Perforce Helix Core Server Deployment Package (for UNIX/Linux)

Perforce Professional Services

Version v2023.1, 2023-07-11

Table of Contents

Preface

The Server Deployment Package (SDP) is the implementation of Perforce's recommendations for operating and managing a production Perforce Helix Core Version Control System. It is intended to provide the Helix Core administration team with tools to help:

- Simplify Management
- Simplify Upgrades
- High Availability (HA)
- Disaster Recovery (DR)
- Fast and Safe Upgrades
- Production Focus
- Best Practice Configurables
- Optimal Performance, Data Safety, and Simplified Backup

This guide is intended to provide instructions of setting up the SDP to help provide users of Helix Core with the above benefits.

This guide assumes some familiarity with Perforce and does not duplicate the basic information in the Perforce user documentation. This document only relates to the Server Deployment Package (SDP). All other Helix Core documentation can be found here: Perforce Support Documentation.

Please Give Us Feedback

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

Chapter 1. Overview

The SDP has four main components:

- Hardware and storage layout recommendations for Perforce.
- Scripts to automate critical maintenance activities.
- Scripts to aid the setup and management of replication (including failover for DR/HA).
- Scripts to assist with routine administration tasks.

Each of these components is covered, in detail, in this guide.

1.1. Using this Guide

Chapter 2, *Setting up the SDP* describes concepts, terminology and pre-requisites

Chapter 4, *Installing the SDP on Unix / Linux* consists of what you need to know to setup Helix Core sever on a Unix platform.

Chapter 5, *Backup, Replication, and Recovery* gives information around the Backup, Restoration and Replication of Helix Core, including some guidance on planning for HA (High Availability) and DR (Disaster Recovery)

Chapter 6, *Upgrades* covers upgrades of `p4d` and related Helix Core executables.

Section 6.3, "Upgrading the SDP" covers upgrading the SDP itself.

Chapter 7, *Maximizing Server Performance* covers optimizations and proactive actions.

Chapter 8, *Tools and Scripts* covers all the scripts used within the SDP in detail.

Appendix A, *SDP Package Contents and Planning* describes the details of the SDP package.

Appendix B, *The journalPrefix Standard* describes the standard for setting the `journalPrefix` configurable.

Appendix C, *Server Spec Naming Standard* describes the standard for naming 'server' specs created with the `p4 server` command.

Appendix D, *Frequently Asked Questions* and Appendix E, *Troubleshooting Guide* are useful for other questions.

Appendix F, *Starting and Stopping Services* gives on overview of starting and stopping services with common init mechanisms, `systemd` and SysV.

1.2. Getting the SDP

The SDP is downloaded as a single zipped tar file the latest version can be found at: https://swarm.workshop.perforce.com/projects/perforce-software-sdp/files/downloads

The file to download containing the latest SDP is consistently named `sdp.Unix.tgz`. A copy of this file also exists with a version-identifying name, e.g. `sdp.Unix.2021.2.28649.tgz`.

The direct download link to use with `curl` or `wget` is illustrated with this command:

```
curl -L -O https://swarm.workshop.perforce.com/projects/perforce-software-
sdp/download/downloads/sdp.Unix.tgz
```

1.3. Checking the SDP Version

Once installed, the SDP `Version` file exists as `/p4/sdp/Version`. This is a simple text file that contains the SDP version string. The version can be checked using a command like `cat`, as in this sample command:

```
$ cat /p4/sdp/Version
Rev. SDP/MultiArch/2020.1/27955 (2021/08/13)
```

That string can be found in Change History section of the SDP Release Notes. This can be useful in determining if your SDP is the latest available, and to see what features are included.

When an SDP tarball is extracted, the `Version` file appears in the top-level `sdp` directory.

Chapter 2. Setting up the SDP

This section tells you how to configure the SDP to setup a new Helix Core server.

The SDP can be installed on multiple server machines, and each server machine can host one or more Helix Core server instances. See Section 2.1, “Terminology Definitions” for detailed definition of terms.

The SDP implements a standard logical directory structure which can be implemented flexibly on one or many physical server machines.

Additional relevant information is available in the System Administrator Guide.

2.1. Terminology Definitions

- **process** - a running process with a process identifier (PID). It should normally be qualified as to what type of process it is:
  - **p4d process** - a running p4d process with it’s own copy of db.* files. P4D processes may be of any one of the standard types, e.g. standard or commit-server, and any of the valid replica types: standby, forwarding-replica, edge-server etc.
  - **p4p process** – proxy instance talking to a single upstream p4d instance
  - **p4broker process** – p4broker talking to a single upstream p4d instance
- **Instance** - a logically independent set of Helix Core data and metadata, represented by entities such as changelist numbers and depot paths, and existing a storage device in the form of db.* files (metadata) and versioned files (archive files). Thus, the instance is a reference to the logical data set, with its set of users, files, changelists.
  - The default SDP instance name is simply `1` (the digit 'one').
  - Any alphanumeric name can be used. It is mainly of interest to administrators, not regular users.
  - Instance names are best kept short, as they are typed often in various admin operational tasks.
  - An **instance** has a well defined name, embedded in its P4ROOT value. If the P4ROOT is `/p4/ace/root`, for example, `ace` is the instance name.
  - An **instance** must operate with at least one p4d process on a master server machine. The instance may also extend to many machines running additional p4d, p4broker, and p4p processes. For the additional p4d processes, they can be replicas of various types, to include standby, edge, and filtered forwarding replicas (to name a few).
  - On all machines on which an instance is physically extended, including proxy, broker, and replica machines, the instance exists as `/p4/N`, where `N` is the instance name.
  - There can be more than one instance a machine.
- **Server machine** - this is a host machine (virtual or physical) with operating system and on which any number of p4d or other processes may be running.

- **Server spec** or **server specification** - is the entity managed using `p4 server` command (and the companion `p4 servers` to list all of them).
- **Server** - this is a vague term. It needs to be fully qualified, and use on its own (unadorned) depends on context. It may mean any one of:
  - Server machine
  - P4d process (this is usually the most common usage - tend to assume this unless otherwise defined.)
  - Any other type of instance!

Thus "p4d server" is unclear as to whether you are talking about a p4d process or a server machine or a combination of both (since there may be a single instance on a single machine, or many instances on a machine, etc). Make sure you understand what is being referred to!

Chapter 3. Pre-Requisites

1. The Helix Core binaries (p4d, p4, p4broker, p4p) have been downloaded (see Chapter 4, *Installing the SDP on Unix / Linux*)
2. *sudo* access is required
3. System administrator available for configuration of drives / volumes (especially if on network or SAN or similar)
4. Supported Linux version, currently these versions are fully supported - for other versions please speak with Perforce Support.
    - Ubuntu 18.04 LTS (bionic)
    - Ubuntu 20.04 LTS (focal)
    - Red Hat Enterprise Linux (RHEL) 7.x
    - Red Hat Enterprise Linux (RHEL) 8.x
    - CentOS 7
    - CentOS 8 (not recommended for production; Rocky Linux replaces CentOS 8)
    - Rocky Linux 8.x
    - SUSE Linux Enterprise Server 12

3.1. Volume Layout and Hardware

As can be expected from a version control system, good disk (storage) management is key to maximizing data integrity and performance. Perforce recommend using multiple physical volumes for **each** p4d server instance. Using three or four volumes per instance reduces the chance of hardware failure affecting more than one instance. When naming volumes and directories the SDP assumes the "hx" prefix is used to indicate Helix volumes. Your own naming conventions/standards can be used instead, though this is discouraged as it will create inconsistency with documentation. For optimal performance on UNIX machines, the XFS file system is recommended, but not mandated. The EXT4 filesystem is also considered proven and widely used.

