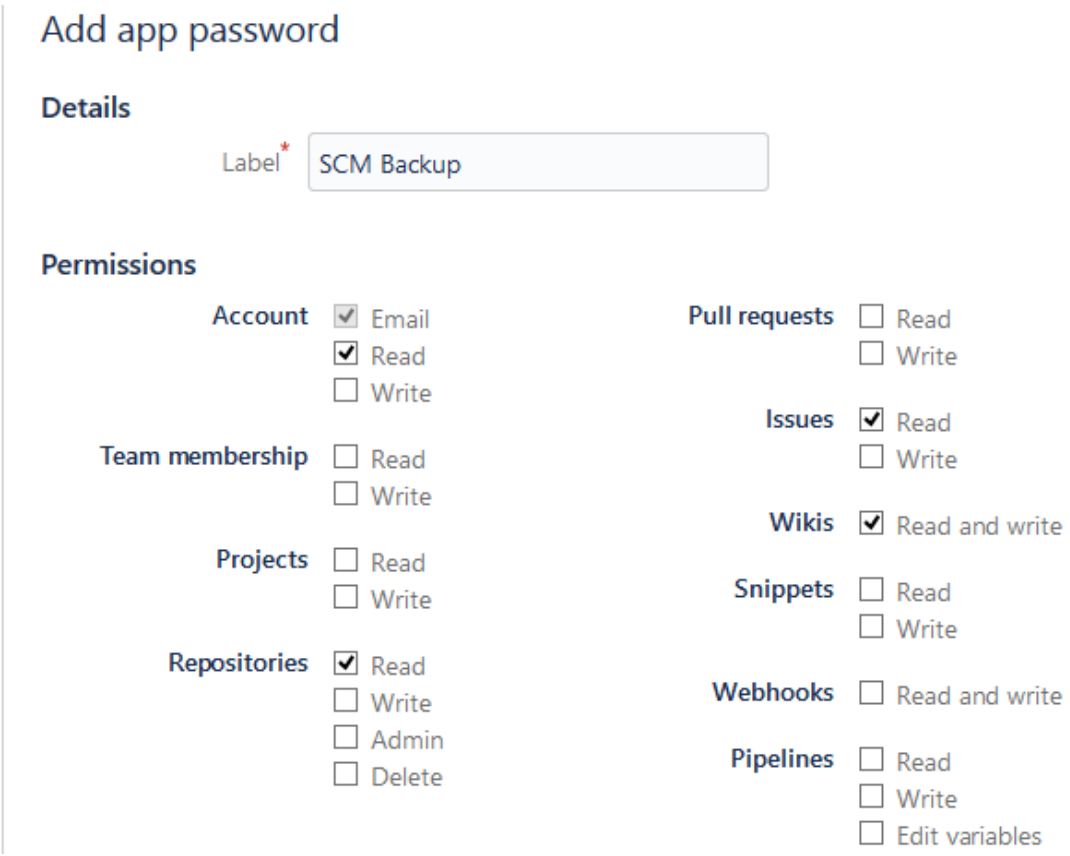
In this case, it shows a warning:



To backup your private repositories as well, you need to authenticate with a user who has sufficient permissions to the workspace's repositories.

Create an app password for SCM Backup for that user:

1. In the user's settings on Bitbucket, go to the **App passwords** area (`https://bitbucket.org/account/user/YOUR-USERNAME/app-passwords`) and create a new app password. Give it at least the following permissions:



   - Account: Read
   - Repositories: Read
   - Issues: Read
   - Wikis: Read and write *(SCM Backup only needs to read, but there's no separate "just read" permission)*

2. Put the username and the app password into the `authName` and `password` properties of the source in the config file.

   Note: By default, a user's workspace ID is equal to the user name, but it's possible to change the workspace ID, so workspace ID and user name don't match anymore.

   In this case, you need to set `name` to the workspace ID and `authName` to the user name.

   Example:

```
sources:

  - title: some_title
    hoster: bitbucket
    type: org
    name: your_workspace
    authName: your_user_name
    password: your_app_password
```

   This will backup the repositories of the workspace `your_workspace`, but authenticate with the user `your_user_name` and the app password.

