P4Bucket

Overview

P4Bucket is a Python script that allows administrators to move binary files stored in Perforce to be archived at a separate location – a bucket. This reduces the overall size of the depot files but not the meta-data stored in the Perforce database. The history of the files stays untouched. The files stored in the bucket can be restored again individually or as a group with the same tool.

P4Bucket will replace each moved file with a placeholder, which contains information about the location of the archived file. P4Bucket also adds some attributes to the file revision being archived which record the bucket the file was archived to as well as information about when the file was archived and by whom.

Additionally, the digest of the placeholder file will be recalculated to ensure that “p4 verify” on the whole depot comes up clean.

Usage

General usage

```
p4bucket command [options]:

init                        - initialize the connection parameters
create <name> -r <root>     - create a bucket <name> with root <root>
edit <name> -r <root>       - change the root of bucket <name> to <root>
delete <name>               - deletes the bucket (must be empty)
list [name] [-a]            - list bucket information on <name> or all
                                   buckets, option -a lists contents
archive –b <name> [-n] [opts] file[revRange] ... - archives the files
restore –b <name> [-n] file[revRange] ... - restores the files

  opts are:
     -s          Archiving of files with lazy copies is normally prevented.
                 With snap enabled, the lazy copies will be snapped first
     -f          Archiving of head revisions is normally prevented.
                 Force will archive head revisions.
     -m size     Limits archiving to files greater than size. Size can be
                 given in Bytes or a number followed by K, M, G, T or P for
                 Kilo, Mega, Giga, Tera or Petabytes.
     -n          Reporting mode. Do not archive or restore, only list.
```

Initialization

P4Bucket needs to be initialized before first usage. The tool needs one configuration file, whose location is stored in the script itself:

```
CONFIG_FILE = "p4bucket.conf"
```

Point the CONFIG_FILE variable to the absolute path where to the location of the file.

Once this variable is defined, run the script once to initialize the connection to the database:

```
python p4bucket.py init
```

This command will ask for the value of three parameters: P4PORT, P4USER, P4CLIENT and LOG.

The user specified with P4USER has to have "super" user access defined in the protection table, or archiving will fail with a Perforce error message.

The LOG determines the absolute path to a log file that protocols all archive and restore operations to make debugging problems easier (see Debugging and Troubleshooting below).

Defining and working with buckets

Create

Buckets are defined by a unique name and a location to which archived files are stored. They are created by the following command:

```
python p4bucket.py create name -r root-directory
```

The root directory does not have to exist; it will be created by the tool when the first file is archived.

Edit

If the root directory of a bucket needs to move to another location, use the following procedure:

- Move the bucket directory to its new location first
- Adjust the bucket information by using the "edit" command:

```
python p4bucket.py edit name -r root-directory
```

This is also useful if the whole bucket has originally been moved to tape, for example, and restored to a new location.

Delete

Bucket directories should never be permanently removed by hand. Instead, restore all files in that bucket (see below), then delete the bucket with the "delete" command:

```
python p4bucket.py delete name
```

List

List all buckets with the "list" command, or a particular bucket with the "list *name*" command. If you provide the "-a" option, P4Bucket will also list all files in that bucket (by performing a file system search, not by checking the Perforce server).

Archive

The archive command archives one or more eligible files in the bucket. The syntax is

```
python p4bucket.py archive name file[revRange]...
```

Eligible files are non-archived binary files, either in compressed or full-file format (type "C" or "F") that are not temporary objects (type "S").

Each file revision in the range specified will be moved to the named bucket and replaced with a placeholder file. The placeholder is a text file that looks like this:

```
# P4BUCKET: This file has been archived.
User=sknop
Date=Wed Feb 25 15:14:59 2009
```

```
        Digest=2145971CF82058B108229A3A2E3BFF35
        Bucket=bucket1
```

Lazy copies, that is, files revisions that have been created through an integration command, will not be archived. Instead, a warning is printed out, pointing out the depot file and file revision for this lazy copy.

There are several restrictions on archiving by design. Some of these restrictions can be enabled or circumvented with certain options:

Head Revision

P4Bucket will not archive head revisions by default. This follows the principle of least surprises: the standard file pattern without a revision range implies a revision range from #1 to #head, thereby archiving all revisions of this file. This is normally not desired by the user of P4Bucket.

You can force the archiving of the head revision (for example, for an obsolete branch) by using the "-f" option.

Lazy copies

Archiving of lazy copies is always prevented, again following the principle of least surprises. Archiving of a lazy copy would affect a range of separate file revisions, with probably undesired effects.

Archiving of a file revision that is the source of at least one lazy copy is also prevented, for the same reason. You can force the archiving of a source of a lazy copy with the "-s" option, but this will break (*snap*) the lazy copy, that is, create a separate copy of the librarian file for each lazy copy.

This means that by using the snap option, the total size of the depot files can actually grow instead of shrink.

The snap operation is available for cases where, for example, a release branch was created and the main branch later archived away.

Limitation on file size

Normally, files of all sizes can be archived. Using the option "-m *size*" this can be limited to files which are larger than the given size. The size can be specified in Bytes, Kilobytes (k), Megabytes (m), Gigabytes (g), Terabytes (t) and Petabytes (p).

Archiving of already archived files

Re-archiving, that is, overwriting the original file in the archive is always prevented.

Restore

The restore command restores one or more files from the bucket back to the depot. The syntax is

```
        python p4bucket.py restore name file[revRange]...
```

The bucket name specified must much the bucket to which the file has been archived.

Try before you buy™

Both *archive* and *restore* have a "-n" option that allows you to verify first what each command will do before actually executing the operation. This can be used to check which files P4Bucket will move. When using "-n", archive and restore will print out each file that would be moved to or from the bucket.

Limitations

There are several limitations to P4Bucket you should be aware of:

1. P4Bucket only works on the same machine that the server is installed on. This is checked for every command to ensure the proper working of the tool. This also implies that the server needs to be running for P4Bucket to work.
2. Lazy copies cannot be archived (see “archive”). Instead, a warning is printed for each lazy copy. Lazy copies can be identified by using the command “p4 fstat -Oc”. This will print out the *lbrFile, lbrRev* and *lbrType* of each revision specified. If the *lbrFile* does not match the *depotFile*, the revision is a lazy copy.

3. Only binary files that are either compressed or in full-file format can be archived. This excludes binary files in delta format (type “D”) and temporary files (type “S”).

DEBUGGING AND TROUBLESHOOTING

All archive and restore operations are logged in a log file that was specified in the “Init” process. Use this file to look for any problems with archived or restored files.

Additionally, each archived file has 4 attributes added: "archiveUser", "archiveDate", "archiveDigest" and “archiveBucket". This attributes can be shown with another fstat command:

```
p4 fstat –Oa file#rev
```

Attributes are shown in the output as “attr-archiveUser”, “attr-archiveDate” and so on.

There is an additional tool, “removeAttr.py” that can be used to remove the attributes of a particular file revision of necessary.

Note that P4Bucket.py uses the attributes to check if a file can be restored. If the attributes are missing, the file cannot be restored any more through P4Bucket. In this case, the file can be restored by hand, although this is not recommended.

To restore an archived file by hand replace the placeholder file with the archived file in the bucket. In addition, you will need to recalculate the digest by running the command “p4 verify –v” on the file revision you restored.