- **Depot data, archive files, scripts, and checkpoints**: Use a large volume, with RAID 6 on its own controller with a standard amount of cache or a SAN or NAS volume (NFS access is fine).

This volume is the only volume that **must** be backed up. The SDP backup scripts place the metadata snapshots on this volume.

+ This volume is normally called `/hxdepots`.

- **Perforce metadata (database files), 1 or 2 volumes:** Use the fastest volume possible, ideally SSD or RAID 1+0 on a dedicated controller with the maximum cache available on it. Typically a single volume is used, `/hxmetadata`. In some sites with exceptionally large metadata, 2 volumes are used for metadata, `/hxmetadata` and `/hxmetadata2`. Exceptionally large in this case means the metadata size on disk is such that (2x(size of db.* files)+room for growth) approaches or exceeds the storage capacity of the storage device used for metadata. That's driven by how big

/hxmetadata volume. So if you have a 16T storage volume and your total size of db.* files is some ~7T or less (so ~14T total), that's probably a reasonable cutoff for the definition of "exceptionally large" in this context.

Do not run anti-virus tools or back up tools against the `hxmetadata` volume(s) or `hxlogs` volume(s), because they can interfere with the operation of the Perforce server executable.

- **Journals and logs:** a fast volume, ideally SSD or RAID 1+0 on its own controller with the standard amount of cache on it. This volume is normally called `/hxlogs` and can optionally be backed up.

  If a separate logs volume is not available, put the logs on the `/hxmetadata` or `/hxmetadata1` volume, as metadata and logs have similar performance needs that differ from `/hxdepots`.

Storing metadata and logs on the same volume is discouraged, since the redundancy benefit of the P4JOURNAL (stored on `/hxlogs`) is greatly reduced if P4JOURNAL is on the same volume as the metadata in the P4ROOT directory.

If multiple controllers are not available, put the `/hxlogs` and `/hxdepots` volumes on the same controller.

On all SDP machines, a `/p4` directory will exist containing a subdirectory for each instance, and each instance named `/p4`. The volume layout is shown in Appendix A, *SDP Package Contents and Planning*. This `/p4` directory enables easy access to the different parts of the file system for each instance.

For example:

- `/p4/1/root` contains the database files for instance `1`
- `/p4/1/logs` contains the log files for instance `1`
- `/p4/1/bin` contains the binaries and scripts for instance `1`
- `/p4/common/bin` contains the binaries and scripts common to all instances

Chapter 4. Installing the SDP on Unix / Linux

4.1. Automated Install

If you are doing a "green field" install, a first-time installation on a new machine that does not yet have any Perforce Helix data, then the Helix Installer should be used.

4.2. Manual Install

The following documentation covers internal details of how the SDP can be deployed manually. Many of the steps below are performed by the Helix Installer.

To install Perforce Helix Core server and the SDP, perform the steps laid out below:

- Set up a user account, file system, and configuration scripts.
- Run the configuration script.
- Start the p4d process and configure the required file structure for the SDP.

1. If it doesn't already exist, create a group called `perforce`:

```
sudo groupadd perforce
```

2. Create a user called perforce and set the user's home directory to `/p4` on a local disk.

```
sudo useradd -d /home/perforce -s /bin/bash -m perforce -g perforce
```

3. Allow the perforce user sudo access - Option 1 (full sudo)

```
sudo touch /etc/sudoers.d/perforce
sudo chmod 0600 /etc/sudoers.d/perforce
sudo echo "perforce ALL=(ALL) NOPASSWD:ALL" > /etc/sudoers.d/perforce
sudo chmod 0400 /etc/sudoers.d/perforce
```

4. Allow the perforce user sudo access - Option 2 (limited sudo)

```
sudo touch /etc/sudoers.d/perforce
sudo chmod 0600 /etc/sudoers.d/perforce
vi /etc/sudoers.d/perforce
```

5. In the text editor, make the file look like this to give limited sudo, replacing `EDTIME_HOSTNAME` with the current machine:

```
Cmnd_Alias P4_SVC = /usr/bin/systemctl start p4d_*, \
   /usr/bin/systemctl start p4d_*, \
   /usr/bin/systemctl stop p4d_*, \
   /usr/bin/systemctl restart p4d_*, \
   /usr/bin/systemctl status p4d_*, \
   /usr/bin/systemctl cat p4d_*, \
   /usr/bin/systemctl start p4dtg_*, \
   /usr/bin/systemctl stop p4dtg_*, \
   /usr/bin/systemctl restart p4dtg_*, \
   /usr/bin/systemctl status p4dtg_*, \
   /usr/bin/systemctl cat p4dtg_*, \
   /usr/bin/systemctl start p4broker_*, \
   /usr/bin/systemctl stop p4broker_*, \
   /usr/bin/systemctl restart p4broker_*, \
   /usr/bin/systemctl status p4broker_*, \
   /usr/bin/systemctl cat p4broker_*, \
   /usr/bin/systemctl start p4p_*, \
   /usr/bin/systemctl stop p4p_*, \
   /usr/bin/systemctl restart p4p_*, \
   /usr/bin/systemctl status p4p_*, \
   /usr/bin/systemctl cat p4p_*, \
   /usr/bin/systemctl start p4prometheus*, \
   /usr/bin/systemctl stop p4prometheus*, \
   /usr/bin/systemctl restart p4prometheus*, \
   /usr/bin/systemctl status p4prometheus*, \
   /usr/bin/systemctl cat p4prometheus*, \
   /usr/bin/setcap, \
   /usr/bin/getcap
perforce  EDITME_HOSTNAME = (root) NOPASSWD: P4_SVC
```

6. Then lock down the file:

```
sudo chmod 0400 /etc/sudoers.d/perforce
```

7. Create or mount the OS server file system volumes (per layout in previous section)
   - `/hxdepots`
   - `/hxlogs`

   and either:

   - `/hxmetadata`

   or

   - `/hxmetadata1`
   - `/hxmetadata2`
8. These directories should be owned by: `perforce:perforce`

```
sudo chown -R perforce:perforce /hx*
```

9. (Optional) if you have different root directories, or are putting all files into one mounted filesystem (only recommended for small repositories), then do something like the following:

   Option 1, all under a single directory `/data`:

   ```
   cd /data
   mkdir hxmetadata hxlogs hxdepots
   sudo chown -R perforce:perforce /data/hx*
   cd /
   ln -s /data/hx* .
   sudo chown -h perforce:perforce /hx*
   ```

   Option 2, different mounted root folders, e.g. `/P4metadata`, `/P4logs`, `/P4depots`:

   ```
   sudo chown -R perforce:perforce /P4metadata /P4logs /P4depots
   ln -s /P4metadata /hxmetadata
   ln -s /P4logs /hxlogs
   ln -s /P4depots /hxdepots
   sudo chown -h perforce:perforce /hx*
   ```

10. Extract the SDP tarball.

    ```
    cd /hxdepots
    tar -xzf /WhereYouDownloaded/sdp.Unix.tgz
    ```

11. Set environment variable SDP.

    ```
    export SDP=/hxdepots/sdp
    ```

12. Make the entire $SDP (`/hxdepots/sdp`) directory writable by `perforce:perforce` with this command:

    ```
    chmod -R +w $SDP
    ```

13. Download the appropriate p4, p4d and p4broker binaries for your release and platform:

    ```
    cd /hxdepots/sdp/helix_binaries
    ./get_helix_binaries.sh
    ```

    If you want to specify a particular release, use the `-r` option as in this example specifying the

r20.2 release:

```
cd /hxdepots/sdp/helix_binaries
./get_helix_binaries.sh -r r20.2
```

4.2.1. Manual Install Initial setup

The next steps highlight the setup and configuration of a new Helix Core instance using the `mkdirs.sh` script included in the SDP.

