P4Transfer - Helix Core Full History Migration Tool

Perforce Professional Services

Version v2021.1, 2021-01-29

Table of Contents

Chapter 1. Introduction

The script P4Transfer.py solves the problem: how to transfer changes between unrelated and unconnected Perforce Helix Core repositories.

If you have a 2015.1 or greater Helix Core server then you can use the new P4DVCS commands (`clone`/`fetch`/`push`). If you have an older version of Helix Core (or very large repository sizes to be transferred) then `P4Transfer.py` may be your best option! See next section for guidance.

While best practice is usually to have 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 a laptop while being on the road but would like to make the results available on the public server. You may also wish to consolidate various Perforce servers into one with all their history (e.g. after an acquisition of another company).

Chapter 2. P4Transfer vs Helix native DVCS functions

Both Helix native DVCS and P4Transfer enable migration of detailed file history from a set of paths on one Helix Core server into another server.

Helix native DVCS has additional functionality, such as creating a personal micro-repo for personal use without a continuously running 'p4d' process.

When it is an option, using Helix native DVCS is preferred. But if that can't be used for some reason, P4Transfer can pretty much always be made to work.

2.1. Guidance on tool differences

Pros for native DVCS:

- Helix native DVCS is a fully supported product feature.
- Helix native DVCS requires consistency among the Helix Core servers: Same case sensitivity setting, same Unicode mode, and same P4D version (with some exceptions).
- Helix native DVCS is easy to setup and use.
- Helix native DVCS has some limitations at large scale (depends on server RAM).
- Helix native DVCS is generally deemed to product the highest possible quality data on the target server, as it transfers raw journal data and archive content.
- Helix native DVCS is very fast.
- Helix native DVCS has no external dependencies.
- Helix native DVCS bypasses triggers.

When P4Transfer might be a better option:

- If Helix native DVCS hits data snags, there may not be an easy way to work around them, unless a new P4D server release address them.
- Helix native DVCS must be explicitly enabled. If not already enabled, it requires 'super' access on both Helix Core servers involved in the process.
- P4Transfer is community supported (and with paid Consulting engagements).
- P4Transfer can make timestamps more accurate if it has 'admin' level access, but does not require it.
- P4Transfer automates front-door workflows, i.e. it does would a human with a 'p4' command line client, fleet fingers and a lot of time could do.
- P4Transfer can work with mismatched server data sets (different case sensitivity, Unicode mode, or P4D settings) **so long as** the actual data to be migrated doesn't cause issues. (For example, if you try to copy paths that actually do have Korean glyphs in the path name to a non-Unicode server, that ain't gonna work.).

- If P4Transfer does have issues with importing some data (like the above example with Unicode paths), you can manually work around those snags, and then pick up with the next changelist. If there aren't too many snags, this trial and error process is viable.
- P4Transfer is reasonably fast as a front-door mechanism.
- P4Transfer can be run as a service for continuous operation, and has successfully run for months or more.
- P4Transfer data quality is excellent in terms of detail (e.g. integration history, resolve options, etc.). However, it is an emulation of history rather than a raw replay of actually history in the native format as done by Helix native DVCS. The nuance in history differences rarely has practical implications.
- P4Transfer requires more initial setup than Helix native DVCS.
- P4Transer requires Python (2.7 or 3.6+).
- P4Transfer can be interfered with by custom policy enforcement triggers on the target server.

Chapter 3. Implementation

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

Source client:

```
Client: workspace_server1

Root: /work/transfer

View:
    //depot/myproject/dev/... //workspace_server1/myproject/...
    //depot/other/dev/... //workspace_server1/other/...
```

Target client:

```
Client: workspace_server2

Root: /work/transfer

View:
    //import/mycode/... //workspace_server2/myproject/...
    //import/stuff/... //workspace_server2/other/...
```

While the depot paths can differ, the client paths (thus the right hand sides of the view mappings) and the root directory have to match between the source/target client workspaces.

P4Transfer works uni-directionally. 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 its state counter using a Perforce counter on the target server (thus requiring `review` privilege as well as `write` privilege – by default it assumes `super` user privilege is required since it updates changelist owners and date/time to the same as the source – this functionality is controlled by the config file).

Chapter 4. Setup

You will need Python 2.7 or 3.6+ and P4Python 2017.2+ to make this script work.