3.2.2 GitHub

Configuration settings for backing up repositories from GitHub.

Warning: Known limitations:

- Issues are not backed up
- **SCM Backup supports GitHub's cloud hosting plans only, the self-hosting option of GitHub Enterprise is not supported**
  see *Why doesn't SCM Backup support backing up local installations of [hoster X]?*

Sources

For the basics, please read the *Sources* section first.

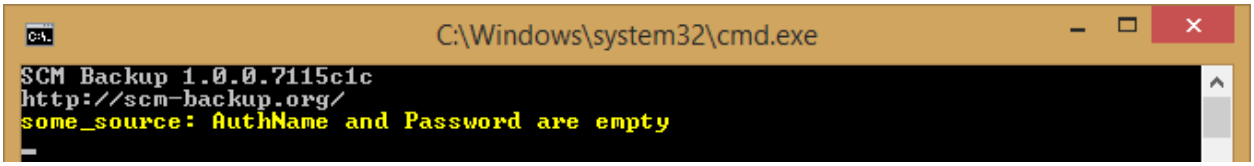
For GitHub, the `hoster` entry in the config file needs to look like this:

```
hoster: github
```

Authentication

Without authentication, SCM Backup can only backup your public repositories.

In this case, it shows a warning:



To backup your private repositories as well, you need to authenticate:

- To backup a user's repositories, you need to authenticate with that user.
- To backup an organization's repositories, you need to authenticate with a user who has sufficient permissions to that organization's repositories.

Create a personal access token for SCM Backup for that user:

1. In the user's settings on GitHub, go to Developer settings Personal access tokens and create a new token. Give it at least the `repo` scope.

   This scope allows SCM Backup to get a list of that user's repositories, including private ones, via the GitHub API (read more about scopes).

2. Put the username and the token into the `authName` and `password` properties of the source in the config file.

   Example:

```
sources:

  - title: some_title
    hoster: github
    type: org
    name: your_org_name
    authName: your_user_name
    password: your_token
```

   This will backup the repositories of the organization `your_org_name`, but authenticate with the user `your_user_name`.

3.2.3 GitLab

Configuration settings for backing up repositories from GitLab.

Warning: Known limitations:

- Issues are not backed up
- **SCM Backup supports GitLab.com (the SaaS option, see Pricing) only, the self-managed options are not supported!** see *Why doesn't SCM Backup support backing up local installations of [hoster X]?*

Sources

For the basics, please read the *Sources* section first.

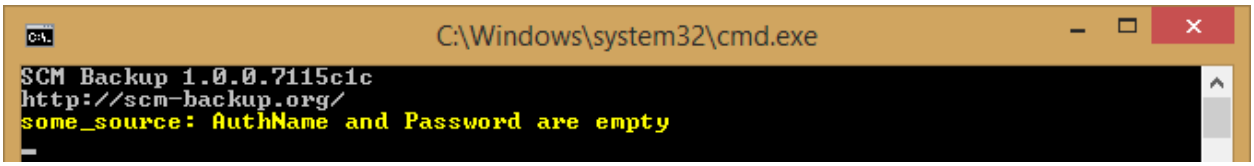
For GitLab, the `hoster` entry in the config file needs to look like this:

```
hoster: gitlab
```

Authentication

Without authentication, SCM Backup can only backup your public repositories.

In this case, it shows a warning:



To backup your private repositories as well, you need to authenticate:

- To backup a user's repositories, you need to authenticate with that user.
- To backup a group's repositories, you need to authenticate with a user who has sufficient permissions to that group's repositories.

Create a personal access token for SCM Backup for that user:

1. In the user's settings on GitLab, go to Settings Access tokens and create a new token with the following scopes:
   - `api`
   - `read_repository`
2. Put the username and the token into the `authName` and `password` properties of the source in the config file.

   Example:

   ```
   sources:

     - title: some_title
       hoster: gitlab
       type: org
       name: your_group_name
       authName: your_user_name
       password: your_token
   ```

   This will backup the repositories of the group `your_group_name`, but authenticate with the user `your_user_name`.

Wikis and rate limits

Unfortunately, the "main" API call doesn't return whether a project has a wiki.

