Server Deployment Package (SDP) for Perforce Helix

Unsupported Scripts and Triggers

Perforce Professional Services

Version v2025.2, 2025-12-05

Table of Contents

Preface . . . 1
1. Samples . . . 2
1.1. bin/htd_move_logs.sh . . . 2
1.2. bin/p4web_base . . . 2
1.3. broker/one_per_user.sh . . . 2
1.4. Triggers . . . 2
1.4.1. Workflow Enforcement Triggers . . . 2
1.4.2. CheckCaseTrigger.py . . . 3
1.4.3. CheckChangeDesc.py . . . 4
1.4.4. CheckFixes.py . . . 5
1.4.5. CheckFixes.yaml . . . 6
1.4.6. CheckFolderStructure.py . . . 6
1.4.7. CheckJobEditTrigger.py . . . 7
1.4.8. CheckStreamNameFormat.py . . . 8
1.4.9. CheckSubmitHasReview.py . . . 9
1.4.10. ControlStreamCreation.py . . . 10
1.4.11. CreateSwarmReview.py . . . 12
1.4.12. DefaultChangeDesc.py . . . 12
1.4.13. DefaultSwarmReviewDesc.py . . . 13
1.4.14. DefaultSwarmReviewDesc.yaml . . . 14
1.4.15. JobIncrement.pl . . . 14
1.4.16. JobsCmdFilter.py . . . 14
1.4.17. P4Triggers.py . . . 15
1.4.18. PreventWsNonAscii.py . . . 15
1.4.19. RequireJob.py . . . 15
1.4.20. SetLabelOptions.py . . . 16
1.4.21. SwarmReviewTemplate.py . . . 16
1.4.22. TFSJobCheck.py . . . 17
1.4.23. ValidateContentFormat.py . . . 17
1.4.24. Workflow.yaml . . . 18
1.4.25. WorkflowTriggers.py . . . 18
1.4.26. archive_long_name_trigger.pl . . . 18
1.4.27. changeid.pl . . . 18
1.4.28. command_block.py . . . 19
1.4.29. dictionary . . . 19
1.4.30. externalcopy.txt . . . 19
1.4.31. otpauthcheck.py . . . 20
1.4.32. otpkeygen.py . . . 20

1.4.33. rad_authcheck.py . . . 20
1.4.34. radtest.py . . . 20
1.4.35. submit_form_1.py . . . 21
1.4.36. submit_form_1_in.py . . . 21
1.5. triggers / tests . . . 21
2. Maintenance . . . 22
2.1. accessdates.py . . . 22
2.2. add_users.sh . . . 22
2.3. addusertogroup.py . . . 22
2.4. checkusers.py . . . 23
2.5. checkusers_not_in_group.py . . . 23
2.6. clean_protect.py . . . 23
2.7. convert_label_to_autoreload.py . . . 24
2.8. convert_rcs_to_unix.sh . . . 24
2.9. countrevs.py . . . 24
2.10. creategroups.py . . . 24
2.11. createusers.py . . . 25
2.12. del_shelve.py . . . 25
2.13. delusers.py . . . 25
2.14. edge_maintenance . . . 25
2.15. email.bat . . . 26
2.16. email.sh . . . 26
2.17. email_pending_client_deletes.py . . . 26
2.18. email_pending_user_deletes.py . . . 26
2.19. EvilTwinDetector.sh . . . 26
2.20. group_audit.py . . . 30
2.21. isitalabel.py . . . 30
2.22. license_status_check.sh . . . 30
2.23. lowercp.py . . . 30
2.24. lowertree.py . . . 30
2.25. maintain_user_from_groups.py . . . 31
2.26. maintenance . . . 31
2.27. maintenance.cfg . . . 31
2.28. make_email_list.py . . . 31
2.29. mirroraccess.py . . . 31
2.30. p4deleteuser.py . . . 32
2.31. p4lock.py . . . 32
2.32. p4unlock.py . . . 32
2.33. protect_groups.py . . . 32
2.34. proxysearch.py . . . 33
2.35. pymail.py . . . 33

2.36. remove_empty_pending_changes.py . . . 33
2.37. remove_jobs.py . . . 33
2.38. removeuserfromgroups.py . . . 34
2.39. removeusersfromgroup.py . . . 34
2.40. sample_cron_entries.txt . . . 34
2.41. sdputils.py . . . 34
2.42. server_status.sh . . . 34
2.43. setpass.py . . . 34
2.44. template.maintenance.cfg . . . 35
2.45. unload_clients.py . . . 35
2.46. unload_clients_with_delete.py . . . 35
2.47. unload_labels.py . . . 35

Preface

This page contains documentation for the SDP Unsupported folder. This folder contains sample scripts that have proven useful in the past. However, they are not maintained nor included in the automated test suite.

Unlike the rest of the SDP, these scripts here in this Unsupported folder are NOT officially supported. They are NOT fully tested and NOT guaranteed to work. Treat files in here with some caution, and be sure to test before using in your own environment.

All triggers and scripts in the `Unsupported` folder are provided as examples. They are Community supported only. Most have worked in customer environments at some point, but they are not maintained and tested at the same quality levels as the main SDP scripts.

Please Give Us Feedback

Perforce welcomes feedback from our users. Please send any suggestions for improving this document or the SDP to consulting-helix-core@perforce.com.

© 2010-2025 Perforce Software, Inc.

Chapter 1. Samples

Scripts and utilities in this folder are examples which were part of the SDP in the past.

1.1. bin/htd_move_logs.sh

Script to compress and move Helix Server structured audit logs

Implementation assumptions and suggestions: * Assumes the rotated log files are named audit-nnn.csv * Do NOT configure your log files to be placed in $P4ROOT * Set TARGETDIR in script

1.2. bin/p4web_base

P4Web base init script for running a p4web instance. Similar function for `/p4/common/bin/p4d_base`.

Please refer to SDP_Guide.Unix for more details.

Can be used from service files or even systemd service definitions.

This is located here because **P4Web** is a deprecated and unsupported product.

1.3. broker/one_per_user.sh

This broker filter script limits commands of a given type to one process running at a time for the same user. It relies on `p4 monitor` having been enabled with `p4 configure set monitor=1` (or higher than 1).

This is called by the `p4broker` process. The p4broker provides an array of fields on STDIN, e.g. "command: populate" and "user: joe", which get parsed to inform the business logic of this script.