Usage

```
USAGE for mkdirs.sh v4.9.1:

mkdirs.sh <instance> [-s <ServerID>] [-t <ServerType>] [-tp <TargetPort>] [-lp
<ListenPort>] [-I <svc>[,<svc2>]] [-MDD /bigdisk] [-MLG /jnl] [-MDB1 /db1] [-MDB2
/db2] [-f] [-p] [-test [-clean]] [-n] [-L <log>] [-d|-D]

or

mkdirs.sh [-h|-man]


DESCRIPTION:

== Overview ==

This script initializes an SDP instance on a single machine.

This script is intended to support two scenarios:

* First time SDP installation on a given machine.
* Adding new SDP instances (separate Helix Core data sets) to an existing
  SDP installation on a given machine.

And SDP instance is a single Helix Core data set, with its own unique
set of one set of users, changelist numbers, jobs, labels, versioned
files, etc. An organization may run a single instance or multiple
instances.

This is intended to be run either as root or as the operating system
user account (OSUSER) that p4d is configured to run as, typically
'perforce'.  It should be run as root for the initial install.
Subsequent additions of new instances do not require root.

== Directory Structure ==

If an initial install as done by a user other than root, various
directories must exist and be writable and owned by 'perforce' before starting:
```

```
* /p4
* /hxdepots
* /hxlogs
* /hxmetadata

The directories starting with '/hx' are configurable.

This script creates an init script in the /p4/N/bin directory.

== Crontab ==

Crontabs are generated for all server types except p4broker.

After running this script, set up the crontab based on templates
generated as /p4/common/etc/cron.d.  For convenience, a sample crontab
is generated for the current machine as /p4/p4.crontab.<SDPInstance>
(or /p4/p4.crontab.<SDPInstance>.new if the former name exists).

These files should be copied or merged into any existing files named
with this convention:

/p4/common/etc/cron.d/crontab.<osuser>.<host>

where <osuser> is the user that services run as (typically 'perforce'),
and <host> is the short hostname (as returned by a 'hostname -s' command).

REQUIRED PARAMETERS:
 <instance>
    Specify the SDP instance name to add.  This is a reference to the Perforce
    Helix Core data set.

OPTIONS:
 -s <ServerID>
    Specify the ServerID, overriding the REPLICA_ID setting in the configuration
    file.

 -S <TargetServerID>
    Specify the ServerID of the P4TARGET of the server being installed.
    Use this only when setting up an HA replica of an edge server.

 -t <ServerType>
    Specify the server type, overriding the SERVER_TYPE setting in the config
    file.  Valid values are:
    * p4d_master - A master/commit server.
    * p4d_replica - A replica with all metadata from the master (not
      filtered in any way).
    * p4d_filtered_replica - A filtered replica or filtered forwarding
      replica.
    * p4d_edge - An edge server.
    * p4d_edge_replica - Replica of an edge server. If used,
```

```
      '-S <TargetServerID>' is required.
   * p4broker - An SDP host running only a standalone p4broker, with no p4d.
   * p4proxy - An SDP host running only a standalone p4p with no p4d.

-tp <TargetPort>
   Specify the target port. Use only if ServerType is p4proxy and p4broker.

-lp <ListenPort>
   Specify the listen port. Use only if ServerType is p4proxy and p4broker.

-I [<svc>[,<svc2>]]
   Specify additional init scripts to be added to /p4/<instance>/bin
   for the instance.

   By default, the p4p service is installed only if '-t p4proxy' is
   specified, and p4dtg is never installed by default.  Valid values
   to specify are 'p4p' and 'dtg' (for the P4DTG init script).

   If services are not installed by default, they can be added later
   using templates in /p4/common/etc/init.d. Also, templates for
   systemd service files are supplied in /p4/common/etc/systemd/system.

-MDD /bigdisk
-MLG /jnl
-MDB1 /db1
-MDB2 /db2
   Specify the '-M*' to specify mount points, overriding DD/LG/DB1/DB2
   settings in the config file.  Sample:

   -MDD /bigdisk -MLG /jnl -MDB1 /fast

   If -MDB2 is not specified, it is set the the same value as -MDB1 if
   that is set, or else it defaults to the same default value as DB1.

-f Specify -f 'fast mode' to skip chown/chmod commands on depot files.
   This should only be used when you are certain the ownership and
   permissions are correct, and if you have large amounts of existing
   data for which the chown/chmod of the directory tree would be
   slow.

-p Specify '-p' to halt processing after preflight checks are complete,
   and before actual processing starts. By default, processing starts
   immediately upon successful completion of preflight checks.

-L <log>
   Specify the path to a log file, or the special value 'off' to disable
   logging.  By default, all output (stdout and stderr) goes to this file
   in the current directory:

   mkdirs.<instance>.<datestamp>.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'.

DEBUGGING OPTIONS:
 -test
    Specify '-test' to execute a simulated install to /tmp/p4 as the install
    root (rather than /p4), and with the mount point directories specified in
    the configuration file prefixed with /tmp/hxmounts, defaulting to:
    * /tmp/hxmounts/hxdepots
    * /tmp/hxmounts/hxlogs
    * /tmp/hxmounts/hxmetadata

 -clean
    Specify '-clean' with '-test' to clean up from prior test installs,
    which will result in removal of files/folders installed under /tmp/hxmounts
    and /tmp/p4.

    Do not specify '-clean' if you want to test a series of installs.

 -n No-Op.  In No-Op mode, no actions that affect data or structures are
    taken.  Instead, commands that would be run are displayed.  This is
    an alternative to -test. Unlike '-p' which stops after the preflight
    checks, with '-n' more processing logic can be exercised, with greater
    detail about what commands that would be executed without '-n'.

 -d     Increase verbosity for debugging.

 -D     Set extreme debugging verbosity, using bash '-x' mode. Also implies -d.

HELP OPTIONS:
 -h Display short help message
 -man   Display man-style help message

FILES:
    The mkdirs.sh script uses a configuration file for many settings.  A
    sample file, mkdirs.cfg, is included with the SDP.  After determining
    your SDP instance name (e.g. '1' or 'abc'), create a configuration
    file for it named mkdirs.<N>.cfg, replacing 'N' with your instance.

    Running 'mkdirs.sh N' will load configuration settings from mkdirs.N.cfg.

UPGRADING SDP:
    This script can be useful in testing and upgrading to new versions of
    the SDP, when the '-test' flag is used.

EXAMPLES:
    Example 1: Setup of first instance

    Setup of the first instance on a machine using the default instance name,
```

```
'1', executed after using sudo to become root:
$ sudo su -
$ cd /hxdepots/sdp/Server/Unix/setup
$ vi mkdirs.cfg

# Adjust settings as desired, e.g P4PORT, P4BROKERPORT, etc.

$ ./mkdirs.sh 1

A log will be generated, mkdirs.1.<timestamp>.log

Example 2: Setup of additional instance named 'abc'.

Setup a second instance on the machine, which will be a separate Helix
Core instance with its own P4ROOT, its own set of users and
changelists, and its own license file (copied from the master instance).

Note that while the first run of mkdirs.sh on a given machine should be
done as root, but subsequent instance additions should be done as the
'perforce' user (or whatever operating system user accounts Perforce
Helix services run as).

$ sudo su - perforce
$ cd /hxdepots/sdp/Server/Unix/setup
$ cp -p mkdirs.cfg mkdirs.abc.cfg
$ vi mkdirs.abc.cfg

# Adjust settings in mkdirs.abc.cfg as desired, e.g P4PORT, P4BROKERPORT, etc.

$ ./mkdirs.sh abc

A log will be generated, mkdirs.abc.<timestamp>.log

Example 3: Setup of additional instance named 'alpha' to run a standalone p4p:

$ ./mkdirs.sh alpha -t p4proxy

Example 4: Setup of a stand instance named '1' to run a standalone p4broker:

$ ./mkdirs.sh 1 -t p4broker
```

If you use a "name" for the instance (not an integer) you MUST modify the P4PORT variable in the `mkdirs.instance.cfg` file.