To determine if there's a wiki which needs to be backed up, SCM Backup has to do the following **for each project that the API call returns**:

1. check if the "wiki" feature is activated for the current project (Repo settings Features, default value for new repos: yes)
2. if yes, make a separate call to the Wikis API to check if this wiki has at least one page

---

Note: This means one additional API call per project with enabled wiki. . . even if the wiki doesn't have a single page.

When you have lots of repos, this has two effects concerning GitLab's rate limits:

- SCM Backup pauses after each wiki API call to avoid hitting the limit of 10 requests per second per IP address

  the more repos you have, the more time will the whole API calling take
- There's another limit of 600 API calls per minute altogether.

  you may hit that limit when you have hundreds of repos and a fast Internet connection.

Both issues can be avoided by disabling the wiki feature in all projects that don't actually use the wiki.

3.2.4 ignoreRepos

Optional: For each source, you can specify a list of repositories you do **not** want to be backed up.

Example:

```
sources:

  - title: some_title
    hoster: github
    type: user
    name: your_user_name
    ignoreRepos:
        - repo1
        - Some-Other-Repo
```

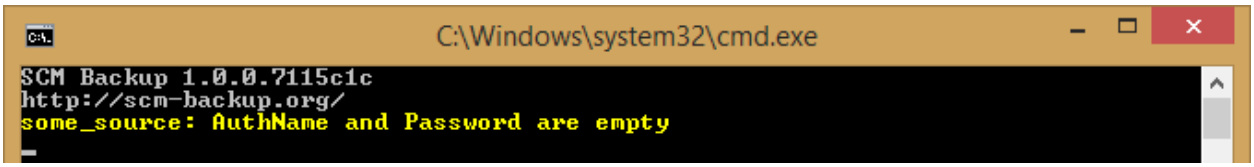
Note:

- The repository names are case-sensitive!
- For hosters where the repositories are "sub-items" of the users (like GitHub), you just need to specify the repository name, not the user name (i.e. `repo` instead of `user/repo`).

3.3 Authentication

Without authentication, SCM Backup can only backup your public repositories.

In this case, it shows a warning:



To backup your private repositories as well, you need to authenticate by setting two more properties for the source:

`authName`

Name of the user which is used for authentication.

`password`

The password/token/whatever (this varies wildly depending on the source code hoster)

Note: If your password contains at least one of the special characters listed here, you should enclose it in quotes.

E.g. `password: 'foo?'` instead of `password: foo?`

If you don't want to save your passwords directly in a config file, SCM Backup is able to get them from environment variables.

Example:

```
password: '%some_variable%'
```

If an environment variable named `some_variable` exists, the string `%some_variable%` will be replaced by the value of that variable.

This works for parts of the password as well:

```
password: 'foo%some_variable%bar'
```

Note: If you use environment variables, you **must** quote the string as shown in the examples above.

`password: %some_variable%` will not work because in YAML, strings containing `%` must always be quoted.

CHAPTER 4

Getting SCM Backup's output

For use cases where SCM Backup is running unattended, there are multiple options to get the output:

4.1 Logging

SCM Backup uses NLog for logging.

All console outputs are also generated via logging (with a CompositeLogger which logs to the console **and** to NLog).

So all console outputs are in the log files as well.

4.1.1 Log levels

To keep it simple, SCM Backup only has four log levels.

The ConsoleLogger outputs all levels except `Debug`.

The NLogLogger maps SCM Backup's log levels to a subset of NLog's log levels.

NLog is configured via NLog's regular NLog.config file, so all possible NLog configuration settings apply.

For example, you can change the minimal log level to `Debug` (default: `Info`), to log additional information:

```
<rules>
  <logger name="*" minlevel="Debug" writeTo="f" />
</rules>
```

4.1.2 Log files

The log files are in a subfolder named `logs` in SCM Backup's application folder.

On each application start, a new log file (`scm-backup.log`) is generated.

Old files are available in the `archive` subfolder.

4.2 Emailing output

The same information which is logged to the console and to the log files, can be sent via email as well.

You need to provide the SMTP settings, as well as the `From` and `To` email adresses, in the *email section in the config file*.

