FetchTransfer - Helix Core Incremental History Migration Tool

Perforce Professional Services

Version v2021.1, 2021-09-08

Table of Contents

Chapter 1. Introduction

The script FetchTransfer.py is a wrapper around the underlying `p4 fetch` to the problem: how to transfer changes between unrelated and unconnected Perforce Helix Core repositories.

This requires a 2015.1 or greater Helix Core server with the Helix native DVCS commands (`p4 clone` /`p4 fetch`/`p4 push`). If you have a pre-2015.1 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. FetchTransfer vs Helix native DVCS functions

See also link:P4Transfer.adoc for reasons when that would be better than FetchTransfer.

Both Helix native DVCS and FetchTransfer 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, FetchTransfer can pretty much always be made to work.

2.1. Guidance on tool differences

Common features:

- `FetchTransfer.py` is a wrapper around the underlying Helix native DVCS.

Pros for basic 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 requires particular configurables to be setup, and may be viewed as a security concern for the extraction of history (this can be controlled with pre-command triggers)
- Helix native DVCS is generally deemed to produce 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 FetchTransfer might be a better option:

- Helix native DVCS has some limitations at larger scales (depends on server RAM as it creates zip files) - won’t scale to TB requiring to be transferred.
- 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.

- FetchTransfer is community supported (and with paid Consulting engagements).
- FetchTransfer can make timestamps more accurate if it has 'admin' level access, but does not require it.
- FetchTransfer can be run as a service for continuous operation.
- FetchTransfer requires a little more initial setup than Helix native DVCS.

Chapter 3. Implementation

Like DVCS, FetchTransfer uses a remote spec to specify mappings between source and target servers. For validation of changelist contents, and also submission of change-map files, there are also client workspaces which duplicate the mappings.

The remote spec and the client workspaces are created automatically from the `view` entries in the config file described below.

FetchTransfer works uni-directionally. The tool will inquire the changes for the workspace files and compare these to a counter.

FetchTransfer 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).

In addition the configurables `server.allowfetch` must be set.

3.1. Classic/local target depots vs Streams targets

TBD

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" (or "pip3") – 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: `pip3 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 FetchTransfer.py

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

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

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

If you have installed P4Python as above, then check the `requirements.txt` for other modules to install via `pip` or `pip3`.

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 may be 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+

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

```
python3 FetchTransfer.py ▯-sample-config > transfer.yaml
```

Then edit the resulting file, paying attention to the comments.

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 FetchTransfer.py -c transfer.yaml -r
2014-07-01 15:32:34,356:FetchTransfer:INFO: Transferring 0 changes
2014-07-01 15:32:34,361:FetchTransfer:INFO: Sleeping for 1 minutes
```

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

4.3. Script parameters

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

The following text may not display properly if your are viewing this FetchTransfer.adoc file in GitHub. Please refer the the .pdf version instead, or open up help.txt file directly.

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

NAME:
    P4Transfer.py

DESCRIPTION:
    This python script (2.7/3.6+ compatible) will transfer Perforce changelists with
all contents
    between independent servers when no remote depots are possible, and P4 DVCS
commands
    (such as p4 clone/fetch/zip/unzip) are not an option.

    This script transfers changes in one direction - from a source server to a target
server.

    Usage:

        python3 P4Transfer.py -h

    The script requires a config file, by default transfer.yaml,
    that provides the Perforce connection information for both servers.

    An initial example can be generated, e.g.

        P4Transfer.py --sample-config > transfer.yaml

    For full documentation/usage, see project doc:

        https://github.com/perforce/p4transfer/blob/main/doc/P4Transfer.adoc
```

```
optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        Default is transfer.yaml
  -n, --notransfer      Validate config file and setup source/target
                        workspaces but don't transfer anything
  -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 as
                        background task
  -s, --stoponerror     Stop on any error even if --repeat has been specified
  --sample-config       Print an example config file and exit
  -i, --ignore-integrations
                        Treat integrations as adds and edits
  --end-datetime END_DATETIME
                        Time to stop transfers, format: 'YYYY/MM/DD HH:mm' -
                        useful for automation runs during quiet periods e.g.
                        run overnight but stop first thing in the morning

Copyright (C) 2012-21 Sven Erik Knop/Robert Cowham, Perforce Software Ltd
```

4.4. Optional Parameters

- `--notransfer` - useful to validate your config file and that you can connect to source/target p4d servers, and report on how many changes might be transferred.
- `--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!)
- `--end-datetime` - useful to schedule a run of FetchTransfer and have it stop at the desired time (e.g. run overnight and stop when users start work in the morning). Useful for scheduling long running transfers (can be many days) in quiet periods (e.g. together with Linux `at` command)

4.5. Long running jobs

On Linux, recommend you execute it as a background process, and then monitor the output:

```
nohup python3 FetchTransfer.py -c transfer.yaml -r > out1 &
```

This will run in the background, and poll for new changes (according to `poll_interval` in the config file.)

You can look at the output file for progress, e.g. (avoiding long lines of text which can be output), or grep the log file:

```
tail -f out1 | cut -c -140
```

```
grep :INFO: log-FetchTransfer-*.log
```

Note that if you edit the configuration file, it will be re-read the next time the script wakes up and polls for input. So you do not need to stop and restart the job.

