P4Transfer

A P4Python tool to transfer changes between unrelated servers

# Motivation

While there should be only one Perforce Server, the reality is often that many different Perforce Servers are required. A typical example is the Perforce public depot, which sits outside the Perforce Network.

Sometimes I start a project on one server and then realize that it would be nice to replicate these changes to another server. For example, I could start out on a local server on my laptop while being on the road but would like to make the results available on the public server.

I know I could use a DVCS for this job, but I like the power and convenience of the Perforce Server. I also like Python and P4Python, so I created a scripting solution to solve this problem: how to transfer changes between unrelated and unconnected Perforce Servers?

# Implementation

The basic idea is to create a client workspace on each Perforce Server that maps the projects to transfer. Both clients must share the same root directory and client side mapping. For example:

 Client: workspace\_server1

 Root: /home/sknop/transfer

 View:

 //depot/myproject/dev/... //workspace\_server1/myproject/...

 //depot/other/dev/... //workspace\_server1/other/...

 Client: workspace\_server2

 Root: /home/sknop/transfer

 View:

 //local\_projects/mycode/... //workspace\_server2/myproject/...

 //other\_projects/stuff/... //workspace\_server2/other/...

While the depot paths can differ, the client paths and the root directory have to match between the client workspaces.

P4Transfer works bi-directionally. The tool will inquire the changes for the workspace files and compare these to a counter. It then follows the following heuristics:

|  |  |  |
| --- | --- | --- |
| Change(Server1) | Change(Server2) | Consequence |
| Higher | Same | Copy from Server1 to Server2 |
| Same | Higher | Copy from Server2 to Server1 |
| Same | Same | Do nothing |
| Higher | Higher | Produce error and terminate |

If both server1 and server2 have been modified you will currently have to merge both parts by hand and reset the counters.

P4Transfer uses a single configuration file that contains the information of both servers as well as the current counter values. The tool updates the configuration file with the resulting counter value as it runs; the file therefore needs to be writable to the tool.

# Setup

You will need Python 2.6+ (including 3.x) and P4Python 2008.2+ to make this script work.

Create the workspaces for both servers, ensuring that the root directories and client views match.

For the initial transfer of a project that does not exist on the other server, set both change counters to 0.

If the files already exist on both sides and are in the same state at the moment, set the counters to the highest change number on each servers, respectively.

If the files differ between both servers, you will need to declare on server to have the definitive state; copy these files into the workspace of the receiving server and use ‘reconcile offline state’ to initialize the servers. Then record the change counters.

Now initialize the configuration file, by default called transfer.cfg. Here is a example:

[server1]

p4client = python-transfer

p4user = sknop

p4port = localhost:1666

#p4passwd = mysupersecret

counter = 527

[server2]

p4client = python-transfer

p4user = sknop

p4port = localhost:20101

counter = 497

The password stored in P4Passwd is optional if you do not want to rely on tickets. The tool performs a login if provided with a password, so it should work with security=3 or auth\_check trigger set.

Note that although the workspaces are named the same for both servers in this example, they are completely different entities.

A typical run of the tool would produce the following output:

~/transfer $ python ~/p4-public/P4Pythonlib/scripts/P4Transfer.py -c python.cfg

Configfile = python.cfg

server1 = [section = server1 P4PORT = localhost:1666 P4CLIENT = python-transfer P4USER = sknop P4PASSWD = None COUNTER = 527]

server2 = [section = server2 P4PORT = 20101 P4CLIENT = python-transfer P4USER = sknop P4PASSWD = None COUNTER = 497]

server1 = P4 [sknop@python-transfer localhost:1666] connected

server2 = P4 [sknop@python-transfer 20101] connected

Nothing to do.

If there are any changes missing, they will be applied consecutively.

P4Transfer has the following options:

|  |  |  |
| --- | --- | --- |
| Option | Value | Comment |
| -c | configfile | Used instead of transfer.cfg |
| -n |  | Dry run, report only |
| -v |  | verbose (not working yet) |
| -i |  | Treat integrations as adds and edits |

The -i option avoids integrations on the target side completely and instead uses adds for branches and edits for integration. This is also the strategy used if the files are integrated from outside the view of the transfer workspaces since there is no guarantee the files are available on the target.

# Usage

The original purpose of P4Transfer was to keep an offline copy of a private branch in sync. P4Offline built into P4V and the upcoming P4Sandbox are more sophisticated than my little script, and I would recommend these for offline purposes.

One feature that neither P4Offline nor P4Sandbox can address is the transfer of changes between two independent Perforce servers. This is where you might attempt to use P4Transfer.

Note that since labeling itself is not versioned no labels or tags are transferred.

# Outlook

## Integration

Branching and integrating with resolve -as, -at, -ay and -am is already implemented, as long as both source and target are within the workspace view.

I have not implemented -ae yet, and the tool will throw an error and stop if you attempt to transfer such a change. I am working on a solution here.

## Bi-directional transfer

It might be possible to attempt a transfer in both directions if the changes on both servers do not have overlapping files. There are quite a few wrinkles to sort out before I can attempt this, though, so no promises here.