hloganal

A versatile utility for analysing log files from **hrefresh**, **hsync** and **hco**. Can be used to print summaries of check-outs/synchronizes during a build, generated lists of changed files or bills of materials, and to create discrete change-sets for use with code analysis tools.

Syntax

hloganal [-debug] [-sanitize] [-D*name*=*value*] [-cfg *configfile*] [-nopurge] -exec *cmd* [*arg1* [*arg2* ...]]

hloganal [-debug] [-D*name*=*value*] [-cfg *configfile*] [-nopurge] -log *logfile*

hloganal -scan *logfile*

-exec mode

Parses the output of *cmd* looking for the a line matching the `<logfile>` regular expression. The first sub-expression is treated as the filename to process (as per `-log` mode). This is primarily intended for running **hrefresh**.

Also in this mode, the argument list is scanned for `-o` and `-oa` arguments and the parameter is treated as an additional logfile to process (as per `-log` mode). This mode is intended for **hco** and **hsync** commands.

The `-sanitize` switch is intended for use with build tools that pass badly formatted filenames to commands (e.g. double-backslash) and cleans up the command line before use.

The `-nopurge` switch disables the automatic deletion of all non-append list files.

-log mode

Processes the named *logfile*, matching `<ignore>`, `<success>`, `<failure>` and `<summary>` lines. Unmatched lines and failures are printed to the output. Optionally `<copy>` and `<list>` can be used to copy matching files to another folder tree or write the filenames to a list file. This will work with most Harvest logfiles.

-scan mode

Processes a logfile looking for Harvest-style error, information and warning messages and prints a summary of the number of occurences of each message number. This is intended to be used in analysing a typical logfile in order to write a config file for hloganal.

Configuration File

Config entries are:

`<logfile>`*server_log_regexp***`<logfile>`** - used in exec mode to locate logfiles

`<ignore>`*regexp***`</ignore>`** - log line to ignore

`<success>`*regexp***`</success>`** - log line to count as a success

`<failure>`*regexp***`</failure>`** - log line to report and count as a failure

`<summary>`*regexp***`</summary>`** - log line indicating check-in or check-out ran

`<copy dst="`*filename***`">`***regexp***`</copy>`**

- first sub-expression is treated as a filename, which has the `basedir` from the `<hloganal>` element stripped from it and the file is copied to the named destination folder using the remaining relative path

`<list file="`*filename***`" append="`**`true|false`**`" multi="`**`true|false`**`">`***regexp***`</list>`**

- first sub-expression is appended to the named list file - `append` controls whether the file is appended from the previous run - default is `false`. Non-appended files will be purged at the start of processing unless `–nopurge` is specified. `multi` controls whether the first match terminates the search (`false`), or whether all matches will write to the list file (`true`) as would be the case if you wanted to write the same entry to multiple files - default is `true`.

Regexp Syntax

^	Start of string
$	End of string
.	Any character
[a-z] or [^xyz]	Characters in range or characters not in range
(sub)	Sub-expression (often used by hloganal)
|	Alternative
?	0 or 1 occurrences
+	1 or more occurrences
*	0 or more occurrences
\	Escape character - literal backslash is \\

Examples

Example 1: Summary of a Check-Out

One very common task during a build is to check-out the latest source code from Harvest, reporting any failures. Unfortunately the interesting messages from Harvest command line tools are written into a separate logfile and most build systems do not work well with this, forcing us to write wrapper scripts to get at the output. With **hloganal**, you can easily get the output direct to the console, filtered to show just the errors:

```
hloganal -cfg example1.xml -exec hco -b broker -en project -st state
     -sy -vp viewpath -cp clientpath -eh dfofile
```

Where `example1.xml` contains:

```
<hloganal>
  <ignore>^I00060040:</ignore>
  <success>^I00020110:</success>
  <success>^I00020052:</success>
  <failure>^E[0-9]+:</failure>
  <summary>^I00060080:</summary>
</hloganal>
```

In the sample input logfile below, the green lines will be considered successes and red as failures. The light blue indicates the summary line, which can be used to detect a partially successful checkout. The dark blue line is unmatched and will be output verbatim.

```
I00060040: New connection with Broker mybkr  established.
I00020110: File \demo\DOCS\DXY2PS.TXT;1  checked out to mybkr\\C:\temp\DOCS\DXY2PS.TXT .
I00020110: File \demo\DOCS\HPGL2PS.TXT;1  checked out to mybkr\\C:\temp\DOCS\HPGL2PS.TXT .
I00020110: File \demo\DOCS\README.TXT;1  checked out to mybkr\\C:\temp\DOCS\README.TXT .
I00020110: File \demo\TESTDATA\DXYTEST\TEST1.DXY;1  checked out to
     mybkr\\C:\temp\TESTDATA\DXYTEST\TEST1.DXY .
I00020052: No need to update file C:\temp\DOCS\DXY2PS.TXT  from repository version
     \demo\DOCS\DXY2PS.TXT;2 .
E03020028: The File Agent error for item mybkr\\C:\temp\DOCS\HPGL2PS.TXT : Permission denied. .
I00060080: Check out summary: Total: 4 ; Success: 4 ; Failed: 0 ; Not Processed: 0 .
Checkout has been executed successfully.
```