This is enabled by adding a block like the following to a broker config file, with this example limiting the `p4 populate` command:

```
command: ^populate$
{
    action  = filter;
    execute = /p4/common/bin/broker/one_per_user.sh;
}
```

1.4. Triggers

1.4.1. Workflow Enforcement Triggers

These triggers are documented in HTML doc or PDF doc

Where appropriate below reference will be made to this section.

© 2010-2025 Perforce Software, Inc.

1.4.2. CheckCaseTrigger.py

This trigger checks for attempts to add new files (including via branching or renaming) which only differ in case in **some part of their path** from existing files.

This avoids issues in these scenarios:

- for a case insensitive server:
  - Windows clients (case insensitive) can accidentally create new directories or files which sync into different directories on Unix (case sensitive)
- for a case sensitive server:
  - Unix clients can create 2 (or more) files which only differ in case, and when synced to a Windows client, only one of them appears

Usage

```
"""
CaseCheckTrigger.py

This trigger ensures users are not adding new files (directly or by branching) which
only
differ in case (either filename or a directory element of their path) from existing
depot paths.
It is useful for both case-sensitive and case-insensitive servers, although most used
for the former.

Example 1: Typical usage from the Helix Core server triggers table:

    Triggers:
        CheckCaseTrigger change-submit //... "/usr/bin/python3
/p4/common/bin/triggers/CheckCaseTrigger.py %changelist% myuser=%user%"

SAMPLE OUTPUT:
    Submit validation failed -- fix problems then use 'p4 submit -c 1234'.
    'CheckCaseTrigger' validation failed:

    Your submission has been rejected because the following files
    are inconsistent in their use of case with respect to existing
    directories:

    Your file:         '//depot/dir/test'
    existing file/dir: '//depot/DIR'

BYPASS LOGIC:
    By default, this trigger can be bypassed by any user by adding the
    token BYPASS_CASE_CHECK to the changelist description.  Specify
    'allowbypass=no' on the command line to disable the ability to
    bypass this trigger

    As an exception, if the user is 'git-fusion-user', the case check
```

```
    is always bypassed if 'myuser' is defined.

DEPENDENCIES:
    This trigger requires P4Triggers.py and the P4Python API.

SEE ALSO:
    See 'p4 help triggers'.

"""
    """CaseCheckTrigger is a subclass of P4Trigger. Use this trigger to ensure
       that your depot does not contain two filenames or directories that only
       differ in case.
       Having files with different case spelling will cause problems in mixed
       environments where both case-insensitive clients like Windows and case-
       sensitive clients like UNIX access the same server.
    """
```

1.4.3. CheckChangeDesc.py

This trigger parses change descriptions to ensure that they only have the required format.

See source for an example of Python regexes.

Usage

```
"""
NAME:
    CheckChangeDesc.py

DESCRIPTION:
    Check the format of a change description on submit (change-submit)

    Can be used together with DefaultChangeDesc.py (form-out)

    To install, add a line to your Perforce triggers table like the following:

        check-change-desc change-submit //... "python
/p4/common/bin/triggers/CheckChangeDesc.py -p %serverport% -u perforce %change% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        check-change-desc change-submit //... "python
/p4/common/bin/triggers/CheckChangeDesc.py %change% "

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable, and set the configuration
variables below.
```


© 2010-2025 Perforce Software, Inc.

```
    To configure, read and modify the following lines up to the comment that
    reads "END OF CONFIGURATION BLOCK".  You may also need to modify the
    definition of which fields constitute a new job based on your jobspec.  This
    is in the allowed_job() function.
"""
# The error messages we give to the user
ERROR_MSG = """
First line of changelist description must be:

          OS-NNNN and some text
or
          WF-NNNN and some text

Other lines of text allowed.
"""
```

1.4.4. CheckFixes.py

Part of Section 1.4.1, “Workflow Enforcement Triggers”

This trigger is intended for use with P4DTG (Defect Tracking Replication) installations.

It will allows fixes to be added or deleted depending on the values of job fields. Thus you can control workflow and disallow fixes if jobs are in a particular state. So if the field JiraStatus is closed, then you are not allowed to add or delete a fix.

Usage

```
"""
NAME:
    CheckFixes.py

DESCRIPTION:
    This trigger is intended for use with P4DTG (Defect Tracking Replication)
installations.

    It will allows fixes to be added or deleted depending on the values of job fields.
    Thus you can control workflow and disallow fixes if jobs are in a particular
state.
    So if the field JiraStatus is closed, then you are not allowed to add or delete a
fix.

    To install, add a line to your Perforce triggers table like the following:

        check-fixes fix-add //... "python /p4/common/bin/triggers/CheckFixes.py -p
%serverport% -u perforce %change% %client% %jobs%"
        check-fixes fix-delete //... "python /p4/common/bin/triggers/CheckFixes.py -p
%serverport% -u perforce --delete %change% %client% %jobs%"
```

© 2010-2025 Perforce Software, Inc.

```
    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        check-fixes fix-add //... "python /p4/common/bin/triggers/CheckFixes.py
%change% %client% %jobs%"
        check-fixes fix-delete //... "python /p4/common/bin/triggers/CheckFixes.py
--delete %change% %client% %jobs%"

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable.
"""
```

1.4.5. CheckFixes.yaml

Sample YAML config file for Section 1.4.4, “CheckFixes.py”

1.4.6. CheckFolderStructure.py

Part of Section 1.4.1, “Workflow Enforcement Triggers”

This trigger will ensure that any attempt to add (or branch) a file does not create a new folder name at specific levels.

With new streams protections options in p4d 2020.1 this can be removed.

Usage

```
"""
NAME:
    CheckFolderStructure.py

DESCRIPTION:
    This trigger is a change-submit trigger for workflow enforcement using
Workflow.yaml for streams.

    It will ensure that any attempt to add (or branch) a file does not create
    a new folder name at specific levels.

    As with other Workflow triggers, it looks for a project matching files in the
change.
    The value of "new_folder_allowed_level" is an integer and controls at which level
within a stream
    new folder names can be created.

    If not set, or set to 0, then new folders can be created in the root of the
stream.
    If set to 1, then for stream
        //streams/main/new_folder/a.txt would not be allowed, but
```


© 2010-2025 Perforce Software, Inc.

```
        //streams/main/existing_folder/new_folder/a.txt would be allowed.
    Note that files can be added just not folders.

    The field "new_folder_exceptions" - an array of users/groups who can override the
trigger.

    To install, add a line to your Perforce triggers table like the following:

        check-folder-structure change-submit //... "python3
/p4/common/bin/triggers/CheckFolderStructure.py -p %serverport% -u perforce %change%"

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        check-folder-structure change-submit //... "python3
/p4/common/bin/triggers/CheckFolderStructure.py %change%"

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable.
"""
```

1.4.7. CheckJobEditTrigger.py

For enforcement of job editing with P4DTG usage (e.g. JIRA).

