RepoTools

Daniel Oberlin

Introduction

This document describes RepoTools - a set of command line programs which are used to monitor and maintain the integrity of large sets of file-based data. Over time, it is possible for files to become corrupted due to media, hardware, OS, or software failures. In some cases, data may become damaged silently. When this happens, simply copying or backing up files does not provide adequate protection. If the data integrity is not checked, then the corrupted files will be perpetuated over time and eventually the original data may become lost. The software described provides a means to detect file corruption early and to allow for the easy repair of problems when they are detected.

RepoTools operates on repositories of data which are simply file system directory structures. A repository is a directory of files and subdirectories which is associated with a manifest file that contains information about each file including a hash of the file contents. As files are added, changed, or removed from the repository the manifest file is updated so that it stays in sync with the intended content of the repository. At any time, a repository can be checked against its manifest to determine if any of the files are missing or corrupted.

Typically, the manifest file is stored just underneath the top level directory of the repository. Because of this, the process of copying a repository is as simple as copying the directory structure to another location. Once a repository has been copied, the integrity of the copy can be checked to make sure that no data was lost or corrupted during the copying process.

RepoTools provides facilities to keep a repository in sync with other copies that may exist. At any time, it is possible to diff two repositories and to report on any files that have changed contents or any files that have been added or removed from either repository. Various modes exist for synchronizing two repositories against each other. In this way, it is possible to maintain reliable backup copies or mirrors of a repository. If a repository is found to be damaged during an integrity check, it is possible to automatically repair the damaged repository with the contents of a copy.

Presently, there are three tools to support these operations:

1. RepoTool (rt) – this tool allows for the creation and maintenance of a single individual repository. This includes checking the status of a repository, validating its integrity, and updating the repository manifest when changes have been made.

2. RepoSync (rs) – this tool allows for the comparison and synchronization of a pair of repositories. This includes diffing two repositories, migrating updates from one to the other, and repairing the contents of a damaged repository using the contents of another.

3. RepoDaemon (red) – this *experimental* tool makes a repository available to RepoSync over a network via the HTTP protocol. This tool is functional, but is described in this document.

These tools were implemented using the Microsoft .NET framework. They are also usable on Mac OSX and other platforms which support the Mono framework. These tools were loosely inspired by the Git source control management system, but this software serves a different purpose and does not provide facilities for revision control.

RepoTool (rt)

Prior to using these tools, they should be installed by copying the executables and associated .DLL files to a convenient location and setting the system PATH (or OS equivalent) so that they can be used from the command line.

While learning the usage of these tools it will be helpful to create a temporary repository of files that can be worked on without risk. A directory of 20-50 small to medium sized files is ideal.

Once the tools are installed, a repository can be created by preparing an initial directory structure of files and issuing the `rt create` command from within the top level directory. By default, the manifest file is named `.repositoryManifest`, and it is placed inside the top level directory of the repository. When a repository is first created, the manifest file is empty. This can be confirmed by issuing the `rt status` command which will report a number of new files which aren't being tracked in the manifest. To complete the creation of the repository, issue the `rt update` command which will bring the manifest up to date with the contents of the repository. Checking the status a second time will show that there are no new files.

The `rt status` performs a quick check comparing the last-modified dates and file sizes against the entries in the manifest. In order to check the data integrity of the files, issue the `rt validate` command which will compute the hash of each file in the repository and check it against the hash stored in the manifest. This operation is more computationally intensive and will take longer than the status operation.

Issuing `rt help`, or simply `rt` will show the syntax and all of the commands and options that are available with the RepoTool command.

In order to see information about the repository, issue the `rt info` command. The GUID listed in the information identifies the repository. Copies or mirrors of the repository will share the same GUID. The last section of information lists regular expressions for filenames which will be ignored. The filenames use UNIX-style regular expressions and pathnames (even when running under windows). By default, the following expression is included in the ignore list:

```
^\./\.repositoryManifest$
```

The backward slash escapes special regex characters, so that this pattern matches `.\.repositoryManifest` exactly. The first '.' Indicates the root of the repository, so this pattern will ignore the manifest file of the repository, which is appropriate.

Windows users might issue the following command:

```
rt edit -ignore /Thumbs\.db$
```

which edits the repository and adds an ignore pattern file pathnames that end with /`Thumbs.db`, which are thumbnail image cache files that are added by the operating system.

In cases where many repositories are being created with large numbers of ignore patterns, it may be inconvenient to add the ignore patterns to each new repository manifest. In this case, it is possible to use a "manifest prototype" which will be used as a template for creating new manifests. Simply create an empty manifest, add the ignore paths to that manifest, and copy the manifest into the same directory that contains the repository tool executable binaries. Then rename the prototype manifest from `.repositoryManifest` to `.manifestPrototype`, and all new manifests will be automatically populated with the ignore patterns listed in the manifest prototype.

The edit command can be used in conjunction with the `–name` and `–description` options in order to provide a name and description for the contents of the repository. These are optional, but when used they should be added when the repository is created so that they are propagated consistently as the repository is copied.

Rename a file in your manifest and issue the `rt status` command. The output should indicate that one file has been removed and one file has been added. In some cases, it may be useful to identify when files have simply been moved or renamed. Using the `–trackMoves` option can provide this information at the expense of taking the time to compute hash values for files that aren't listed in the manifest. To see this in action, issue the `rt status –trackMoves` command. The output should reflect that one file has been moved. Assuming that you intended to rename the file, it makes sense to issue the `rt update` command so that the repository is brought up to date with these changes.

RepoSync (rs)

In most cases, a repository will not exist in isolation but rather in conjunction with one or more copies which are used for redundancy. Copying a repository is as easy as copying the repository directory structure to a new location. To see this, make a copy of your test repository by copying the root directory of the repository to someplace different. From the command line, change directories to the new repository copy and issue the `rt validate` command to confirm that the repository was copied without error.

The RepoSync tool manages comparisons and updates between related repositories. Compare the status of the repository copy with that of the original by issuing the following command:

```
rs diff PATHNAME_ORIGINAL PATHNAME_COPY
```

where each PATHNAME is replaced by a path to the root directory of the specified repository. The result of the command should indicate that there are no differences – that is, the repositories are in sync with each other. The diff command produced its results by comparing the manifests of the two repositories and it assumed that the manifests were up to date with the contents of the repositories. In general, a repository manifest should be updated as soon as intentional changes are made to the repository – including file modifications, file additions and file removals.

In order to see `rs diff` in a useful context, make a few changes to each of the two repositories. In the repository copy, add a new file. In the original repository, delete a file. In the repository copy, make a change to one of the files – this could be editing a text file or replacing a binary file with new data.

From the command line, update the manifests of the original repository and the repository copy to reflect these changes by using the `rs update` command. At this point, repeating the `rs diff` command should reveal the differences between the two repositories.

RepoSync provides four major functionalities: update, sync, mirror, and repair. The syntax and options of the tool are described by using the `rs help` command. Briefly:

- **update:** propagate any changes from the *source* repository to the *destination* repository, but don't change anything in the *source* repository.

- **sync:** update *source* and *destination* with anything new from either side. Use the file system last modified dates to ensure that both sides have the most recent version of any file.

- **mirror:** make *destination* match *source*. Replacing or removing files in *destination* as necessary. No changes are made in *source*.

- **repair:** replace any corrupt or damaged files in *source* with new copies from *destination* when possible.