As you can see in the output below, the error and unmatched lines are reported, but the remaining lines are completely digested into a series of counts. From this we can see that 5 items were successfully checked out (or not changed), 1 item failed (with the reported error) and 1 line was ignored (the connection with broker line).

```
Processing hco.log...
E03020028: The File Agent error for item mybkr\\C:\temp\DOCS\HPGL2PS.TXT : Permission denied. .
Checkout has been executed successfully.
success:   5
failure:   1
ignore:    1
summary:   1
unmatched: 1
1 failures detected
```

Example 2: List of Changed Files

During continuous integration builds, it is very useful to know what files changed between the current build and the previous build. Using either **hsync**/**hrefresh** or **hco** in sync mode, you will get a logfile containing a list of checkouts for the changed files, as well as a large number of lines for files that did not need to be updated. With **hloganal** we can take just the checkout lines, extract the filename component and then write this to a separate "change list" file, for later inclusion into the build output, or to drive other build tools:

```
hloganal -cfg example2.xml -exec hsync -b broker -en project -st state -vp viewpath -cp clientpath -sy -fv
     -iv -o logfile -eh dfofile
```

In `example2.xml` you will see that the regular expression for the checkout lines contains a sub-expression (in round brackets) which matches the client filename portion of the message (excluding the broker name):

```
<hloganal>
  <list file="changes.txt" append="false" multi="false">^I00020110: File .+  checked out to
     .+\\\\(.+) \.</list>
</hloganal>
```

Given the same input as the previous example, `changes.txt` will now contain:

```
C:\temp\DOCS\DXY2PS.TXT
C:\temp\DOCS\HPGL2PS.TXT
C:\temp\DOCS\README.TXT
C:\temp\TESTDATA\DXYTEST\TEST1.DXY
```

Example 3: Bill of Materials

For production builds, it is important to know the versions of every source item that went into the build. Similar to the change list, we can make use of the output from **hsync/hrefresh** or **hco** to get a list of every source item and its mapped version number from Harvest. The no need to update messages helpfully include the repository path and mapped version of the item, as to the checkout lines. With **hloganal** we can extract this information into a bill of materials file:

```
hloganal -cfg cfgfile -exec hrefresh -b broker -pr project -st state -iv
```

Note that with **hrefresh**, the viewpath, clientpath and dfofile all come from the hrefresh config file. hrefresh runs hsync under the covers, but unlike hsync, hrefresh can only check-out to an agent. For hloganal to find the logfile produced by hsync when run in this way, it requires and extra `<logfile>` tag which matches the logfile location message from hrefresh.

Where `example3.xml` contains:

```
<hloganal>
  <logfile>See server log file (.+) for details</logfile>
  <list file="bom.txt">^I00020110: File (.+)  checked out to .+ \.</list>
  <list file="bom.txt">^I00020052: No need to update file .+  from repository version (.+)
      \.</list>
</hloganal>
```

Given the same input as the previous examples, `bom.txt` will now contain:

```
\demo\DOCS\DXY2PS.TXT;1
\demo\DOCS\HPGL2PS.TXT;1
\demo\DOCS\README.TXT;1
\demo\TESTDATA\DXYTEST\TEST1.DXY;1
\demo\DOCS\DXY2PS.TXT;2
```

Example 4: Change-Sets

Code analysis tools are often used to check coding standards. On large codebases, it can take a long time to scan the entire codebase and developers are quickly overwhelmed with warning messages. It is often more useful to scan just the changed files and send a report at a given time interval, e.g. daily. To do this, all we need to do is to run **hsync/hrefresh** on a schedule and pass the output through **hloganal**. In response to a checkout message, we direct hloganal to copy the file to a temporary folder. After scanning the log, we have a folder containing just the changed files, which we can give to the code analysis tool.

```
<hloganal basedir="C:\temp">
  <copy dst="C:\temp2">^I00020110: File .+  checked out to .+\\\\(.+) .</copy>
</hloganal>
```

Example 5: Scan Mode

In scan mode, the hloganal looks for messages in the standard Harvest format and simply counts the number of each type. The command used is slightly different and it can only be used with an existing logfile:

```
hloganal -scan logfile
```

Again given the same input as the other examples, the output looks like this:

```
Scanning hco.log...
I00020110       4
I00060080       1
I00020052       1
I00060040       1
E03020028       1
```

This information is intended to aid writing config files for hloganal, but it can be useful for other purposes.