Usage

```
"""
NAME:
    CheckJobEditTrigger.py

DESCRIPTION:
    This trigger is intended for use with P4DTG (Defect Tracking Replication)
installations.

    It can (optionally as configured):

     1.  Prevent creation of new jobs in Perforce by anyone other than the
         replication user.
     2   Prevent modification of read-only fields of jobs in Perforce by anyone
         other than the replication user.
     3.  Create newly mirrored jobs using the same name as that in the defect
         tracker.

    To install, add a line to your Perforce triggers table like the following:

        job_save_check form-in job "python
/p4/common/bin/triggers/CheckJobEditTrigger.py -p %serverport% -u perforce %user%
```

```
%formfile% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        job_save_check form-in job "python
/p4/common/bin/triggers/CheckJobEditTrigger.py %user% %formfile% "

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable, and set the configuration
variables below.

    To configure, read and modify the following lines up to the comment that
    reads "END OF CONFIGURATION BLOCK".  You may also need to modify the
    definition of which fields constitute a new job based on your jobspec.  This
    is in the allowed_job() function.
"""
# CONFIGURATION BLOCK

# The error messages we give to the user
MSG_CANT_CREATE_JOBS = """

You are not allowed to create new jobs!
Please create or modify the JIRA issue and let changes be replicated.
"""
MSG_CANT_CHANGE_FIELDS = """

You have changed one or more read-only job fields:
"""


# The list of writeable fields that users can change.
# Changes to any other fields are rejected.
# Please validate this against your jobspec.
WRITEABLE_FIELDS = ["Status", "Date"]

# Replicator user - this user is allowed to change fields
REPLICATOR_USER = "p4dtg"
JIRA_USER = "jira"


# END OF CONFIGURATION BLOCK
```

1.4.8. CheckStreamNameFormat.py

Usage

```
"""
NAME:
```


© 2010-2025 Perforce Software, Inc.

```
    CheckStreamNameFormat.py

DESCRIPTION:
    Trigger to only require stream names to follow a regex (on 'form-save' event)

    To install, add a line to your Perforce triggers table like the following:

        check-stream-name form-save stream "python
/p4/common/bin/triggers/CheckStreamNameFormat.py -c config.yaml -p %serverport% -u
perforce %formfile% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        check-stream-name form-save stream "python
/p4/common/bin/triggers/CheckStreamNameFormat.py -c /p4/common/config/Workflow.yaml
%formfile% "

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable, and setup the YAML configuration
file (specified by -c parameter above):

# ----------------------------------------
# config.yaml

# msg_new_folder_not_allowed: An array of lines for the message
# For legibility it is good to have the first line blank
msg_invalid_stream_name:
  - ""
  - "The format of your stream name is not valid."
  - "Only the following formats are allowed:"

# valid_stream_name_formats: An array of python regexes to apply to stream
# names. Quote all values. ".*" will allow all names.
valid_stream_name_formats:
  - "//streams/(dev|rel|main)_.*"
  - "//morestreams/(teamA|teamB)/(dev)_.*"

"""
```

1.4.9. CheckSubmitHasReview.py

Part of Section 1.4.1, “Workflow Enforcement Triggers”

Since Swarm 2019.1 this trigger is no longer required since it can be replaced with Swarm’s Workflow functionality.

© 2010-2025 Perforce Software, Inc.

Usage

```
"""
NAME:
    CheckSubmitHasReview.py

DESCRIPTION:
    This trigger is intended for use with Swarm installations.

    It will make sure that any changelist being submitted has an associated Swarm
review.
    Note it is possible to configure a list of users who are allowed to bypass the
trigger,
    e.g. a jenkins user who submits after building. See Workflow.yaml

    To install, add a line to your Perforce triggers table like the following:

        create_swarm_review change-submit //... "python
/p4/common/bin/triggers/CheckSubmitHasReview.py -c /p4/common/config/Workflow.yaml -p
%serverport% -u perforce %change% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        create_swarm_review change-submit //... "python
/p4/common/bin/triggers/CheckSubmitHasReview.py -c /p4/common/config/Workflow.yaml
%change% "

    Note that -c/--config specifies a yaml file - see Workflow.yaml for example.

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable.
"""
```

1.4.10. ControlStreamCreation.py

Usage

```
"""
NAME:
    ControlStreamCreation.py

DESCRIPTION:
    Trigger to only allow certain users/groups to create streams (form-save)

    To install, add a line to your Perforce triggers table like the following:

        control-stream-create form-save stream "python
```

 © 2010-2025 Perforce Software, Inc.

```
/p4/common/bin/triggers/ControlStreamCreation.py -c config.yaml -p %serverport% -u
perforce %user% %formfile% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        control-stream-create form-save stream "python
/p4/common/bin/triggers/ControlStreamCreation.py -c /p4/common/config/Workflow.yaml
%user% %formfile% "

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable, and setup the YAML configuration
file (specified by -c parameter above):

# ----------------------------------------
# config.yaml

# msg_new_folder_not_allowed: An array of lines for the message
# For legibility it is good to have the first line blank
msg_cant_create_stream:
  - ""
  - "You are not allowed to create new streams."
  - "Check with the admins to be added to an appropriate group."

# can_create_streams: An array of projects with user or group names who are allowed
# to create streams matching path regexes.
#   name: Project name
#   stream_paths: An array of python regex patterns
#       to which the check is applied - should be quoted. Use ".*" to match everything
can_create_streams:
  - name:       Name of this project
    stream_paths:
      - "//my_streams_depot/.*"
    allowed_users_groups:
      - user1
      - group1
      - group2

  - name:       Project A
    stream_paths:
      - "//depot2/.*"
    allowed_users_groups:
      - user2
      - group3

"""
```

1.4.11. CreateSwarmReview.py

Part of Section 1.4.1, “Workflow Enforcement Triggers”

This trigger creates Swarm reviews automatically for committed changes.

It is deprecated since Swarm now supports this as part of its workflow.