The instance name must map to the name of the cfg file or the default file will be used with potentially unexpected results.

Examples:

- `mkdirs.sh 1` requires `mkdirs.1.cfg`

- `mkdirs.sh ion` requires `mkdirs.ion.cfg`

3. Put the Perforce license file for the p4d server instance into `/p4/1/root`

if you have multiple instances and have been provided with port-specific licenses by Perforce, the appropriate license file must be stored in the appropriate `/p4/<instance>/root` folder.

the license file must be renamed to simply the name `license`.

Your Helix Core instance is now setup, but not running. The next steps detail how to make the Helix Core p4d instance a system service.

You are then free to start up the `p4d` instance as documented in Appendix F, *Starting and Stopping Services*.

Please note that if you have configured SSL, then refer to Section 4.2.1.1, "Use of SSL".

4.2.1.1. Use of SSL

As documented in the comments in mkdirs.cfg, if you are planning to use SSL you need to set the value of:

```
SSL_PREFIX=ssl:
```

Then you need to put certificates in `/p4/ssl` after the SDP install or you can generate a self signed certificate as follows:

Edit `/p4/ssl/config.txt` to put in the info for your company. Then run:

```
/p4/common/bin/p4master_run <instance> /p4/<instance>/bin/p4d_<instance> -Gc
```

For example using instance 1:

```
/p4/common/bin/p4master_run 1 /p4/1/bin/p4d_1 -Gc
```

In order to validate that SSL is working correctly:

```
source /p4/common/bin/p4_vars 1
```

Check that P4TRUST is appropriately set in the output of:

```
p4 set
```

Update the P4TRUST values:

```
p4 trust -y
p4 -p $P4MASTERPORT trust -y
```

Check the stored P4TRUST values:

```
p4 trust -l
```

Check you are not prompted for trust:

```
p4 login
p4 info
```

4.2.1.2. Configuration script mkdirs.cfg

The `mkdirs.sh` script executed above resides in `$SDP/Server/Unix/setup`. It sets up the basic directory structure used by the SDP. Carefully review the config file `mkdirs.instance.cfg` for this script before running it, and adjust the values of the variables as required. The important parameters are:

Parameter	Description
DB1	Name of the hxmetadata1 volume (can be same as DB2)
DB2	Name of the hxmetadata2 volume (can be same as DB1)
DD	Name of the hxdepots volume
LG	Name of the hxlogs volume
CN	Volume for /p4/common
SDP	Path to SDP distribution file tree
SHAREDDATA	TRUE or FALSE - whether sharing the /hxdepots volume with a replica - normally this is FALSE
ADMINUSER	P4USER value of a Perforce super user that operates SDP scripts, typically perforce or p4admin.
OSUSER	Operating system user that will run the Perforce instance, typically perforce.
OSGROUP	Operating system group that OSUSER belongs to, typically perforce.
CASE_SENSITIVE	Indicates if p4d server instance has special case sensitivity settings

Parameter	Description
SSL_PREFIX	Set if SSL is required so either "ssl:" or blank for no SSL
P4ADMINPASS P4SERVICEPASS	Password to use for Perforce superuser account - can be edited later in /p4/common/config/.p4password.p4_1.admin Service User's password for replication - can be edited later - same dir as above.
P4MASTERHOST	Fully qualified DNS name of the Perforce master server machine for this instance. If this p4d instance is an HA for an edge server this should refer to the DNS of the edge server machine. Otherwise replicas should refer to the commit-server machine.

For a detailed description of this config file it is fully documented with in-file comments, or see

4.2.2. SDP Init Scripts

The SDP includes templates for initialization scripts ("init scripts") that provide basic service `start` /`stop`/`status` functionality for a variety of Perforce server products, including:

- p4d
- p4broker
- p4p
- p4dtg

During initialization for an SDP instance, the SDP `mkdirs.sh` script creates a set of initialization scripts based on the templates, and writes them in the instance-specific bin folder (the "Instance Bin" directory), `/p4/N/bin`. For example, the `/p4/1/bin` folder for instance `1` might contain any of the following:

```
p4d_1_init
p4broker_1_init
p4p_1_init
p4dtg_1_init
```

The set of `*_init` files in the Instance Bin directory defines which services (p4d, p4broker, p4p, and/or p4dtg) are active for the given instance on the current machine. A common configuration is to run both p4d and p4broker together, or only run a p4p on a machine. Unused init scripts must be removed from the Instance Bin dir. For example, if a p4p is not needed for instance 1 on the current machine, then `/p4/1/bin/p4p_1_init` should be removed.

For example, the init script for starting p4d for instance 1 is `/p4/1/bin/p4d_1_init`. All init scripts

accept at least `start`, `stop`, and `status` arguments. How the init scripts are called depends on whether your operating system uses the systemd or older SysV init mechanism. This is detailed in sections specific to each init mechanism below.

Templates for the init scripts are stored in:

```
/p4/common/etc/init.d
```

4.2.2.1. Configuring systemd

Configuring systemd for p4d

RHEL/CentOS 7 or 8, SuSE 12, Ubuntu (>= v16.04), Amazon Linux 2, and other Linux distributions utilize **systemd** / **systemctl** as the mechanism for controlling services, replacing the earlier SysV init process. Templates for systemd *.service files are included in the SDP distribution in `$SDP/Server/Unix/p4/common/etc/systemd/system`.

Note that using `systemd` is strongly recommended on systems that support it, for safety reasons. However, enabling services to start automatically on boot is optional.

To configure p4d for systemd, run these commands as the root user:

```
I=1
```

Replace the `1` on the right side of the `=` with your SDP instance name, e.g. xyz if your P4ROOT is /p4/xyz/root. Then:

```
cd /etc/systemd/system
sed -e "s:__INSTANCE__:$I:g" -e "s:__OSUSER__:perforce:g"
$SDP/Server/Unix/p4/common/etc/systemd/system/p4d_N.service.t > p4d_${I}.service
chmod 644 p4d_${I}.service
systemctl daemon-reload
```

If you are configuring p4d for more than one instance, repeat the `I=` command with each instance name on the right side of the `=`, and then repeat the block of commands above.

Once configured, the following are sample management commands to start, stop, and status the service. These following commands are typically run as the `perforce` OSUSER using `sudo` where needed:

```
systemctl cat p4d_1
systemctl status p4d_1
sudo systemctl start p4d_1
sudo systemctl stop p4d_1
```

if running with SELinux in enforcing mode, see Section 4.2.2.2, “Enabling systemd under SELinux”

Systemd Required if Configured

If you are using `systemd` and you have configured services as above, then you can no longer run the `\*_init` scripts directly for normal service `start`/`stop`, though they can still be used for `status`. The `sudo systemctl` commands **must** be used for `start`/`stop`. Attempting to run the underlying scripts directly will result in an error message if systemd is configured. This is for safety: systemd's concept of service status (up or down) is only reliable when systemd starts and stops the service itself. The SDP init scripts require the systemd mechanism (using the `systemctl` command) to be used if it is configured. This ensures that services will gracefully stop the service on reboot (which would otherwise present a risk of data corruption for p4d on reboot).

The SDP requires systemd to be used if it is configured, and we strongly recommend using system on systems that use it. We recommend this to eliminate the risk of corruption on reboot, and also for consistency of operations. However, the SDP does not require systemd to be used. The SDP uses `systemctl cat` of the service name (e.g. `p4d_1`) to determine if systemd is configured for any given service.

Configuring systemd for p4p

Configuring p4p for systemd is identical to the configuration the for p4d, except that you would replace `p4d` with `p4p` in the sample commands above for configuring p4d.