Note: To specify multiple recipients, separate them with a semicolon:

```
email:
  from: sender@example.com
  to: foo@example.com;bar@example.com
  [...]
```

If this is set, SCM Backup will send a mail to the specified adress after each finished backup.

4.3 Exit code

When the SCM Backup process finishes, it will return exit code `0` on success and `1` on failure.

CHAPTER 5

Restoring your backups

5.1 Folder structure

For an example of the folder structure SCM Backup creates, we'll use the default config file that comes with SCM Backup. It looks like this (only the parts that are relevant here):

```
# all backups go here
localFolder: 'c:\scm-backup'

sources:

  - title:  github_singleuser
    hoster: github
    type: user
    name: scm-backup-testuser
```

- Inside the `localFolder`, SCM Backup creates one subfolder for each source defined in the config, named like the source's `title`
- Inside that, it creates one subfolder for each repository
- Inside *that*, it creates subfolders:
  - `repo` for the actual repository
  - `wiki` if the repository has a wiki

At the time of writing, the GitHub user "scm-backup-testuser" has two visible repositories (used for integration tests):

- scm-backup-testuser/scm-backup *(has a wiki)*
- scm-backup-testuser/wiki-doesnt-exist *(doesn't have a wiki)*

So SCM Backup would create this folder structure (`C:` is the main drive on Windows):

```
C:
└──scm-backup
    └──github_singleuser
        ├──scm-backup-testuser#scm-backup
        │   ├──repo
        │   └──wiki
        └──scm-backup-testuser#wiki-doesnt-exist
            └──repo
```

Note: The `repo` and `wiki` subfolders are the bare repositories that the next section refers to.

5.2 How to restore

Generally, SCM Backup creates local repositories and pulls from the remote repositories into the local ones.

Those local repositories are **bare repositories**, i.e. they don't contain a working directory.

When you look inside the repository directories, you'll see some directories and files, depending on whether it's a Git/Mercurial/etc. repository.

Your complete history and your source code are in there - you just don't see the actual files!

The repository is backed up without a working directory, because it's not necessary.

All the data already exists inside the repository, a second copy of everything in the working directory would just be a waste of space.

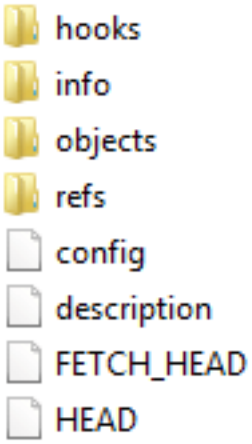
The easiest way to restore your working directory is to clone the bare repository that SCM Backup created *(called* `bare-repo` *in the examples)*, which will create a clone **with** a working directory *(called* `working-repo` *in the examples)*.

For more details, please see the sub-page for the respective SCM:

5.2.1 Restoring Git repositories

How a bare repository looks like

It contains a few folders (`objects`, `refs...`) and some files:



How to restore

Clone the bare repository into a "regular" one:

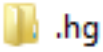
```
git clone bare-repo working-repo
```

`working-repo` will have a working directory.

5.2.2 Restoring Mercurial repositories

How a bare repository looks like

It contains a single folder named `.hg`:



How to restore

Clone

Clone the bare repository into a "regular" one:

```
hg clone bare-repo working-repo
```

`working-repo` will have a working directory.

Update

Updating a bare repo to any revision will create a working directory in that repository.

To do this, `hg update` to any revision. For the newest revision, update to `tip` *(a tag which points to the latest commit)*:

```
cd bare-repo
hg update tip
```

In TortoiseHG, right-click on any revision **Update**.

CHAPTER 6

How to Contribute

6.1 Contribute to the application

6.1.1 How to run the integration tests

For each supported hoster, SCM Backup needs to:

- make API calls to get a list of repositories
- use Git/Mercurial etc. to clone repositories

So there are integration tests for each hoster which do these things as well, some of them with authentication.

We created test users especially for these tests (for example: user and organization used for GitHub integration tests), but of course we can't publish their passwords.

So in order to run any of these integration tests, you need to setup your *own* test users and private test repositories.

SCM Backup's integration tests read the users, passwords etc. from a file named `environment-variables.ps1` in the main project directory, which is not in the repository.

