SoftwareEntwicklung ▸ Beratung ▸ Schulung

Hook'oholic

What can be done with hook scripts?

Web Site:
www.soebes.com
Blog:
blog.soebes.com
Email:
info@soebes.com

Dipl.Ing.(FH) Karl Heinz Marbaise

Agenda

1. The Basics
2. Important Rules
3. Windows and Hooks
4. Log Message
5. Issue Tracker
6. Revision Properties
7. Lock/Unlock
8. Read Only Tags
9. Replication
10. Triggering others
11. Authorization

1. The Basics

- A Hook script runs on the Server **NOT** on the client
- A Hook is called **without any environment** neither with a PATH
- A Hook script can be implemented in any programming language you like
  - Fill in any programming language you know ;-)

1. The Basics Existing Hooks

- Commit
  - start-commit, pre-commit, post-commit
- Locking
  - pre-lock, post-lock, pre-unlock
- Revision property change
  - pre-revprop-change, post-revprop-change

1. The Basics
Existing Hooks

- The following Hooks have more or less informational character only:
  - post-commit
  - post-lock
  - post-revprop-change

1. The Basics
Existing Hooks

- They can be used to send/store information
  - Commit emails
  - Files have been locked/unlocked/stolen locks
  - Revision property changes

1. The Basics
Error Messages

- If you would like to give a message to the user just print it out to **stderr** instead of **stdout**.
- The information will be mashalled back to the client, but **only** if you abort the operation.

2. Important Rules

- **Never** change contents (neither files nor properties) during a Hook script.
  - The synchronisation of Client/Server will be lost!
- If something is not as it should be, just simply say it to the user and abort the operation.

2. Important Rules

- Use **svnlook** instead of svn client to be sure nothing can be errornously changed.
- Use **svn** only if you need other informations than directory trees. For example the size information about files.
  - svn list -r Revision URL -v

2. Important Rules

- The Hook scripts in the hook directory must executable by the user who will invoke it. This is usually the user of Apache or svnserver.
- The working directory of Hook scripts is undefined. So if you depend on it you have to set it accordingly.

2. Important Rules

- Set your localization **within** the scripts to be sure all scripts work with the localization for example en_EN.UTF8, cause you will get in trouble if you change your language/country configuration.
  - This will simplify parsing of the output of svnlook and friends.

3. Windows

- Basic Template for pre-commit.bat (for example with Perl)

```
@echo off
SET PERL=C:\Perl\bin\perl.exe
SET SCRIPTS=C:\hook-scripts
set repo="%1"
set trx="%2"
%PERL% -w -I"%SCRIPTS%" "%SCRIPTS%/pre-commit.pl"
          "%repo%" "%trx%"
set err=%errorlevel%
exit %err%
```

4. The Log Message

- The simplest one...
  - Hook Script which checks that an commit message exists. In other words it's length is greater zero.

4. The Log Message

- The simplest one
  - What do we need ?
    - The pre-commit Hook
    - **svnlook log** -t TXN REPOS to get the commit message
    - Any programming language you like

4. The Log Message

- The first (wrong) approach of the script:

```
#!/bin/bash
REPOS="$1"
TXN="$2"
MESSAGE=`svnlook log -t $TXN $REPOS`;
if [ ${#MESSAGE} -gt 0 ]; then
      exit 0;
else
      echo "A Log message must be given." >&2
      exit 1;
fi
```

4. The Log Message

- The second approach of the script:

```
#!/bin/bash
REPOS="$1"
TXN="$2"
MESSAGE=`/path/svnlook log -t "$TXN" "$REPOS"`;
if [ ${#MESSAGE} -gt 0 ]; then
      exit 0;
else
      echo "A Log message must be given." >&2
      exit 1;
fi
```

4. The Log Message

- The Windows version:

```
@echo off
setlocal
set "REPOS=%~1"
set "TXN=%~2"
for /f "tokens=*" %%i in ('C:\SVN\bin\svnlook.exe log \
      -t "%TXN%" "%REPOS%"') do set "LOGMSG=%%i"
if not "%LOGMSG%"=="" exit 0
echo. 1>&2
echo You have to give a log message! 1>&2
exit 1
```

4. The Log Message

- The simplest one
  - Hook Script which checks that the commit message length has at least 10 characters.
  - What do we need ?
    - The same as before.

4. The Log Message

The simplest one (Bash script):

```
#!/bin/bash
REPOS="$1"
TXN="$2"
MESSAGE=`/path/svnlook log -t "$TXN" "$REPOS"`;
if [ ${#MESSAGE} -gt 10 ]; then
      exit 0;
else
      echo "A Log message must be at least 10 ..." >&2
      exit 1;
fi
```

4. The Log Message

- The not so simple one:
  - Hook Script which checks the content of the commit message based on a template:
    - Check to see if all commits are done against a ticket system (valid ticket number)
    - May be you need to a have a fixed format.

5. Issue Tracker

- Via a commit you have fixed an issue and you would like to close the ticket in your issue tracking system automatically.
  - You need the post-commit Hook.
    - Many Systems like trac, JIRA have such scripts available.
  - Redmine has direct support without Hook scripts.

5. Issue Tracker

- You have to extract the issue number from the log message. Using regular expression will be a good approach.
- Check if the issue number is an existing issue number.
- May be you should check if this commit has a relationship to the issue.
  - Working on a branch (e.g. B_TICKET33)

6. Revision Properties Changing Log Msg.

- Allow to change the Log Message
  - We need the pre-revprop-change Hook.
    - We have to see if we are modifying a property.
    - Only the svn:log property is allowed to change.