Note SELinux fix (Section 4.2.2.2, “Enabling systemd under SELinux”) may be similarly required.

Configuring systemd for p4dtg

Configuring p4dtg for systemd is identical to the configuration the for p4d, except that you would replace `p4d` with `p4dtg` in the sample commands above for configuring p4d.

Note SELinux fix (Section 4.2.2.2, “Enabling systemd under SELinux”) may be similarly required.

Configuring systemd p4broker - multiple configs

Configuring p4broker for systemd can be similar to configuration the for p4d, but there are extra options as you may choose to run multiple broker configurations. For example, you may have:

- a default p4broker configuration that runs when the service is live,
- a "Down for Maintenance" (DFM) broker used in place of the default broker during maintenance to help lock out users broadcasting a friendly message like "Perforce is offline for scheduled maintenance."
- SSL broker config enabling an SSL-encrypted connection to a server that might not yet require

SSL encryption for all users.

The service name for the default broker configuration is always `p4broker_N`, where `N` is the instance name, e.g. `p4broker_1` for instance `1`. This uses the default broker config file, `/p4/common/config/p4_1.broker.cfg`.

Host Specific Broker Config

For circumstances where host-specific broker configuration is required, the default broker will use a `/p4/common/config/p4_N.broker.<short-hostname>.cfg` if it exists, where `<short-hostname>` is whatever is returned by the command `hostname -s`. The logic in the broker init script will favor the host-specific config if found, otherwise it will use the standard broker config.

When alternate broker configurations are used, each alternate configuration file must have a separate systemd unit file associated with managing that configuration. The service file must specify a configuration tag name, such as 'dfm' or 'ssl'. That tag name is used to identify both the broker config file and the systemd unit file for that broker. If the broker config is intended to run concurrently with the default broker config, it must listen on a different port number than the one specified in the default broker config. If it is only intended to run in place of the standard config, as with a 'dfm' config, then it should listen on the same port number as the default broker if a default broker is used, or else the same port as the p4d server if brokers are used only for dfm. The systemd service for a broker intended to run only during maintenance should not be enabled, and thus only manually started/stopped as part of maintenance procedures.

If maintenance procedures involve a reboot of a server machine, you may also want to disable all services during maintenance and re-enable them afterward.

For example, say you want a default broker, a DFM broker, and an SSL broker for instance 1. The default and SSL brokers will run continuously, and the DFM broker only during scheduled maintenance. The following broker config files would be needed in `/p4/common/config`:

- `p4_1.broker.cfg` - default broker, targets p4d on port 1999, listens on port 1666
- `p4_1.broker.ssl.cfg` - SSL broker, targets p4d on port 1999, listens on port 1667
- `p4_1.broker.dfm.cfg` - DFM broker, targets p4d on port 1999 , listens on port 1666.

Then, create a systemd *.service file that references each config. For the default broker, use the template just as with p4d above. Do the following as the `root` user:

```
I=1
```

Replace the `1` on the right side of the `=` with your SDP instance name, e.g. xyz if your P4ROOT is /p4/xyz/root. Then:


```
cd /etc/systemd/system
sed -e "s:__INSTANCE__:$I:g" -e "s:__OSUSER__:perforce:g"
$SDP/Server/Unix/p4/common/etc/systemd/system/p4broker_N.service.t >
p4broker_$I.service
chmod 644 p4broker_$I.service
systemctl daemon-reload
```

Once configured, the following are sample management commands to start, stop, and status the service. These following commands are typically run as the `perforce` OSUSER using `sudo` where needed:

```
systemctl cat p4broker_1
systemctl status p4broker_1
sudo systemctl start p4broker_1
sudo systemctl stop p4broker_1
```

For the non-default broker configs for the SSL and DFM brokers, start by copying the default broker config to a new *.service file with `_ssl` or `_dfm` inserted into the name, like so:

```
cd /etc/systemd/system
cp p4broker_1.service p4broker_1_dfm.service
cp p4broker_1.service p4broker_1_ssl.service
```

Next, modify the p4broker_1_dfm.service file and p4broker_1_ssl.service files with a text editor, making the following edits:

- Find the string that says `using default broker config`, and change the word `default` to `dfm` or `ssl` as appropriate, so it reads something like `using dfm broker config`.
- Change the ExecStart and ExecStop definitions by appending the `dfm` or `ssl` tag. For example, change these two lines:

  ```
  ExecStart=/p4/1/bin/p4broker_1_init start
  ExecStop=/p4/1/bin/p4broker_1_init stop
  ```

to look like this for the `dfm` broker:

```
ExecStart=/p4/1/bin/p4broker_1_init start dfm
ExecStop=/p4/1/bin/p4broker_1_init stop dfm
```

After any modifications to systemd *.services files are made, reload them into with:

```
systemctl daemon-reload
```

At this point, the services `p4broker_1`, `p4broker_1_dfm`, and `p4broker_1_ssl` can be started and stopped normally.

Finally, enable those services you want to start on boot. In our example here, we will enable the default and ssl broker services to start on boot, but not the DFM broker:

```
systemctl enable p4broker_1
systemctl enable p4broker_1_ssl
```

You must be aware of which configurations listen on the same port, and not try to runs those configurations concurrently. In this case, ensure the default and dfm brokers don't run at the same time. So, for example, you might start a maintenance window with:

```
sudo systemctl stop p4broker_1 p4d_1
sudo systemctl start p4broker_1_dfm
```

and end maintenance in the opposite order:

```
sudo systemctl stop p4broker_1_dfm
sudo systemctl start p4broker_1 p4d_1
```

Details may vary depending on what is occurring during maintenance.

Note SELinux fix (Section 4.2.2.2, "Enabling systemd under SELinux") may be similarly required.

4.2.2.2. Enabling systemd under SELinux

If you have `SELinux` in `Enforcing` mode, then you may get an error message when you try and start the service:

```
$ systemctl start p4d_1
$ systemctl status p4d_1
:
 Active: failed
Process: 1234 ExecStart=/p4/1/bin/p4d_1_init start (code=exited, status=203/EXEC)
:

$ journalctl -u p4d_1 --no-pager | tail
:
... p4d_1.service: Failed to execute command: Permission denied
... p4d_1.service: Failed at step EXEC spawning p4d_1_init: Permission denied
```

This can be easily fixed (as `root`):

```
semanage fcontext -a -t bin_t /p4/1/bin/p4d_1_init
restorecon -vF /p4/1/bin/p4d_1_init
```

If not already installed then `yum install policycoreutils-python-utils` gets you the basic commands mentioned above - you don't need the full `setools` which comes with a GUI!

Then try again:

```
systemctl start p4d_1
systemctl status p4d_1
```

The status command should show `Active: active`

For troubleshooting SELinux, we recommend the setroubleshoot utility

Look for denied in /var/log/audit.log and then `ls -alZ <file>` for any file that triggered the denied message and go from there.

4.2.2.3. Configuring SysV Init Scripts

To configure services for an instance on systems using the SysV init mechanism, run these commands as the `root` user: Repeat this step for all instance init scripts you wish to configure as system services.

```
cd /etc/init.d
ln -s /p4/1/bin/p4d_1_init
chkconfig --add p4d_1_init
```

With that done, you can `start`/`stop`/`status` the service as `root` by running commands like:

```
service p4d_1_init status
service p4d_1_init start
service p4d_1_init stop
```

On SysV systems, you can also run the underlying init scripts directly as either the `root` or `perforce` user. If run as `root`, the script becomes `perforce` immediately, so that no processing occurs as root.

4.2.3. Configuring Automatic Service Start on Boot

You may want to configure your server machine such that the Helix Core Server for any given instance (and/or Proxy and/or Broker) will start automatically when the machine boots.

This is done using Systemd or Init scripts as covered below.