Usage

```
"""
NAME:
    CreateSwarmReview.py

DESCRIPTION:
    This trigger is intended for use with Swarm installations

    It will search for any jobs associated with the changelist and find any reviews
    associated with that job.
    If found it will update the review with this changelist
    Otherwise it will create a new Swarm review with a template description
    and with configurable reviewers as requested.

    To install, add a line to your Perforce triggers table like the following:

        create_swarm_review change-commit //... "python
/p4/common/bin/triggers/CreateSwarmReview.py -c Workflow.yaml -p %serverport% -u
perforce %change% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        create_swarm_review change-commit //... "python
/p4/common/bin/triggers/CreateSwarmReview.py -c Workflow.yaml %change% "

    Note that -c/--config specifies a yaml file - see Workflow.yaml for example.

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable.
"""
```

1.4.12. DefaultChangeDesc.py

Creates a default changelist description.

Usage

```
"""
NAME:
```

 © 2010-2025 Perforce Software, Inc.

```
    DefaultChangeDesc.py

DESCRIPTION:
    Create a default change description (form-out)

    Can be used together with CheckChangeDesc.py (change-submit)

    To install, add a line to your Perforce triggers table like the following:

        default-change-desc form-out change "python
/p4/common/bin/triggers/DefaultChangeDesc.py -p %serverport% -u perforce %user%
%formfile% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        default-change-desc form-out change "python
/p4/common/bin/triggers/DefaultChangeDesc.py %user% %formfile% "

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable, and set the configuration
variables below.

    To configure, read and modify the following lines up to the comment that
    reads "END OF CONFIGURATION BLOCK".  You may also need to modify the
    definition of which fields constitute a new job based on your jobspec.  This
    is in the allowed_job() function.
"""
# CONFIGURATION BLOCK

# The error messages we give to the user
DEFAULT_CHANGE_DESC = """OS-NNNN/WF-NNNN <Description of change>

# Please choose either OS or WF JIRA issue ID and your description
# Description may be multi line.
# Comment lines like these beginning with '#' can be edited out or will
# be removed automatically when change is submitted.
# The description will be validated before you save.
"""

# END OF CONFIGURATION BLOCK
```

1.4.13. DefaultSwarmReviewDesc.py

Updates Swarm review changelist with suitable template text. This can encourage users to record information against the review, or use a checklist.

Part of Section 1.4.1, “Workflow Enforcement Triggers”

© 2010-2025 Perforce Software, Inc.

Usage

```
"""
NAME:
    DefaultSwarmReviewDesc.py

DESCRIPTION:
    Create a default change description (form-in) for swarm user - so updates Swarm
review changelist

    Can be used together with CheckChangeDesc.py (change-submit)

    To install, add a line to your Perforce triggers table like the following:

        default-swarm-desc form-in change "python
/p4/common/bin/triggers/DefaultSwarmReviewDesc.py -p %serverport% -u perforce %user%
%formfile% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        default-swarm-desc form-in change "python
/p4/common/bin/triggers/DefaultSwarmReviewDesc.py %user% %formfile% "

    You may need to provide the full path to python executable, or edit the path to
the trigger.
"""
```

1.4.14. DefaultSwarmReviewDesc.yaml

Example YAML config file for Section 1.4.13, “DefaultSwarmReviewDesc.py”

1.4.15. JobIncrement.pl

The `JobIncrement.pl` trigger enables increment of jobs with custom name, e.g PROJX-123, even without integration with an external defect tracking system.

See: JobIncrement.html.

1.4.16. JobsCmdFilter.py

This script is designed to run as a jobs command filter trigger on the server. It ensures that multiple wildcards are not specified in any search string (which can cause the server to perform excessive processing and impact performance)

Usage:

```
jobs-cmd-filter command pre-user-jobs "/p4/common/bin/triggers/JobsCmdFilter.py
```

 © 2010-2025 Perforce Software, Inc.

```
%args%"
```

1.4.17. P4Triggers.py

Base class for many of the Python triggers.

1.4.18. PreventWsNonAscii.py

This script is designed to run as a form save trigger on the server. It will cause a workspace save to fail if any non-ascii characters are present in the workspace spec.

It also will block odd characters from the workspace name.

Usage:

```
PreventWSNonASCII form-save client "/p4/common/bin/triggers/PreventWsNonAscii.py
%formfile%"
```

1.4.19. RequireJob.py

Allows admins to require jobs to be associated with all submits in particular areas of repository.

Part of Section 1.4.1, “Workflow Enforcement Triggers”

Usage

```
"""
NAME:
    RequireJob.py

DESCRIPTION:
    This is a trigger which allows admins to require a job to be linked with any
submits.

    It takes a config file of the format for Workflow.yaml which controls
    whether this trigger fires or not (in addition to trigger table entries).

    It refers to field 'swarm_user' and allows that user, if specified, to bypass this
trigger.
    Also one or more users specified by 'submit_without_review_users'.

    To install, add a line to your Perforce triggers table like the following:

        require-job change-submit //... "python /p4/common/bin/triggers/RequireJob.py
-c /p4/common/config/Workflow.yaml -p %serverport% -u perforce %change% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):
```

© 2010-2025 Perforce Software, Inc.

```
        require-job change-submit //... "python /p4/common/bin/triggers/RequireJob.py
-c /p4/common/config/Workflow.yaml %change% "

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable.
"""
```

1.4.20. SetLabelOptions.py

This script is designed to run as a form-in and form-out trigger on the server. It sets the autoreload option for static labels and doesn't allow the user to change it.

Usage:

```
setlabelopts form-out label "/p4/common/bin/triggers/SetLabelOptions.py %formfile%"
setlabelopts form-in label "/p4/common/bin/triggers/SetLabelOptions.py %formfile%"
```

1.4.21. SwarmReviewTemplate.py

Part of Section 1.4.1, "Workflow Enforcement Triggers"

Usage

```
"""
NAME:
    SwarmReviewTemplate.py

DESCRIPTION:
    This trigger is intended for use with Swarm installations and other workflow
triggers.

    It is intended to fire as a shelve-commit trigger which only fires for the swarm
user.

    It updates the Review description with template text as appropriate.

    It will search for any jobs associated with the changelist and find any reviews
    associated with that job. If found it will update the review with this changelist

    To install, add a line to your Perforce triggers table like the following:

        swarm_template_review shelve-commit //... "python
/p4/common/bin/triggers/SwarmReviewTemplate.py -c Workflow.yaml -p %serverport% -u
perforce %change% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):
```

 © 2010-2025 Perforce Software, Inc.

```
        swarm_template_review shelve-commit //... "python
/p4/common/bin/triggers/SwarmReviewTemplate.py -c Workflow.yaml %change% "

    Note that -c/--config specifies a yaml file - see Workflow.yaml for example.

    You may need to provide the full path to python executable, or edit the path to
the trigger.

    Also, don't forget to make the file executable.
"""
```

1.4.22. TFSJobCheck.py

Simple trigger to ensure jobs always contain a particular string.

