# hprecommit

Dual-purpose utility for running pre-commit scans of the files being checked-in or for verifying the paths to the files being checked-in or out.

In scan mode, **hprecommit** is run as a client-side UDP which copies the user's files matching the given patterns to a temporary folder and then runs the specified command. The exit code from the command is returned after the temporary folder has been deleted. In this mode it is possible to run code analysis tools to check that the code being checked-in conforms to coding standards.

In verify mode, **hprecommit** is run as a server-side UDP and verifies that the repository paths confirm to an Ant-style pattern. The exit code is 1 if any of the paths do not match, or if any of the paths are excluded. In this mode it is possible to provide limited use check-in and check-out processes that can only affect a portion of the repository, typically for build config files or a binaries area.

## Syntax

hprecommit [-e] [-o] [-s] [-c] *pattern1* [*pattern2* ...] [-x *pattern3* [*pattern4* ...]]

hprecommit [-v] [-c] *pattern1* [*pattern2* ...] [-x *pattern3* [*pattern4* ...]]

### Scan mode

This is the default mode (if `-v` is NOT specified). In this mode, **hprecommit** must run a pre-linked client-side UDP on a check-in process in order to gain access to the files that the user is attempting to check-in. The command to run and the versions and files for the check-in are passed on standard input:

```
command
[version]
[file]
```

Files and versions are reconciled into pairs and then filtered using the patterns provided. Any file that matches a pattern or is not explicitly excluded is not copied to a temporary folder. The path to the temporary folder is stored in the environment variable `WORKDIR`. The command is then run, and it will typically be passed `WORKDIR` as folder upon which to operate. The temporary folder is then deleted and the exit code of the command is returned as the exit code of the process.

The additional options `-e` and `-o` cause the exit code to be non-zero if output from the command appears on standard error and standard output respectively. The output from the command is reported on corresponding stream regardless of whether these options are specified.

The `-s` option abbreviates the output by stripping all references to the temporary working folder.

The `-c` option makes the pattern matching case-insensitive.

### Verify mode

In verify mode (`-v` was specified), **hprecommit** is run as a pre-linked UDP process on either a check-out or check-in process. It can be run either client-side or server-side.

The versions to check are passed on standard input:

```
[version]
```

An exit code of 1 is returned if any of the repository paths do not match the given patterns or are excluded, otherwise the exit code is 0.

## Pattern Syntax

The pattern syntax is similar to that used by Ant:

| | |
|---|---|
| / or \ | Directory separator |
| * | A single file or folder level or any character except a directory separator |
| ** | Any number of file or folder levels |
| ? | Any character |

## Example 1: Running PMD on Java Files

To run PMD over any Java files that are being checked-in, we add a pre-linked client-side UDP to the check-in process for the development state, with the following program:

```
hprecommit -o -e -s -c **/*.java
```

This runs hprecommit over all .java files, matching names case-insensitively. Any output from PMD on either standard output or standard error will prevent a check-in. The working folder will be stripped from the output to make it more readable.

We set the standard input to be:

```
pmd.bat -d "%WORKDIR%" -f text -R custom-java.xml
[version]
[file]
```

This runs PMD over the temporary folder containing just the Java files being checked in. We request a text report and provide a set of custom rules to check against.

It is typical to either install hprecommit and PMD in the same place on each developer's machine and reference them via an absolute path or place them on a publicly accessible read-only network share and reference them via an UNC path. In the example above they are assumed to be in the developer's PATH.

## Example 2: Check in to Binaries folder only

In a lifecycle that has builds in each (or several) of the states, we might check the binaries into a path for each state, so that we can have multiple versions of each of the build artifacts. For example, DEV might check into `\demo\binaries\DEV` and UAT into `\demo\binaries\UAT`. To do this we provide a Check-in Binaries process in each state with server-side pre-linked UDP having the following program:

```
hprecommit -v /demo/binaries/[state]/**
```

And we set standard input to be:

```
[version]
```

When this runs, it will reject any check-ins that reference repository paths outside of the binaries area for that state.

 