You need to create your own by copying/renaming environment-variables.ps1.sample, and changing the values.

The public repositories are hardcoded in the tests, the tests just authenticate with the user/password from the environment variables.

xUnit.Net Cheat Sheet

Running a single test

Edit `run-all-tests.ps1`, look for the **INTEGRATION TESTS** section and add the `--filter` parameter to the `dotnet test` call, like this:

```
--filter "FullyQualifiedName=ScmBackup.Tests.Integration.Hosters.
↪BitbucketBackupMercurialTests.MakesBackup"
```

6.1.2 Implementing a new hoster

Steps how to implement support for backing up a new source code hoster, using the implementation for Bitbucket as an example.

Basics

In the `ScmBackup` project, create a new subfolder in the "Hosters" folder and name it like the hoster you are implementing, e.g. `Bitbucket`.

Inside the folder, create the classes listed below.

Note: SCM Backup uses naming conventions to put everything together, so make sure that:

- all classes have exactly the same prefix
- the part after the prefix is exactly like in the examples below

---

Note: To see examples, take a look at:

- **the respective classes of the existing hosters**
  - Examples: GitHub, Bitbucket
- **their tests:**
  - `...ConfigSourceValidatorTests` in the unit tests
  - `...ApiTests/...BackupTests` in the integration tests

Hoster

- Example class name: `BitbucketHoster`
- Must implement `IHoster`

ConfigSourceValidator

Validates all config sources for that hoster.

- Example class name: `BitbucketConfigSourceValidator`
- Must inherit from `ConfigSourceValidatorBase`, which implements `IConfigSourceValidator` and contains "default" rules which apply to all hosters
- Tests: Create a new class in `ScmBackup.Tests.Hosters` which inherits from `IConfigSourceValidatorTests`

Api

- Example class name: `BitbucketApi`
- Must implement `IHosterApi`
- Should call the hoster's API and return a list of repository metadata for the current user or organization

- Tests: Create a new class in `ScmBackup.Tests.Integration.Hosters` which inherits from `IHosterApiTests`

Backup

- Example class name: `BitbucketBackup`
- Must inherit from `BackupBase`, which implements `IBackup` and creates the actual backups by cloning the repositories.
- Tests: Create a new class in `ScmBackup.Tests.Integration.Hosters` which inherits from `IBackupTests`

Note: When a hoster supports multiple SCMs, you want to test backups with all of them, so you should create a separate test class for each SCM.

An example for this is Bitbucket, which supports Git **and** Mercurial, so there are BitbucketBackupGitTests and BitbucketBackupMercurialTests.

More about the tests

The base classes for the tests (`IConfigSourceValidatorTests`, `IHosterApiTests`, `IBackupTests`) contain all the tests and a few properties, some of them abstract or virtual.

The child classes just need to inherit from the respective base class and fill the properties *(for repo URLs, commit IDs etc.)*.

So the same tests are executed for each `IConfigSourceValidator`, `IHosterApi` and `IBackup` implementation (please see also *How to run the integration tests*).

Note: For special cases, which only apply to a certain implementation, you can create additional tests directly in the child class instead of the base classes.

One example for this is the Github API. There's a special quirk which only occurs in the Github API.

Because of this, we have a special integration test for this, directly in the GithubApiTests class, so it's only executed there, and not for all `IHosterApi` implementations.

Documentation

Add the hoster to the lists on the website's front page, and on the Introduction page in this documentation.

6.1.3 Implementing a new SCM

Note: To see example code, take a look at the existing SCM implementations and their tests:

- GitScm / GitScmTests
- MercurialScm / MercurialScmTests

Add ScmType

Add the new SCM to the ScmType enum.

IScm implementation

In the `ScmBackup` project, create a new class in the "Scm" folder. Name it like the SCM you are implementing, e.g. `GitScm`

The class must implement the interface IScm.

When the respective SCM has a command-line tool *(like most current SCMs do)*, the easiest way to implement the class is by inheriting from the abstract CommandLineScm class.

(`CommandLineScm` handles the plumbing to actually execute the command line tool, including looking for the executable at *the path specified in the config*)

ScmAttribute

All SCM implementations need to have an attribute, so SCM Backup is able to properly recognize them.