6. Revision Properties Changing Log Msg.

- First approach of the script:

```
REPOS="$1"
REV="$2"
USER="$3"
PROPNAME="$4"
ACTION="$5"
...
if [ "$ACTION" = "M" -a "$PROPNAME" = "svn:log" ]; then
      exit 0;
fi
echo "Changing revision properties other than svn:log is \
prohibited" >&2
exit 1
```

6. Revision Properties Changing Log Msg.

```
REPOS="$1"
REV="$2"
USER="$3"
PROPNAME="$4"
ACTION="$5"
SVNLOOK="/usr/local/bin/svnlook"
AUTHOR=`$SVNLOOK author -r $REV "$REPOS"`
if [ "$ACTION" = "M" -a "$PROPNAME" = "svn:log" ]; then
        if [ "$AUTHOR" = "$USER" ]; then
                exit 0;
        else
                echo "You can change only your own Log-mesage" >&2
                exit 1;
        fi
fi
echo "Only svn:log is allowed to change!" >&2
exit 1
```

6. Revision Properties A Set of Properties

- Make sure to have set particular properties on particular file types.
- Some of the properties have a list of valid values. Like the following:
  - my:docstate (draft, final)
  - my:type (confidential, public, internal)

6. Revision Properties A Set of Properties

- The pre-commit Hook is the correct place.
  - The steps are:
    - Get a list of changed files/directories.
    - Extract a list of added, removed or modified files.
    - Get the properties for every file
    - Check if the properties are correct for the particular file type.

6. Revision Properties A Set of Properties

- The implementation steps:
  - Call svnlook changed ...:

```
U   project/trunk/.../x.doc
_U  project/trunk/.../x.xls
A   project/trunk/.../a.java
A   project/trunk/.../y.mak
```

  - Create a list with names and extension from the above.

6. Revision Properties A Set of Properties

- The implementation steps:
  - Check every entry, based on the extension:
    - Does this file has all required properties?
      - use svnlook proplist ....
    - Do the properties have the correct values?
      - use svnlook propget ...

7. Locking/Unlock

- Allow only people to unlock (steal locks) their own locks
  - You need to get the lock owner (via svnlook lock)
  - If there isn't a lock owner just allow locking.
  - If the lock owner is equal to the user everything is fine.

7. Locking/Unlock

- Otherwise the lock owner and the user are different
  - We don't allow to lock, cause this means someone it trying to steal someone else lock.
- The same must be implemented in the pre-unlock Hook script.
- Example can be found in the hook folder of your SVN installation.

7. Locking/Unlock

- It would be good practices to send an email (or ICQ, Jabber etc.) to inform about locking/unlocking a file.

8. Make Tags Read-Only

- Requirements
  - Having at least a naming convention and a usual folder for Tags like the best practice in SVN:
    - Multiple projects
      - /project/tags/....
    - Single project
      - /tags/...

8. Make Tags Read-Only

- Requirements
  - Everyone can create a Tag.
  - Checking out a Tag should be suitable.
  - Deleting a Tag can be done.
  - No one is allowed to change the contents of a tag which means committing on a Tag is not permitted.

8. Make Tags Read-Only

- Make Tags read only
  - svnlook will be your friend and print out the necessary information. If a Tags is created the following will be printed out by svnlook:

```
svnlook changed REPOS TXN --copy-info

A + project1/tags/TAGNR1
    (from project1/trunk/:r1)
```

8. Make Tags Read-Only Solution

- You can implement these requirements yourself via regular expressions etc. and test it...

Or you can simply download svnperms.py, create a configuration file and use it.

8. Make Tags Read-Only

- The configuration for this:

```
[repo]
trunk/.* = *(add,remove,update)
tags/[^/]+/ = *(add,remove)
branches/[^/]+/.* =
*(add,remove,update)
```

9. Replication

- Synchronizing different (read-only) Repositories via Hook scripting (using svnsync)
  - In the destination you have to activate pre-revprop-change Hook.
  - The revision 0 of the destination is used to store some information about the synchronization process.

9. Replication

- Synchronizing different (read-only) Repositories via Hook scripting
  - We use the post-commit Hook
  - Since 1.5 you can synchronize only particular code lines (e.g. branches) with svnsync.
  - You can do a replication via svnadmin dump --incremental as well.

9. Replication

- Very **important** about the replication of repositories is **not** to miss to revision-property changes which have to be synchronized to the destination as well.
  - This can be handled via post-revprop-change Hook Script.

10.Triggering others

- Build Server triggering by every commit on a particular code line
  - Trigger a CI System for a commit on a particular branch (Integration line)
  - Many CI System support polling of the Repository.
  - You should use the post-commit Hook.

11.Authorization

- Permission control which is not based on path based authorization
  - You can't really control the access, in particular reading from the Repository.
  - You can only control the committing of information to the Repository.
    - During a view to the repository ...has to be solved different

11.Authorization

- Currently the authorization is path-based (path-based-authorization) if you use it.
  - With this authorization it's simply possible to prevent people from reading, writing and seeing parts of the repository.

11.Authorization

- Via Hook scripts it's only possible to prevent people from writing something into the repository.
- The is (currently) no possibility to crontol the reading from the repository via Hook-Scripts.

A. The Missing Hook('s)

- Checkout Hook
    - start-checkout, pre-checkout, post-checkout
- Every read access to the repository might trigger an Hook script.
- Other thoughts ?

On-line Sources

- [1] Homepage of Subversion
  - http://subversion.tigris.org
- [2] Contributation Area
  - http://subversion.tigris.org/tools_contrib.html
- [2] Book about Subversion
  - http://www.svnbook.org
- [3] Subversion Forum
  - http://www.svnforum.org

On-line Sources

- [5] German Subversion forum
  - http://forum.subversionbuch.de
- [6] Forum for Software Configuration Management
  - http://www.xing.com/net/skm

Questions?

subconf2008@soebes.com

- Thank you for your attention.