P4Transfer

A P4Python tool to transfer changes between unrelated Perforce 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 you start a project on one server and then realize that it would be nice to replicate these changes to another server. For example, you 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 uni-directionally (there is an alternative called PerforceExchange which is bi-directional). The tool will inquire the changes for the workspace files and compare these to a counter.

P4Transfer uses a single configuration file that contains the information of both servers as well as the current counter values. The tool maintains is state counter using a Perforce counter on the target server (thus requiring review privilege – although since it updates changelist owners and date/time to the same as the source, it requires super user privilege anyway).

# Setup

You will need Python 2.7+ (including 3.x) and P4Python 2008.2+ to make this script work.

Note that if running it on Windows, and especially if the source server has filenames containing say umlauts or other non-ASCII characters, then Python 2.7 is required.

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

Now initialize the configuration file, by default called transfer.cfg. This can be generated by the script:

PerforceTransfer.py –sample\_config > transfer.cg

Then edit the resulting file.

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:

C:\work\> python PerforceTransfer.py -c transfer.cfg -r

2014-07-01 15:32:34,356:P4Transfer:INFO: Transferring 0 changes

2014-07-01 15:32:34,361:P4Transfer:INFO: Sleeping for 1 minutes

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

P4Transfer has various options – these are documented via the –help.

C:\> py -2 P4Transfer.py --help

usage: P4Transfer.py [-h] [-n] [-c CONFIG] [-m MAXIMUM] [-k] [-p] [-r] [-s]

 [--sample\_config] [-i]

P4Transfer

optional arguments:

 -h, --help show this help message and exit

 -n, --preview Preview only, no transfer

 -c CONFIG, --config CONFIG

 Default is transfer.cfg

 -m MAXIMUM, --maximum MAXIMUM

 Maximum number of changes to transfer

 -k, --nokeywords Do not expand keywords and remove +k from filetype

 -p, --preflight Run a sanity check first to ensure target is empty

 -r, --repeat Repeat transfer in a loop - for continuing transfer

 -s, --stoponerror Stop on any error even if --repeat has been specified

 --sample\_config Print an example config file and exit

 -i, --ignore Treat integrations as adds and edits

Copyright (C) 2012-14 Sven Erik Knop/Robert Cowham, Perforce Software Ltd

## Configuration Options

The comments in the file are mostly self-explanatory. It is important to specify the main values for the [source] and [target] sections.

In the [general] section, you can customize the change\_description\_format value to decide how transferred change descriptions are formatted.

Keywords in the format string are prefixed with $. Use \n for newlines. Keywords allowed are: $sourceDescription, $sourceChange, $sourcePort, $sourceUser.

### Examples

Assume the source description is “Original change description”.

Default format:

$sourceDescription\n\nTransferred from p4://$sourcePort@$sourceChange

might produce:

Original change description

Transferred from p4://source-server:1667@2342

Custom format:

Originally $sourceChange by $sourceUser on $sourcePort\n$sourceDescription

might produce:

Originally 2342 by FBlogs on source-server:1667

Original change description

# Usage

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

## Integration

Branching and integrating with is implemented, as long as both source and target are within the workspace view. Otherwise, the integrate action is downgraded to an add or edit.

## Setting up as a service on Windows

P4Transfer can be setup as a service on Windows using srvinst.exe and srvanay.exe to wrap the Python interpreter.

Please contact consulting@perforce.com for more details.