4.2.3.1. Automatic Start for Systems using systemd

Once systemd services are configured, you can enable the service to start on boot with a command like this, run a s `root`:

```
systemctl enable p4d_1
```

The `enable` command configures the services to start automatically when the machine reboots, but does not immediately start the service. *Enabling services is optional*; you can start and stop the services manually regardless of whether it is enabled for automatic start on boot.

4.2.3.2. For systems using the SysV init mechanism

Once SysV services are configured, you can enable the service to start on boot with a command like this, run as `root`:

```
chkconfig p4d_1_init on
```

4.2.4. SDP Crontab Templates

The SDP includes basic crontab templates for master, replica, and edge servers in:

```
/p4/common/etc/cron.d
```

These define schedules for routine checkpoint operations, replica status checks, and email reviews.

4.2.5. Completing Your Server Configuration

1. Ensure that the admin user configured above has the correct password defined in `/p4/common/config/.p4passwd.p4_1.admin`, and then run the `p4login1` script (which calls the `p4 login` command using the `.p4passwd.p4_1.admin` file).
2. For new server instances, run this script, which sets several recommended configurables:

   ```
   cd /p4/sdp/Server/setup/configure_new_server.sh 1
   ```

For existing servers, examine this file, and manually apply the `p4 configure` command to set configurables on your Perforce server instance.

Initialize the perforce user's crontab with one of these commands:

```
crontab /p4/p4.crontab
```

and customize execution times for the commands within the crontab files to suite the specific installation.

The SDP uses wrapper scripts in the crontab: `run_if_master.sh`, `run_if_edge.sh`, `run_if_replica.sh`. We suggest you ensure these are working as desired, e.g.

```
/p4/common/bin/run_if_master.sh 1 echo yes
/p4/common/bin/run_if_replica.sh 1 echo yes
/p4/common/bin/run_if_edge.sh 1 echo yes
```

The above should output `yes` if you are on the master (commit) machine (or replica/edge as appropriate), but otherwise nothing. Any issues with the above indicate incorrect values for `$MASTER_ID`, or for other values within `/p4/common/config/p4_1.vars` (assuming instance `1`). You can debug this with:

```
bash -xv /p4/common/bin/run_if_master.sh 1 echo yes
```

If in doubt contact support.

4.2.6. Validating your SDP installation

Source your SDP environment variables and check that they look appropriate - for <instance> `1`:

```
source /p4/common/bin/p4_vars 1
```

The output of `p4 set` should be something like:

```
P4CONFIG=/p4/1/.p4config (config 'noconfig')
P4ENVIRO=/dev/null/.p4enviro
P4JOURNAL=/p4/1/logs/journal
P4LOG=/p4/1/logs/log
P4PCACHE=/p4/1/cache
P4PORT=ssl:1666
P4ROOT=/p4/1/root
P4SSLDIR=/p4/ssl
P4TICKETS=/p4/1/.p4tickets
P4TRUST=/p4/1/.p4trust
P4USER=perforce
```

There is a script `/p4/common/bin/verify_sdp.sh`. Run this specifying the <instance> id, e.g.

```
/p4/common/bin/verify_sdp.sh 1
```

The output should be something like:

```
verify_sdp.sh v5.6.1 Starting SDP verification on host helixcorevm1 at Fri 2020-08-14
17:02:45 UTC with this command line:
/p4/common/bin/verify_sdp.sh 1
```

```
If you have any questions about the output from this script, contact support-helix-
core@perforce.com.
--------------------------------------------------------------------------------
Doing preflight sanity checks.
Preflight Check: Ensuring these utils are in PATH: date ls grep awk id head tail
Verified: Essential tools are in the PATH.
Preflight Check: cd /p4/common/bin
Verified: cd works to: /p4/common/bin
Preflight Check: Checking current user owns /p4/common/bin
Verified: Current user [perforce] owns /p4/common/bin
Preflight Check: Checking /p4 and /p4/<instance> are local dirs.
Verified: P4HOME has expected value: /p4/1
Verified: This P4HOME path is not a symlink: /p4/1
Verified: cd to /p4 OK.
Verified: Dir /p4 is a local dir.
Verified: cd to /p4/1 OK.
Verified: P4HOME dir /p4/1 is a local dir.
```

Finishing with:

```
Verifications completed, with 0 errors and 0 warnings detected in 57 checks.
```

If it mentions something like:

```
Verifications completed, with 2 errors and 1 warnings detected in 57 checks.
```

then review the details. If in doubt contact Perforce Support: support-helix-core@perforce.com

4.3. Local SDP Configuration

There are many scenarios where you may need to override a default value that the SDP provides. These changes must be done in specific locations so that your changes persist across SDP upgrades. There are two different scopes of configuration to be aware of and two locations you can place your configuration in:

Location	Scope	Description
/p4/common/site/config/$P4SERVER.vars.local	SDP Instance Specific	Single configuration file that is scoped to a single SDP Instance

Location	Scope	Description
/p4/common/site/config/$P4SERVER.vars.local.d/*	SDP Instance Specific	Directory of configuration files that are scoped to a single SDP Instance
/p4/common/site/config/p4_vars.local	SDP Wide	Single configuration file that is scoped to all SDP Instances
/p4/common/site/config/p4_vars.local.d/*	SDP Wide	Directory of configuration files that are scoped to all SDP Instances

4.3.1. Load Order

1. `/p4/common/bin/p4_vars`
2. `/p4/common/site/config/p4_vars.local`
3. `/p4/common/site/config/p4_vars.local.d/*`
4. `/p4/common/config/$P4SERVER.vars`
5. `/p4/common/site/config/$P4SERVER.vars.local.d/*`

4.4. Setting your login environment for convenience

Consider adding this to your `.bashrc` for the perforce user as a convenience for when you login:

```
echo "source /p4/common/bin/p4_vars 1" >> ~/.bashrc
```

Obviously if you have multiple instances on the same machine you might want to setup an alias or two to quickly switch between them.

4.5. Configuring protections, file types, monitoring and security

After the server instance is installed and configured, either with the Helix Installer or a manual installation, most sites will want to modify server permissions ("Protections") and security settings. Other common configuration steps include modifying the file type map and enabling process monitoring. To configure permissions, perform the following steps:

1. To set up protections, issue the `p4 protect` command. The protections table is displayed.
2. Delete the following line:

   ```
   write user * * //depot/...
   ```

3. Define protections for your repository using groups. Perforce uses an inclusionary model. No access is given by default, you must specifically grant access to users/groups in the protections

table. It is best for performance to grant users specific access to the areas of the depot that they need rather than granting everyone open access, and then trying to remove access via exclusionary mappings in the protect table even if that means you end up generating a larger protect table.

4. To set the default file types, run the p4 typemap command and define typemap entries to override Perforce's default behavior.
5. Add any file type entries that are specific to your site. Suggestions:
   - For already-compressed file types (such as `.zip`, `.gz`, `.avi`, `.gif`), assign a file type of `binary+Fl` to prevent p4d from attempting to compress them again before storing them.
   - For regular binary files, add `binary+l` to make so that only one person at a time can check them out.

   A sample file is provided in `$SDP/Server/config/typemap`

If you are doing things like games development with `Unreal Engine` or `Unity`, then there are specific recommended typemap to add in KB articles: Search the Knowledge Base

1. To make your changelists default to restricted (for high security environments):

```
p4 configure set defaultChangeType=restricted
```

4.6. Operating system configuration

Check Chapter 7, *Maximizing Server Performance* for detailed recommendations.