Note that Section 1.4.3, “CheckChangeDesc.py” is a more general form of this trigger.

Trigger table entry:

```
TFSJobCheck change-submit //depot/path/... "/p4/common/bin/triggers/TFSJobCheck.py
%changelist%"
```

1.4.23. ValidateContentFormat.py

This trigger is intended for file content validation as part of change-content trigger Works for YAML, XML - only checks files matching required file extensions - see below.

Particularly useful to ensure that `Workflow.yaml` file itself doesn’t get accidentally corrupted!

Part of Section 1.4.1, “Workflow Enforcement Triggers”

Usage

```
"""
NAME:
    ValidateContentFormat.py

DESCRIPTION:
    This trigger is intended for file content validation as part of change-content
trigger

    Works for YAML, XML - only checks files matching required file extensions - see
below.

    To install, add a line to your Perforce triggers table like the following:

        validate-yaml change-content //....yaml "python
/p4/common/bin/triggers/ValidateContentFormat.py -p %serverport% -u perforce --yaml
```

© 2010-2025 Perforce Software, Inc.

```
%change% "
        validate-xml  change-content //....xml  "python
/p4/common/bin/triggers/ValidateContentFormat.py -p %serverport% -u perforce --xml
%change% "

    or (if server is standard SDP and has appropriate environment defaults for P4PORT
and P4USER):

        validate-yaml change-content //....yaml "python
/p4/common/bin/triggers/ValidateContentFormat.py --yaml %change% "
        validate-xml  change-content //....xml  "python
/p4/common/bin/triggers/ValidateContentFormat.py --xml %change% "

    You may need to provide the full path to python executable, or edit the path to
the trigger.
    Also, don't forget to make the file executable.
"""
```

1.4.24. Workflow.yaml

Sample generic configuration file for use with Section 1.4.1, “Workflow Enforcement Triggers”

1.4.25. WorkflowTriggers.py

Base class for use by various Workflow triggers as per Section 1.4.1, “Workflow Enforcement Triggers”

Imports Section 1.4.17, “P4Triggers.py”

1.4.26. archive_long_name_trigger.pl

This script unconverts #@*% from ascii in archive files.

The reason for doing this is when they are expanded they can overflow the unix file path length.

Usage:

```
$file_prefix -op <operation> -rev <revision> -lbr <path/file>  < stdin
```

Trigger:

```
arch archive //... "/p4/common/bin/triggers/archive_long_name_trigger.pl -op %op%
-lbr %file% -rev %rev%"
```

1.4.27. changeid.pl

The `changeid.pl` script is a sample Perforce server trigger for use in conjunction with the Perforce

 © 2010-2025 Perforce Software, Inc.

Helix Defect Tracking Gateway (P4DTG) replication to jobs.

If you intend to use this script, it shoudl be copied from `/p4/sdp/Unsupported/Samples/triggers/changeid.pl` to `/p4/common/site/bin/triggers/changeid.pl`. The directory `/p4/common/site/bin/triggers` may need to be created first. Any modifications to the script (per below) should be made to the copied file.

This script can optionally:

1. Prevent creation of new jobs in Perforce by anyone other than the replication user.
2. Prevent modification of read-only fields of jobs in Perforce by anyone other than the replication user.
3. Create newly mirrored jobs using the same name as that in the defect tracker.

To install, add a line to your triggers table like the following (on UNIX/Linux):

```
jobId form-in job "/p4/common/site/bin/triggers/changeid.pl %user% %formfile%"
```

Make the file executable, and change the path to the Perl interpreter on the first line of the script if needed. Then set the configuration in the script itself. To configure, read and modify lines up to the comment that reads "END OF CONFIGURATION BLOCK" in the `changeid.pl` script. You may also need to modify the definition of which fields constitute a new job based on your jobspec. This is in the `allowed_job()` function.

1.4.28. command_block.py

This is command trigger to allow you to block commands from all but listed users.

Trigger table entry examples:

```
command-block command pre-user-obliterate "/p4/common/bin/triggers/command_block.py
%user%"
command-block command pre-user-(obliterate|protect$)
"/p4/common/bin/triggers/command_block.py %user%"
```

1.4.29. dictionary

This file contains dictionary translations for parsing requests and generating responses.

Used by Section 1.4.33, “rad_authcheck.py”

1.4.30. externalcopy.txt

Documents how to use externalcopy programs to transfer data between commit and edge.

© 2010-2025 Perforce Software, Inc.

1.4.31. otpauthcheck.py

This trigger will check to see if the userid is in a the LOCL_PASSWD_FILE first and authenticate with that if found. If the users isn't in the local file, it checks to see if the user is a service user, and if so, it will authenticate against just the auth server. Finally, it will check the user's password and authenticator token if the other two conditions don't match.

The trigger table entry is:

```
authtrigger auth-check auth "/p4/common/bin/triggers/vip_authcheckTrigger.py %user%
%serverport%"
```

1.4.32. otpkeygen.py

This script generates a key for a user and stores the key in the Perforce server.

Used together with Section 1.4.31, "otpauthcheck.py"

1.4.33. rad_authcheck.py

This trigger will check to see if the userid is in a the LOCL_PASSWD_FILE first and authenticate with that if found. If the users isn't in the local file, it checks to see if the user is a service user, and if so, it will authenticate against LDAP. Finally, it will check the user against the Radius server if the other two conditions don't match.

You need to install the python-pyrad package and python-six package for it to work. It also needs the file named dictionary in the same directory as the script.

Set the Radius servers in RAD_SERVERS below Set the shared secret Pass in the user name as a parameter and the password from stdin.

The trigger table entry is:

```
authtrigger auth-check auth "/p4/common/bin/triggers/rad_authcheck.py %user%
%serverport% %clientip%"
```

The script current is set such that the Perforce user names should match the RSA ID's. In the case of one customer, the RSA ID's were all numeric, so we made the Perforce usernames be realname_RSAID and had this script just strip off the realname_ part. Example commented out in the main function.

1.4.34. radtest.py

This is a radius test script. You need to install the python-pyrad package and python-six package for it to work. It also needs the file named dictionary in the same directory as the script.