The easiest way to install P4Python is probably using "pip" – make sure this is installed. Then:

```
pip install p4python
```

If the above needs to build and fails, then this usually works for Python 3.6: `pip install p4python==2017.2.1615960`

Alternatively, refer to P4Python Docs

If you are on Windows, then look for an appropriate version on the Perforce ftp site (for your Python version), e.g. http://ftp.perforce.com/perforce/r20.1/bin.ntx64/

4.1. Installing P4Transfer.py

The easiest thing to do is to download this repo either by:

- running `git clone https://github.com/perforce/p4transfer.git`
- or by downloading the project zip file and unzipping.

The minimum requirements are the modules `P4Transfer.py` and `logutils.py`

4.2. Getting started

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 currently due to the way Unicode is processed. Python 3.6+ on Mac/Unix should be fine with Unicode as long as you are using P4Python 2017.2+

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:

```
python3 P4Transfer.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\> python3 P4Transfer.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.

4.3. Script parameters

P4Transfer has various options – these are documented via the `-h` or `--help` parameters.

```
$ python3 P4Transfer.py -h
usage: P4Transfer.py [-h] [-c CONFIG] [-m MAXIMUM] [-k] [-r] [-s] [--sample-config] [-
i]
                     [--end-datetime END_DATETIME]

P4Transfer

optional arguments:
  -h, --help            show this help message and exit
  -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
  -r, --repeat          Repeat transfer in a loop - for continuous 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
  --end-datetime END_DATETIME
                        Time to stop transfers, format: 'YYYY/MM/DD HH:mm'

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

4.4. Optional Parameters

- `--maximum` - useful to perform a test transfer of a single changelist when you get started (although remember this might be a changelist with a lot of files!)
- `--keywords` - useful to avoid issues with expanding of keywords on a different server - this makes it hard to compare source/target results.
- `--end-datetime` - useful to schedule a run of P4Transfer and have it stop at the desired time (e.g. run overnight and stop when users start in the morning). Useful for long running transfers (can be many days)

4.5. 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.

```
P4Transfer.py --sample-config > transfer.cfg
```

```
cat transfer.cfg
```

```
# Save this output to a file to e.g. transfer.cfg and edit it for your configuration

[general]
# counter_name: Unique counter on target server to use for recording source changes
processed. No spaces.
#    Name sensibly if you have multiple instances transferring into the same target p4
repository.
#    The counter value represents the last transferred change number - script will
start from next change.
#    If not set, or 0 then transfer will start from first change.
counter_name = p4transfer_counter

# instance_name: Name of the instance of P4Transfer - for emails etc. Spaces allowed.
instance_name = Perforce Transfer from XYZ

# For notification - if smtp not available - expects a pre-configured nms FormMail
script as a URL
mail_form_url =

# The mail_* parameters must all be valid (non-blank) to receive email updates during
processing.
# mail_to: One or more valid email addresses - comma separated for multiple values
#     E.g. somebody@example.com,somebody-else@example.com
mail_to =

# mail_from: Email address of sender of emails, E.g. p4transfer@example.com
mail_from =

# mail_server: The SMTP server to connect to for email sending, E.g.
smtpserver.example.com
mail_server =

# ================================================================================
# Note that for any of the following parameters identified as (Integer) you can
specify a
# valid python expression which evaluates to integer value, e.g.
#     24 * 60
#     7 * 24 * 60
```

```
# ------------------------------------------------------------------------------
# sleep_on_error_interval (Integer): How long (in minutes) to sleep when error is
encountered in the script
sleep_on_error_interval = 60

# poll_interval (Integer): How long (in minutes) to wait between polling source server
for new changes
poll_interval = 60

# change_batch_size (Integer): changelists are processed in batches of this size
change_batch_size = 20000

# The following *_interval values result in reports, but only if mail_* values are
specified
# report_interval (Integer): Interval (in minutes) between regular update emails being
sent
report_interval = 30

# error_report_interval (Integer): Interval (in minutes) between error emails being
sent e.g. connection error
#     Usually some value less than report_interval. Useful if transfer being run with
--repeat option.
error_report_interval = 15

# summary_report_interval (Integer): Interval (in minutes) between summary emails
being sent e.g. changes processed
#     Typically some value such as 1 week (10080 = 7 * 24 * 60). Useful if transfer
being run with --repeat option.
summary_report_interval = 7 * 24 * 60

# sync_progress_size_interval (Integer): Size in bytes controlling when syncs are
reported to log file.
#    Useful for keeping an eye on progress for large syncs over slow network links.
sync_progress_size_interval = 500 * 1000 * 1000

# change_description_format: The standard format for transferred changes.
#    Keywords prefixed with $. Use \n for newlines. Keywords allowed:
#     $sourceDescription, $sourceChange, $sourcePort, $sourceUser
change_description_format = $sourceDescription\n\nTransferred from
p4://$sourcePort@$sourceChange

# change_map_file: Name of an (optional) CSV file listing mappings of source/target
changelists.
#    If this is blank (DEFAULT) then no mapping file is created.
#    If non-blank, then a file with this name in the target workspace is appended to
#    and will be submitted after every sequence (batch_size) of changes is made.
#    Default type of this file is text+CS32 to avoid storing too many revisions.
#    File must be mapped into target client workspace.
#    File can contain a sub-directory, e.g. change_map/change_map.csv
change_map_file =
```

```
# superuser: Set to n if not a superuser (so can't update change times - can just
transfer them).
superuser = y

[source]
# P4PORT to connect to, e.g. some-server:1666
p4port =
# P4USER to use
p4user =
# P4CLIENT to use, e.g. p4-transfer-client
p4client =
# P4PASSWD for the user - valid password. If blank then no login performed.
# Recommended to make sure user is in a group with a long password timeout!.
p4passwd =

[target]
# P4PORT to connect to, e.g. some-server:1666
p4port =
# P4USER to use
p4user =
# P4CLIENT to use, e.g. p4-transfer-client
p4client =
# P4PASSWD for the user - valid password. If blank then no login performed.
# Recommended to make sure user is in a group with a long password timeout!.
p4passwd =
```