4.6.1. Configuring email for notifications

Use Postfix - which Integrates easily with Gmail, Office365 etc just search for postfix and the email provider. Examples:

- https://www.howtoforge.com/tutorial/configure-postfix-to-use-gmail-as-a-mail-relay/
- https://support.google.com/accounts/answer/185833?hl=en#zippy=%2Cwhy-you-may-need-an-app-password
- https://www.middlewareinventory.com/blog/postfix-relay-office-365/#3_Office_365_SMTP_relay_Discussed_in_this_Post

Please note that for Gmail:

- You must turn on 2FA for the account which is trying to create an app password
- The organization must allow 2FA (2-Step Verification) - this is normally turned off in Google Workspace (formerly known as G Suite).

Testing of email once configured:

```
echo "Test email" | mail -s "Test email subject" user@example.com
```

If there are problems sending email, then this may find the problem:

```
grep postfix /var/log/*
cat /var/log/maillog
```

4.6.2. Swarm Email Configuration

The advantage of installing Postfix is that it is easily testable from the command line as above.

The Swarm configuration then becomes editing `config.php` as below (optional sender address) and restarting Swarm in the normal way (resetting its cache first):

```
    // this block should be a peer of 'p4'
    'mail' => array(
        // 'sender' => 'swarm@my.domain',   // defaults to 'notifications@hostname'
        'transport' => array(
            'name' => 'localhost',          // name of SMTP host
            'host' => 'localhost',          // host/IP of SMTP host
            ),
        ),
    ),
```

Restarting Swarm (on CentOS):

```
cd /opt/perforce/swarm/data
rm cache/*cache.php
systemctl restart httpd
```

4.6.3. Configuring PagerDuty for notifications

The default behavior of the SDP is to use email for delivering alerts and log files. This section details replacing email with PagerDuty.

4.6.3.1. Prerequisites

- PagerDuty Account
- PagerDuty Service where SDP/Helix Core incidents will be created
- Events API V2 Integration added to PagerDuty Service, this will produce an Integration Key which will be used later
- Install PagerDuty CLI

4.6.3.2. SDP Configuration

The following can be added to `/p4/common/site/config/p4_vars.local` to configure the SDP to use PagerDuty:

```
# set this environment variable to the Integration Key that was created when adding
the
# Events API V2 Integration to your PagerDuty Service
export PAGERDUTY_ROUTING_KEY="2ac2....e5c3"
```

4.6.3.3. Optional variables

The SDP will automatically set the Title of the PagerDuty Incident based on the exception that occurred. The SDP will also include the log file from the exception (example: checkpoint log, p4verify log, etc).

If you have multiple Helix Core servers it will be helpful to include some additional context with the incident so you know which server the alert is coming from.

The following environment variable can optionally be used to add additional context to the PagerDuty Incident:

```
# export PAGERDUTY_CUSTOM_FIELD=""
```

Example Additional Context Configuration

The following snippet will create environment variables in `p4_vars.local` that will provide additional context in each PagerDuty Incident:

```
curl -s -H Metadata:true --noproxy "*" "http://169.254.169.254/metadata/instance?api-
version=2021-02-01" > /tmp/azure_metadata
cat <<-EOF >> /p4/common/site/config/p4_vars.local
export PAGERDUTY_ROUTING_KEY="2ac2....e5c3"
export VM_ID="$(jq -r '.compute.vmId' /tmp/azure_metdata)"
export REGION="$(jq -r '.compute.location' /tmp/azure_metdata)"
export AZURE_SUBSCRIPTION_ID="$(jq -r '.compute.subscriptionId' /tmp/azure_metdata)"
export PAGERDUTY_CUSTOM_FIELD=\$(cat <<-END
##############################################
Azure Subscription: \$AZURE_SUBSCRIPTION_ID
Region: \$REGION
Azure VM ID: \$VM_ID
##############################################
END
)
EOF
```

The following context will be added as a field on the PagerDuty Incident:

```
###############################################
Azure Subscription: f306878d-d321-4731-4cd3-f3afafbbd3ac
Region: eastus
Azure VM ID: 5ee13bfe-8a0c-486f-ae08-c43e44255d15
###############################################
```

4.6.4. Configuring AWS Simple Notification Service (SNS) for notifications

The default behavior of the SDP is to use email for delivering alerts and log files. This section details replacing email with AWS SNS.

4.6.4.1. Prerequisites

- AWS CLI installed
- Authorization for `publish` to a AWS SNS topic

4.6.4.2. SDP Configuration

The following can be added to `/p4/common/config/p4_1.vars` to configure the SDP to use SNS:

```
# SNS Alert Configurations
# Two methods of authentication are supported: key pair (on prem, azure, etc) and IAM
role (AWS deployment)
# In the case of IAM role the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment
variables must not be set, not even empty strings
```

```
# To test SNS delivery use the following command: aws sns publish --topic-arn
$SNS_ALERT_TOPIC_ARN --subject test --message "this is a test"
```

```
# export AWS_ACCESS_KEY_ID=""
# export AWS_SECRET_ACCESS_KEY=""
```

```
export AWS_DEFAULT_REGION="us-east-1"
export SNS_ALERT_TOPIC_ARN="arn:aws:sns:us-east-1:541621974560:Perforce-Notifications-
SnsTopic-1FIRH0KEAXTU"
```

4.6.4.3. Example IAM Policy

The following is an example policy that could be used for either an IAM Role or an IAM user with key/secret:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "sns:Publish",
            "Resource": "arn:aws:sns:us-east-1:541621974560:Perforce-Notifications-*",
            "Effect": "Allow"
        }
    ]
}
```

4.7. Other server configurables

There are various configurables that you should consider setting for your server instance.

Some suggestions are in the file: `$SDP/Server/setup/configure_new_server.sh`

Review the contents and either apply individual settings manually, or edit the file and apply the newly edited version. If you have any questions, please see the configurables section in Command Reference Guide appendix (get the right version for your server!). You can also contact support regarding questions.

4.8. Archiving configuration files

Now that the server instance is running properly, copy the following configuration files to the hxdepots volume for backup:

- Any init scripts used in `/etc/init.d` or any systemd scripts to `/etc/systemd/system`
- A copy of the crontab file, obtained using `crontab -l`.
- Any other relevant configuration scripts, such as cluster configuration scripts, failover scripts, or disk failover configuration files.

4.9. Installing Swarm Triggers

On the commit server (**NOT** the Swarm machine), get it setup to connect to the Perforce package repo (if not already done). See: https://www.perforce.com/perforce-packages

Install the trigger package, e.g.:

- `yum install helix-swarm-triggers` (if Red Hat family, i.e. RHEL, Rocky Linux, CentOS, Amazon Linux).
- `apt install helix-swarm-triggers` (for Ubuntu)

Then (for SDP environments for ease):

```
sudo chown -R perforce:perforce /opt/perforce/etc
```

Then install the triggers on the p4d server. Something like:

```
vi /opt/perforce/etc/swarm-triggers.conf
```

Make it look something like (in SDP env):

```
SWARM_HOST='https://swarm.p4.p4bsw.com'
SWARM_TOKEN='MY-UUID-STYLE-TOKEN'
ADMIN_USER='swarm'
ADMIN_TICKET_FILE='/p4/1/.p4tickets'
P4_PORT='ssl:1666'
P4='/p4/1/bin/p4_1'
EXEMPT_FILE_COUNT=0
EXEMPT_EXTENSIONS=''
VERIFY_SSL=1
TIMEOUT=30
IGNORE_TIMEOUT=1
IGNORE_NOSERVER=1
```

Then test that config file:

```
chmod +x /p4/sdp/Unsupported/setup/swarm_triggers_test.sh
/p4/sdp/Unsupported/setup/swarm_triggers_test.sh
```

Get that to be happy. May require iteration of the conf file, trigger install, etc.

Then install triggers on the server.