Set the Radius servers in radsvrs below Set the shared secret Pass in the user name as a parameter

 © 2010-2025 Perforce Software, Inc.

and the password from stdin.

1.4.35. submit_form_1.py

This script modifies the description of the change form for the Perforce users listed in the submit_form_1_users group.

Trigger table entry:

```
submitform1 form-out change "/p4/common/bin/triggers/submit_form_1.py %formfile%
%user%"
```

1.4.36. submit_form_1_in.py

This script checks the input of the description form for the path specified in the triggers table.

Trigger table entry:

```
submitform1_in change-submit //depot/somepath/...
"/p4/common/bin/triggers/submit_form_1_in.py %changelist% %user%"
```

1.5. triggers / tests

This directory contains test harnesses for various triggers document in the section above.

They import a common base class `p4testutils.py` and then run various unit tests.

The tests are one of two types: * simple unit tests * integration tests using a real `p4d` instance (running in DVCS-style mode).

The tests make it straight forward to ensure that you haven't broken any tests.

You need to ensure you have a `p4d` executable in your PATH.

Run any individual test:

```
python3 TestRequireJob.py
```

© 2010-2025 Perforce Software, Inc.

Chapter 2. Maintenance

These are example scripts which have proven useful in the past.

They are NOT fully tested and NOT guaranteed to work, although in most cases they are fairly simple and reliable.

Treat with some caution!!!

2.1. accessdates.py

This script is normally called by another script, such as Section 2.45, “unload_clients.py”

However, if run standalone, it will generate 4 files with a list of specs that should be archived based on the number of weeks in Section 2.27, “maintenance.cfg”

The file generated are:

```
branches.txt
clients.txt
labels.txt
users.txt
```

2.2. add_users.sh

This script adds a bunch of users from a users_to_add.csv file of the form:

```
<user>,<email>,<full_name>[,<group1 group2 group3 ...]
```

This first line of the `users_to_add.csv` file is assumed to be a header and is always ignored.

```
vi users_to_add.csv
```

```
./add_users.sh 2>&1 | tee add_users.$(date +'%Y%m%d-%H%M').log
```

2.3. addusertogroup.py

This script adds a user or users to the specified group.

Usage:

```
python addusertogroup.py [instance] user group
```

 © 2010-2025 Perforce Software, Inc.

- instance defaults to 1 if not given.
- user = user_name or a file containing a list of user names, one per line.
- group = name of Perforce group to add the user(s) to.

2.4. checkusers.py

This script will generate a list of all user names that are listed in any group, but do not have a user account on the server. The results are written to `removeusersfromgroups.txt`.

Usage:

```
python checkusers.py
```

You can pass that to `removeuserfromgroups.py` to do the cleanup.

2.5. checkusers_not_in_group.py

This script will generate a list of all standard Perforce user names that are not listed in any group. It prints the results to the screen, so you may want to redirect it to a file.

Usage:

```
python checkusers_not_in_group.py
```

2.6. clean_protect.py

This script will drop all lines in the protect table that have a group referenced from the file `groups.txt` passed into the script. The list of groups to drop is passed in as the first parameter and the protect table is passed in as the 2nd parameter.

Usage:

```
python protect_groups.py remove_groups.txt p4.protect
```

`remove_groups.txt` is generated using Section 2.33, “protect_groups.py” - See that script for details.

Run `p4 protect -o > p4.protect` to generate the protections table.

You can redirect the output of this script to a file called `new.p4.protect` and then you can compare the original `p4.protect` and the `new.p4.protect`. If everything looks okay, you can update the protections table by running:

```
p4 protect -i < new.p4.protect
```

© 2010-2025 Perforce Software, Inc.

2.7. convert_label_to_autoreload.py

Converts label and sets `autoreload` option see Command Reference

Usage:

```
convert_label_to_autoreload.py <label or file_with_list_of_labels>
```

2.8. convert_rcs_to_unix.sh

Executes command which converts Windows RCS files with CRLF endings to Unix LF endings:

```
find . -type f -name '*,v' -print -exec perl -p -i -e 's/\r\n/\n/' {} \;
```

2.9. countrevs.py

This script will count the total number of files and revisions from a list of files in the Perforce server.

Usage:

```
p4 files //... > files.txt
```

then run:

```
python countrevs.py files.txt
```

2.10. creategroups.py

This script creates groups on the server based on the entries in a local file called `groups.txt` which contains lines in this format:

```
group,groupname1
username1
username2
:
group,groupname2
username2
username3
```

Run:

© 2010-2025 Perforce Software, Inc.

```
python creategroups [instance]
```

Instance defaults to 1 if not given.

2.11. createusers.py

This script will create a group of users all at once based on an input file. The input file should contain the users in the following format, one per line:

```
user,email,fullname
```

Run

```
python createusers.py userlist.csv <instance>
```

Instance defaults to 1 if not passed in.

2.12. del_shelve.py

This script will delete shelves and clients that have not been accessed in the number of weeks defined by the variable weeks in `maintenance.cfg`.

Run the script as:

```
del_shelve.py [instance]
```

If no instance is given, it defaults to 1.

2.13. delusers.py

Calls [_p4deleteusers_py] automain module which will remove allusers that haven't accessed Perforce in the variable userweeks set in `maintenance.cfg`

Usage:

```
delusers.py
```

2.14. edge_maintenance

Runs regular tasks such as:

- removes server.locks dir

© 2010-2025 Perforce Software, Inc.

- unloading clients

2.15. email.bat

This script is used to send email to all of your Perforce users.

Create a file called `message.txt` that contains the body of your message. Then call this script and pass the subject in quotes as the only parameter.

It makes a copy of the previous email list, then call make_email_list.py to generate a new one from Perforce.

The reason for making the copy is so that you will always have an email list that you can use to send email with. Just comment out the call to `python make_email_list.py` in the script and run it. It will use the current list to send email from. This is handy in case your server goes off-line.

2.16. email.sh

Unix version of Section 2.15, “email.bat”

2.17. email_pending_client_deletes.py

This script will email users that their clients haven't been accessed in the number of weeks defined in the weeks variable (of `maintenance.cfg`), and warn them that it will be deleted in one week if they do not use it.

2.18. email_pending_user_deletes.py

This script will email users that their user accounts haven't been accessed in the number of weeks defined in the weeks variable (of `maintenance.cfg`), and warn them that it will be deleted in one week if they do not use it.