Apply the ScmAttribute to the class and set the `Type` parameter to the `ScmType` you added in the first step.

Example for Git:

```
namespace ScmBackup.Scm
{
    [Scm(Type = ScmType.Git)]
    internal class GitScm : CommandLineScm, IScm
    {

    }
}
```

Integration tests

In the `ScmBackup.Tests.Integration` project, create a new test class in the Scm folder which inherits from `IScmTests`. Name it accordingly, e.g. `GitScmTests`.

`IScmTests` contains all the tests and a few abstract properties for repo URLs, commit IDs etc.

The child classes just need to set these, so the same tests are executed for all `IScm` implementations.

Please see also *How to run the integration tests*.

6.2 Contribute to the documentation

The documentation is built with Sphinx and hosted on Read the Docs.

The source code is here on GitHub.

6.2.1 Headlines

To make sure that the Read the docs Sphinx theme renders correctly, it's important that the headline styling is consistent across the whole documentation.

SCM Backup's documentation uses these stylings:

- `=====` for level 1 *(the top headline of each page)*
- `-----` for level 2
- `+++++` for level 3

6.3 Making a new release

To make a new release version, the following steps must be followed:

6.3.1 1. Determine the new version number

SCM Backup uses Semantic Versioning.

The new version number **must** be in "three-digit" `MAJOR.MINOR.PATCH` format, for example `1.0.0` !!

6.3.2 2. Prepare a blog post

Here are the posts, write a new one with the list of changes.

6.3.3 3. Release the application

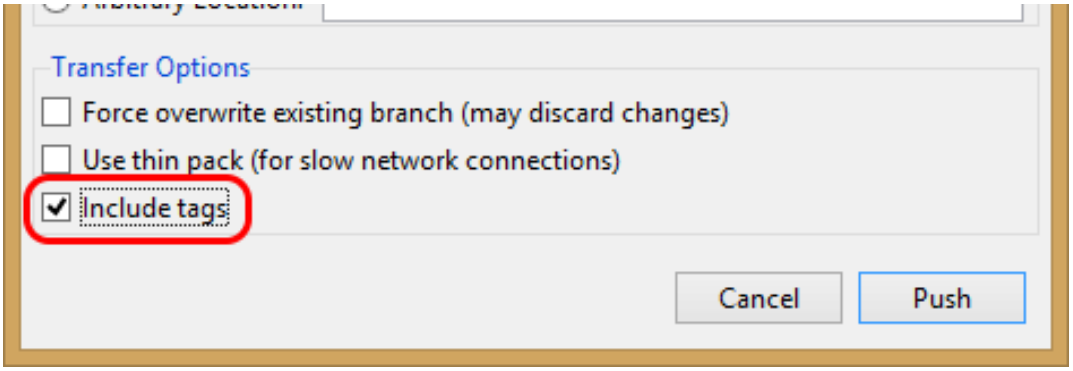
Each push to `master` creates a new CI build on AppVeyor anyway.

Create a new release by creating a Git tag in the main repository with the new version number.

The CI build will recognize this and automatically use this version number to create a new GitHub release.

Note: Don't forget to actually push the tag! Git doesn't do this automatically.

- From the command line, it's `git push origin 1.0.0`.
- In Git GUI, you need to set this checkbox when pushing:



Edit the release on GitHub and paste into the description:

- the list of changes from the blog post
- the following Markdown to show an image with the download count: `![GitHub downloads](https://img.shields.io/github/downloads/christianspecht/scm-backup/TAG/total)`

6.3.4 4. Release the docs

- Set the version and release numbers in the Sphinx configuration file `conf.py` to the new version number.

  Set `version` to the short `X.Y` format, e.g. `1.0`.

  Set `release` to the full three-digit format determined in step 1, e.g. `1.0.0`.

  Apparently Read the Docs uses this number at least in the automatically created PDF.

- Create a new Git tag *(like in the main repository)* in the documentation repository as well, **but in the short X.Y format**.

  This will create a version of the documentation for this release, making use of Read the Docs' versioning capabilities.