4.5.1. Changelist comment formatting

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`.

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
```

4.5.2. Recording a change list mapping file

There is an option in the configuration file to specify a change_map_file. If you set this option (default is blank), then P4Transfer will append rows to the specified CSV file showing the relationship between source and target changelists, and will automatically check that file in after every process.

```
change_map_file = change_map.csv
```

The result change map file might look something like this:

```
$ head change_map.csv
sourceP4Port,sourceChangeNo,targetChangeNo
src-server:1666,1231,12244
src-server:1666,1232,12245
src-server:1666,1233,12246
src_server:1666,1234,12247
src-server:1666,1235,12248
```

It is very straight forward to use standard tools such as grep to search this file. Because it is checked in to the target server, you can also use “p4 grep”.

Chapter 5. Usage

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

5.1. 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.

5.2. 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, or NSSM - The Non-Sucking Service Manager

Please contact `consulting@perforce.com` for more details.

Chapter 6. Support

Any errors in the script are highly likely to be due to some unusual integration history, which may have been done with an older version of the Perforce server.

If you have an error when running the script, please use summarise_log.sh to create a summary log file to send. E.g.

```
summarise_log.sh log-P4Transfer-20141208094716.log > sum.log
```

If you get an error message in the log file such as:

```
P4TLogicException: Replication failure: missing elements in target changelist:
/work/p4transfer/main/applications/util/Utils.java
```

or

```
P4TLogicException: Replication failure: src/target content differences found: rev = 1
action = branch type = text depotFile = //depot/main/applications/util/Utils.java
```

Then please also send the following:

A Revision Graph screen shot from the source server showing the specified file around the changelist which is being replicated. If an integration is involved then it is important to show the source of the integration.

Filelog output for the file in the source Perforce repository, and filelog output for the source of the integrate being performed. e.g.

```
p4 -ztag filelog /work/p4transfer/main/applications/util/Utils.java@12412
p4 -ztag filelog /work/p4transfer/dev/applications/util/Utils.java@12412
```

where 12412 is the changelist number being replicated when the problem occurred.

6.1. Re-running P4Transfer after an error

When an error has been fixed, you can usually re-start P4Transfer from where it left off. If the error occurred when validating changelist say 4253 on the target (which was say 12412 on the source) but found to be incorrect, the process is:

```
p4 -p target-p4:1666 -u transfer_user -c transfer_workspace obliterate
//transfer_workspace/...@4253,4253
```

```
(re-run the above with the -y flag to actually perform the obliterate)
```

Ensure that the counter specified in your config file is set to a value less than 4253 such as the changelist immediately prior to that changelist. Then re-run P4Transfer as previously.