2.19. EvilTwinDetector.sh

Detects "evil twins".

Usage

```
   echo "USAGE for $THISSCRIPT v$Version:

$THISSCRIPT [-i <instance>] { -d <stream_depot> | -s //source/stream } [-w <work_dir>]
[-f] [-L <log>] [-si] [-v<n>] [-D]

or

$THISSCRIPT [-h|-man|-V]
"
   if [[ $style == -man ]]; then
```

 © 2010-2025 Perforce Software, Inc.

```
      echo -e "
DESCRIPTION:
    Detect and optionally fix \"evil twins\" in a given stream depot.

    The term \"evil twin\" has origins in the ClearCase version control
    system, the first to have truly sophisticated branching and merging
    capability.  With this sophistication came complexity, and an evil
    twin is one such complexity.  An evil twin can occur in any
    version control system with sophisticated branching and merging
    (Perforce, Git, ClearCase, AccuRev, etc.).

    In Perforce, an evil twin occurs when two files with the same name
    relative to the root of a stream are created in both streams with
    'p4 add', rather than the preferred workflow of creating src/foo.c
    in one stream and branching it into another.  When a file is created
    with 'p4 add' twice rather than the preferred 'add then branch' flow,
    there is no integration history connecting the files.  This creates
    a problem when merging across streams, as their is no 'base' or
    common ancestor that is needed to calculate merge results.

    In ClearCase, this was a bad situation indeed - one such file needed
    to be designated as the \"evil\" twin, and that file and its history
    would need to be obliterated ('rmelem' in ClearCase parlance). In
    Perforce, the situation is not nearly so bad.  Rather than designating
    one of the twins as evil, we simply do a 'baseless integration' and
    establish a family history, after which point the two twins can have
    a happy family reunion.

    Note that many ClearClase sites deployed an Evil Twin prevention
    trigger that could prevent evil twins from occurring in the first
    place, and deploying such a trigger is a best practice in
    ClearCase environments.  There is a similar trigger for Perforce,
    but it is not commonly deployed, since evil twins aren't nearly
    as problematic in Perforce.  They don't occur often.

    At a high level:
    * Evil twin detection means doing an integrate preview, and checking
    for the \"can't integrate without -i flag\" warning.  The -i flag is
    for baseless integrations, essentially evil twins.
    * Evil twin correction means doing an integrate with the -i flag on
    the individual files, and then doing a 'p4 resolve -ay' (accept
    \"yours,\" i.e. accept target), and submitting. That creates an
    integration record, visible as a merge arrow in P4V.

    This solution using 'p4 resolve -ay' is the best choice for mass
    detection and resolution of evil twins.  The assumption here is that
    the contents are already correct, and you only want to draw merge
    arrows and avoid changing content in the processing.

    For evil twins that occur organically in natural development, e.g.
    due to developer error or miscoordination among developers working
```

```
    in different streams, you might choose to resolve with -ay or -at
    (accept \"theirs,\" i.e. accept source).  Natural evil twins are
    uncommon and occur typically only for individual files.  (And if they
    are common, that could be a sign of a larger process/communication
    issue among teams).

    === WARNING WARNING WARING ===
    This script may require a large amount of temporary storage, as it
    creates a large number of stream workspaces and does a sync in them.
    See the '-w' flag.

OPTIONS:
 -i <instance>
    Specify the SDP instance name.

    This is required unless the SDP environment has previously been loaded
    in the current shell, i.e. by sourcing the $P4CBIN/p4_vars
    file and specifying the instance, e.g. :

    source $P4CBIN 1

 -d <stream_sdepot>
    Specify a stream depot to process, e.g. -d fgs to process the //fgs stream depot.

    This argument is required unless '-s //source/stream' is specified,
    in which case the stream depot is inferred from the stream path.

 -s //source/stream
    Specify a particular source stream to search.  The specified stream
    and its parent will be checked for evil twins.

 -w <work_dir>
    Specify a working directory, used for temporary workspace storage,
    etc. The default is \"/scratch\".

    === WARNING WARNING WARING ===
    This script may require a large amount of temporary storage, as it
    creates a large number of stream workspaces and does a sync in them.

    This flag controls the scratch storage area used by this script.

    The workspace storage directory will be in an \"etd\" directory
    below the specified <work_dir>  So by default, workspace root
    directories will appear one directory level below \"/scratch/etd\".
    This script will attempt to create the \"etd\" directory under
    specified <work_dir> if it does not already exist, and abort
    if it cannot be created.


 -f Fix evil twins with forced integrates doing baseless merges and
    resolving with '-ay', effectively drawing merge arrows to establish
```


© 2010-2025 Perforce Software, Inc.

```
    a branching relationship, but not affecting target file content.

    By default, a report shows the commands that would be run, but they
    are not executed.

 -v<n>    Set verbosity 1-5 (-v1 = quiet, -v5 = highest).

 -L <log>
    Specify the path to a log file, or the special value 'off' to disable
    logging.  All output (stdout and stderr) are captured in the log.

    NOTE: This script is self-logging.  That is, output displayed on the screen
    is simultaneously captured in the log file.  Do not run this script with
    redirection operators like '> log' or '2>&1', and do not use 'tee.'

-si Operate silently.  All output (stdout and stderr) is redirected to the log
    only; no output appears on the terminal.  This cannot be used with
    '-L off'.

 -D     Set extreme debugging verbosity.

HELP OPTIONS:
 -h Display short help message
 -man   Display man-style help message
 -V Display version info for this script and its libraries.

EXAMPLES:
    First, setup environment:
    cd $P4CBIN
    source ./p4_vars 1

    Then run as per the following examples, suitable for a long-running     script.

    Example 1: Search for all Evil Twins in the fgs depot:
    nohup ./$THISSCRIPT -i 1 -w /big/scratch -d fgs < /dev/null > /tmp/etd.log 2>&1 &
    tailf \$(ls -t \$LOGS/${THISSCRIPT%.sh}.*.log|head -1)

    Example 2: Find and fix all Evil Twins in the fgs depot:
    nohup ./$THISSCRIPT -i 1 -w /big/scratch -d fgs -f < /dev/null > /tmp/etd.log 2>&1
&
    tailf \$(ls -t \$LOGS/${THISSCRIPT%.sh}.*.log|head -1)

    Example 3: Find and fix all Evil twins between //fgs/dev and its parent.
    nohup ./$THISSCRIPT -i 1 -w /big/scratch -s //fgs/dev < /dev/null > /tmp/etd.log
2>&1 &
    tailf \$(ls -t \$LOGS/${THISSCRIPT%.sh}.*.log|head -1)
"
```

2.20. group_audit.py

This script emails the owners of each group with instructions on how to validate the membership of the groups they own.