4.6. Setting up environment

The following simple setup will allow you to cross check easily source and target servers. Assume we are in a directory: `/some/path/FetchTransfer`

```
export P4CONFIG=.p4config
mkdir source target
cd source
vi .p4config
```

and create appropriate values as per your config file for the source server e.g.:

```
cat .p4config
P4PORT=source-server:1666
P4USER=FetchTransfer
P4CLIENT=FetchTransfer_client
```

And similarly create a file in the `target` sub-directory.

This will allow you to quickly and easily `cd` between directories and be able to run commands against respective source and target p4d instances.

4.7. 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.

```
FetchTransfer.py --sample-config > transfer.yaml
```

```
cat transfer.yaml
```

The following included text may not display correctly when this .adoc file is viewed in GitHub - instead download the PDF version of this doc, or open fetch_transfer.yaml directly.

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

# 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: FetchTransfer_counter

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

# For notification - if smtp not available - expects a pre-configured nms FormMail
script as a URL
#   E.g. expects to post using 2 fields: subject, message
# Alternatively, use the following entries (suitable adjusted) to use Mailgun for
notifications
#   api: "<Mailgun API key"
#   url: "https://api.mailgun.net/v3/<domain or sandbox>"
#   mail_from: "Fred <fred@example.com>"
#   mail_to:
#   - "fred@example.com"
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"
# Such values should be quoted (in order to be treated as strings)
# -------------------------------------------------------------------------------
# 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: 1000

# 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

# max_logfile_size (Integer): Max size of file to (in bytes) after which it should be
rotated
#     Typically some value such as 20MB = 20 * 1024 * 1024. Useful if transfer being
run with --repeat option.
max_logfile_size: 20 * 1024 * 1024

# 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
#    Note that due to the way client workspace views are created the local filename
#    should include a valid source path including depot name, e.g.
#       //depot/export/... -> depot/export/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 - if this is on localhost and you
just
    # want to specify port number, then use quotes: "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!.
    # Make sure your P4TICKETS file is correctly found in the environment
  p4passwd:
    # P4CHARSET to use, e.g. none, utf8, etc - leave blank for non-unicode p4d
instance
  p4charset:

target:
    # P4PORT to connect to, e.g. some-server:1666 - if this is on localhost and you
just
    # want to specify port number, then use quotes: "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!
    # Make sure your P4TICKETS file is correctly found in the environment
  p4passwd:
    # P4CHARSET to use, e.g. none, utf8, etc - leave blank for non-unicode p4d
instance
  p4charset:

# workspace_root: Root directory to use for both client workspaces.
#    This will be used to update the client workspace Root: field for both
source/target workspaces
#    They must be the same.
#    Only really used by target to check in the map file if specified
workspace_root: /work/transfer

# target_remote: Name of remote spec to setup - ensure this is unique and only used by
this script!
target_remote: FetchTransfer_remote

# views: An array of source/target view mappings
#    You are not allowed to specify both 'views' and 'stream_views' - leave one or
other blank!!
#    Each value is a string - normally quote. Standard p4 wildcards are valid.
#    These values are used to construct the appropriate View: fields for source/target
client workspaces
```

```
#    It is allowed to have exclusion mappings - by specifying the '-' as first
character in 'src'
#    entry - see last example below.
views:
- src: //depot/source_path1/...
  targ: //import/target_path1/...
- src: //depot/source_path2/...
  targ: //import/target_path2/...
- src: -//depot/source_path2/exclude/*.tgz
  targ: //import/target_path2/exclude/*.tgz
```

4.7.1. Changelist comment formatting

TBC

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
```

4.7.2. Recording a change list mapping file

TBC

There is an option in the configuration file to specify a change_map_file. If you set this option (default is blank), then FetchTransfer 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 = depot/import/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”.

> You will need to ensure that the change_map filename is properly mapped in the local workspace - thus it must include `<depot>` and other pathname components in the path. When you have created your target workspace, run `p4 client -o` to check the view mapping.

Chapter 5. Misc Usage Notes

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

5.1. Setting up as a service on Windows

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

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

6.1. Re-running FetchTransfer after an error

When an error has been fixed, you can usually re-start FetchTransfer 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 FetchTransfer as previously.

Chapter 7. Contributor’s Guide

Pull Requests are welcome. Code changes should normally be accompanied by tests.

See TestFetchTransfer.py for unit/integration tests.

Most tests generate a new `p4d` repository with source/target servers and run test transfers.

The use the "rsh" hack to avoid having to spawn p4d on a port.

7.1. Test dependencies

Tests assume there is a valid `p4d` in your current PATH.

7.2. Running a single test

Pick your single test class (e.g. `testAdd`):

```
python3 TestFetchTransfer.py TestFetchTransfer.testAdd
```

This will:

- generate a single log file: `log-TestFetchTransfer-*.log`
- create a test sub-directory `_testrun_transfer` with the following structure:

  ```
  source/
      server/         # P4ROOT and other files for server - uses rsh hack for p4d
      client/         # Root of client used to checkin files
      .p4config       # Defines P4PORT for source server
  target/             # Similar structure to source/
  transfer_client/    # The root of shared transfer client
  ```

This test directory is created new for each test, and then left behind in case of test failures. If you want to manually do tests or view results, then `export P4CONFIG=.p4config`, and `cd` into the source/target directory to be able to run normal `p4` commands as appropriate.

7.3. Running all tests

```
python3 TestFetchTransfer.py
```

It will generate many log files (`log-TestFetchTransfer-*.log`) which can be examined in case of failure or removed.