6.3.5 5. Publish the blog post

Period.

CHAPTER 7

Legal Stuff

7.1 License

SCM Backup is licensed under the GPL.

7.2 Acknowledgements

SCM Backup uses the following OSS projects:

- Json.NET
- MailKit
- NLog
- Octokit
- RichardSzalay.MockHttp
- Simple Injector
- xUnit.net
- YamlDotNet

Special thanks to Steven for invaluable advice.

CHAPTER 8

FAQ

8.1 Why doesn't SCM Backup support backing up local installations of [hoster X]?

SCM Backup only supports backing up from the hosters' cloud offerings. Most hosters also support some kind of "install on your own server" version of their product, **but SCM Backup doesn't support any of them, and this is very unlikely to change.**

The main reason is: Supporting local installations would make automated testing much more complicated.

SCM Backup has a load of integration tests which are executed for each supported hoster, calling the hoster's API and actually backing up test repositories - all by just using the hoster's existing cloud infrastructure.

For "install on your own server" versions, setting this up would be much more complicated: For each hoster, we'd need a local installation running somewhere... and reachable from the internet, so our CI provider (currently AppVeyor) can access it.

And not all hosters even offer a free-as-in-beer version of their local product - for example, the only version of Bitbucket Server that's freely available seems to be a 30 day test version.

Note: If you are using a local installation of a hoster supported by SCM Backup *(or any other Git/Mercurial/whatever server that can be installed on a machine you control)*, you probably don't need a tool like SCM Backup!

You (or at least someone in your organization) should have enough access to the machine, so you can directly backup the folder from that machine which contains all the repositories.

CHAPTER 9

Info for Bitbucket Backup users

Bitbucket Backup is a previous application written by the author of SCM Backup. It's similar to SCM Backup, but limited to Bitbucket and one user or team only.

Here is some useful information for Bitbucket Backup users switching to SCM Backup:

9.1 Setup

Bitbucket Backup is written in .NET 4, SCM Backup is written in .NET Core; see the different *System Requirements*.

Plus, there's no MSI setup anymore, just a zip file with the binaries.

9.2 Configuration

When you run Bitbucket Backup for the first time, it asks for all configuration values and stores them in the user's settings.

SCM Backup is able to backup multiple accounts from multiple hosters, so asking for all the config values at runtime isn't practical anymore.

Instead, you save them all into a *configuration file*.

A minimal working configuration file to backup your Bitbucket user would look like this:

```
localFolder: 'c:\your-backup-folder' # your backups are stored here

sources:

  - title: some_title  # must be unique in the whole config file, will be used as␣
↪subfolder name
    hoster: bitbucket
    type: user
```

(continues on next page)

(continued from previous page)

```
    name: your_user_name
    authName: your_user_name
    password: your_app_password
```

. . . or like this for a team:

```
localFolder: 'c:\your-backup-folder'

sources:

  - title: some_other_title
    hoster: bitbucket
    type: org
    name: your_team_name
    authName: your_user_name
    password: your_app_password
```

. . . or like this to backup both the user **and** the team *(which Bitbucket Backup can't do)*:

```
localFolder: 'c:\your-backup-folder'

sources:

  - title: some_title
    hoster: bitbucket
    type: user
    name: your_user_name
    authName: your_user_name
    password: your_app_password

  - title: some_other_title
    hoster: bitbucket
    type: org
    name: your_team_name
    authName: your_user_name
    password: your_app_password
```

Read more about possible settings for *sources* and *Bitbucket*.

9.3 Emailing output

Like Bitbucket Backup, SCM Backup is able to send an email with log information, but the configuration is different. See *how it's done in SCM Backup*.

Bitbucket Backup takes advantage of `SmtpClient`'s ability to read configuration file settings by itself.

So all possible options for `<mailSettings>` were available, and Bitbucket Backup didn't need to bother to support or even know about them all, because `SmtpClient` directly read them from the app's config file.

Apparently this is not possible in .NET Core and maybe `SmtpClient` is kind of deprecated anyway, so SCM Backup is using MailKit instead, which doesn't read values from the config and never will.

So SCM Backup has to know about every possible config value, and time will tell whether those available now will work for everyone.