2.21. isitalabel.py

Determine if a label exists in Perforce.

Usage:

```
isitalabel.py labelname
```

Program will print out results of the search. Sets ISALABEL in the environment to 0 if found, 1 if not found. Also will exit with errorlevel 1 if the label is not found

2.22. license_status_check.sh

Determines how many days/hours etc are remaining for a license.

Usage:

```
license_status_check.sh <instance>
```

2.23. lowercp.py

This script will make a lowercase copy of the source folder, and report any conflicts found during the copy. Run this script from the source path adjust the target path below.

Pass in the folder to be copied in lower case to the target.

ie: To copy `/p4/1/depots/depot` to `/depotdata2/p4/1/depots`, run:

```
cd /p4/1/depots
lowercp.py depot
```

2.24. lowertree.py

This script is used to rename a tree to all lowercase. It is helpful if you are trying to convert your server from case sensitive to case insensitive.

It takes the directory to convert as the first parameter.

© 2010-2025 Perforce Software, Inc.

2.25. maintain_user_from_groups.py

Usage:

```
maintain_user_froup_groups.py [instance]
```

Defaults to instance 1 if parameter not given.

What this scripts does:

```
Reads users from groups
Creates any missing user accounts
Removes accounts that are not in the group
```

2.26. maintenance

This is an example maintenance script to run the recommended maintenance scripts on a weekly basis. You need to make sure you update the hard coded locations to match yours.

2.27. maintenance.cfg

Created from Section 2.44, “template.maintenance.cfg”

2.28. make_email_list.py

This script creates a list of email address for your users directly from their Perforce user accounts. It is intended to be used as part of email.bat, but can be used with any mail program that can read addresses from a list.

This can be replaced by a single command using recent options for p4 CLI:

```
p4 -F "%email%|%user%" users
```

2.29. mirroraccess.py

This script will add a user to all the groups of another user in Perforce.

Usage:

```
python mirroraccess.py instance user1 user2 <user3> <user4> ... <userN>
```

- user1 = user to mirror access from.
- user2 = user to mirror access to.

© 2010-2025 Perforce Software, Inc.

- <user3> ... <userN> = additional users to mirror access to.

2.30. p4deleteuser.py

Removes user and any clients/shelves owned by that user.

```
Note: This script will pull all the servers addresses off the servers output. In order
for that to work, you must add the Address field on each server form.
```

```
Usage:
```

```
p4deleteuser.py [instance] user_to_remove
p4deleteuser.py [instance] file_with_users_to_remove
```

2.31. p4lock.py

This script locks a perforce label.

The only command line parameter is the label

```
p4lock.py LABEL
```

See Section 2.32, “p4unlock.py”

2.32. p4unlock.py

This script unlocks a perforce label.

The only command line parameter is the label

```
p4unlock.py labelname
```

2.33. protect_groups.py

This script will list all the groups mentioned in "p4 protect" that are not a Perforce group.

You need to pull the groups from p4 protect with:

```
p4 protect -o | grep group |  cut -d " " -f 3 | sort | uniq > protect_groups.txt and
pass protect_groups.txt to this script.
```


© 2010-2025 Perforce Software, Inc.

Usage:

```
python protect_groups.py [instance] protect_groups.txt > remove_groups.txt
```

- instance defaults to 1 if not given.

2.34. proxysearch.py

This script will search your server log file and find any proxy servers that are connecting to the server. The server log file needs to be set to a debug level 1 or higher in order to record all the commands coming into the server.

Just pass the log file in as parameter to this script and it will print out a list of the proxy servers.

2.35. pymail.py

Reads values from `maintenance.cfg` for things like `mailhost`. Sends email according to parameters below.

Usage:

```
pymail.py -t <to-address or address file> -s <subject> -i <input file>
```

2.36. remove_empty_pending_changes.py

This script will remove all of the empty pending changelists on the server.

Usage:

```
remove_empty_pending_changes.py
```

2.37. remove_jobs.py

This script will remove all of the fixes associated with jobs and then delete the jobs listed in the file passed as the first argument. The list can be created with `p4 jobs > jobs.txt`. The script will handles the extra text in the lines.

Usage:

```
remove_jobs.py [SDP_INSTANCE] list_of_jobs_to_remove
```

© 2010-2025 Perforce Software, Inc.

2.38. removeuserfromgroups.py

This script will look for the specified users in all groups and remove them.

Usage:

```
removeuserfromgroups.py [instance] USER
removeuserfromgroups.py [instance] FILE
```

- USER can be a single username or, it can be a FILE with a list of users.
- instance defaults to 1 if not given.

2.39. removeusersfromgroup.py

This script will remove the specified users from given group.

Usage:

```
removeusersfromgroup.py [instance] USER groupname
removeusersfromgroup.py [instance] FILE groupname
```

- USER can be a single username or, it can be a FILE with a list of users.
- instance defaults to 1 if not given.

2.40. sample_cron_entries.txt

These is a sample crontab entry to call the `maintenance` script from cron.

2.41. sdputils.py

This module is a library for other modules in this directory and is imported by some of them.

2.42. server_status.sh

This script is used to check the status of all the instances running on this machine. It can handle named and numbered instances.

2.43. setpass.py

This script will set the password for a user to the value set in the password variable in the main function.

The name of the user to set the password for is passed as a parameter to the file.

© 2010-2025 Perforce Software, Inc.

Usage:

```
python setpass.py [instance] user
```

- instance defaults to 1 if not given.

2.44. template.maintenance.cfg

Example for editing and copying to `maintenance.cfg`.

Read by many of the scripts.

Values include things like default number of weeks to use to detect **old** users/clients/labels.

2.45. unload_clients.py

This script will unload clients not accessed since the weeks parameter specified in `maintenance.cfg`.

2.46. unload_clients_with_delete.py

This script will unload clients that have not been accessed since the number of weeks specified in the `maintenance.cfg` file.

This version of unload clients will delete clients with exclusive checkouts since unload will not unload those clients.

It also deletes shelves from the clients to be deleted since delete will not delete a client with a shelf.

2.47. unload_labels.py

This script will unload labels that have not been accessed since the number of weeks specified in the `maintenance.cfg` entry.

© 2010-2025 Perforce Software, Inc.