```
cd /p4/1/tmp
p4 triggers -o > temp_file.txt

/opt/perforce/swarm-triggers/bin/swarm-trigger.pl -o >> tmp_file.txt

vi tmp_file.txt # Clean up formatting, make it syntactically correct.

p4 triggers -i < temp_file.txt
p4 triggers -o # Make sure it's there.
```

Then test!


```
mkdir /p4/1/tmp/swarm_test
cd /p4/1/tmp/swarm_test

export P4CONFIG=.p4config
echo P4CLIENT=swarm_test.$(hostname -s)>>.p4config

# Make a workspace, map View to some location where we can edit harmlessly,
# or use a stream like //sandbox/main
p4 client

p4 add chg.txt

# The important thing is '#review' which trigger will process
p4 change -o | sed 's:<enter description here>:#review' > chg.txt
p4 change -i < chg.txt

p4 shelve -c CL  #  Use CL listed in output from prior command
p4 describe -s CL   # if #review gets replace by something like #review-12345, you're
Done!
```

Chapter 5. Backup, Replication, and Recovery

Perforce server instances maintain *metadata* and *versioned files*. The metadata contains all the information about the files in the depots. Metadata resides in database (db.*) files in the server instance's root directory (P4ROOT). The versioned files contain the file changes that have been submitted to the repository. Versioned files reside on the hxdepots volume.

This section assumes that you understand the basics of Perforce backup and recovery. For more information, consult the Perforce System Administrator's Guide and failover.

5.1. Typical Backup Procedure

The SDP's maintenance scripts, run as `cron` tasks, periodically back up the metadata. The weekly sequence is described below.

Seven nights a week, perform the following tasks:

1. Truncate the active journal.
2. Replay the journal to the offline database. (Refer to Figure 2: SDP Runtime Structure and Volume Layout for more information on the location of the live and offline databases.)
3. Create a checkpoint from the offline database.
4. Recreate the offline database from the last checkpoint.

Once a week, perform the following tasks:

1. Verify all depot files.

Once every few months, perform the following tasks:

1. Stop the live server instance.
2. Truncate the active journal.
3. Replay the journal to the offline database. (Refer to Figure 2: SDP Runtime Structure and Volume Layout for more information on the location of the live and offline databases.)
4. Archive the live database.
5. Move the offline database to the live database directory.
6. Start the live server instance.
7. Create a new checkpoint from the archive of the live database.
8. Recreate the offline database from the last checkpoint.
9. Verify all depots.

This normal maintenance procedure puts the checkpoints (metadata snapshots) on the hxdepots volume, which contains the versioned files. Backing up the hxdepots volume with a normal backup utility like *rsync* preserves the critical assets necessary for recovery.

To ensure that the backup does not interfere with the metadata backups (checkpoints), coordinate backup of the hxdepots volume using the SDP maintenance scripts.

The preceding maintenance procedure minimizes service outage, because checkpoints are created from offline or saved databases while the live p4d server process is running on the live databases in P4ROOT.

With no additional configuration, the normal maintenance prevents loss of more than one day's metadata changes. To provide an optimal Recovery Point Objective (RPO), the SDP provides additional tools for replication.

5.2. Planning for HA and DR

The concepts for HA (High Availability) and DR (Disaster Recovery) are fairly similar - they are both types of Helix Core replica.

When you have server specs with `Services` field set to `commit-server`, `standard`, or `edge-server` - see deployment architectures you should consider your requirements for how to recover from a failure to any such servers.

See also Replica types and use cases

The key issues are around ensuring that you have have appropriate values for the following measures for your Helix Core installation:

- RTO - Recovery Time Objective - how long will it take you to recover to a backup?
- RPO - Recovery Point Objective - how much data are you prepared to risk losing if you have to failover to a backup server?

We need to consider planned vs unplanned failover. Planned may be due to upgrading the core Operating System or some other dependency in your infrastructure, or a similar activity.

Unplanned covers risks you are seeking to mitigate with failover:

- loss of a machine, or some machine related hardware failure (e.g. network)
- loss of a VM cluster
- failure of storage
- loss of a data center or machine room
- etc...

So, if your main `commit-server` fails, how fast should be you be able to be up and running again, and how much data might you be prepared to lose? What is the potential disruption to your organization if the Helix Core repository is down? How many people would be impacted in some way?

You also need to consider the costs of your mitigation strategies. For example, this can range from:

- taking a backup once per 24 hours and requiring maybe an hour or two to restore it. Thus you

might lose up to 24 hours of work for an unplanned failure, and require several hours to restore.

- having a high availability replica which is a mirror of the server hardware and ready to take over within minutes if required

Having a replica for HA or DR is likely to reduce your RPO and RTO to well under an hour (<10 minutes if properly prepared for) - at the cost of the resources to run such a replica, and the management overhead to monitor it appropriately.

Typically we would define:

- An HA replica is close to its upstream server, e.g. in the same Data Center - this minimizes the latency for replication, and reduces RPO
- A DR replica is in a more remote location, so maybe risks being further behind in replication (thus higher RPO), but mitigates against catastrophic loss of a data center or similar. Note that "further behind" is still typically seconds for metadata, but can be minutes for submits with many GB of files.

5.2.1. Further Resources

- High Reliability Solutions

5.2.2. Creating a Failover Replica for Commit or Edge Server

A commit server instance is the ultimate store for submitted data, and also for any workspace state (WIP - work in progress) for users directly working with the commit server (part of the same "data set")

An edge server instance maintains its own copy of workspace state (WIP). If you have people connecting to an edge server, then any workspaces they create (and files they open for some action) will be only stored on the edge server. Thus it is normally recommended to have an HA backup server, so that users don’t lose their state in case of failover.

There is a concept of a "build edge" which is an edge server which only supports build farm users. In this scenario it may be deemed acceptable to not have an HA backup server, since in the case of failure of the edge, it can be re-seeded from the commit server. All build farm clients would be recreated from scratch so there would be no problems.

5.2.3. What is a Failover Replica?

A Failover is the hand off of the role of a master/primary/commit server from a primary server machine to a standby replica (typically on a different server machine). As part of failover processing the secondary/backup is promoted to become the new master/primary/commit server.

As of 2018.2 release, p4d supports a `p4 failover` command that performs a failover to a `standby` replica (i.e. a replica with `Services:` field value set to `standby` or `forwarding-standby`). Such a replica performs a `journalcopy` replication of metadata, with a local pull thread to update its `db.*` files. After the failover is complete, traffic must be redirected to the server machine where newly promoted standby server operates, e.g. with a DNS change (possibly automated with a post-failover

trigger).

See also: Configuring a Helix Core Standby.

On Linux the SDP script `mkrep.sh` greatly simplifies the process of setting up a replica suitable for use with the `p4 failover` command. See: Section 5.3.4, “Using mkrep.sh”.

5.2.4. Mandatory vs Non-mandatory Standbys

You can modify the `Options:` field of the server spec of a `standby` or `forwarding-standby` replica to make it `mandatory`. This setting affects the mechanics of how failover works.

When a `standby` server instance is configured as mandatory, the master/commit server will wait until this server confirms it has processed journal data before allowing that journal data to be released to other replicas. This can simplify failover if the master server is unavailable to participate in the failover, since it provides a guarantee that no downstream servers are **ahead** of the replica.

This guarantee is important, as it ensures downstream servers can simply be re-directed to point to the standby after the master server has failed over to its standby, and will carry on working without problems or need for human intervention on the servers.

Failovers in which the master does not participate are generally referred to as *unscheduled* or *reactive*, and are generally done in response to an unexpected situation. Failovers in which the master server is alive and well at the start of processing, and in which the master server participates in the failover, are referred to as *scheduled* or *planned*.

If a server which is marked as `mandatory` goes offline for any reason, the replication to other replicas will stop replicating. In this scenario, the server spec of the replica can be changed to `nomandatory`, and then replication will immediately resume, so long as the replication has not been offline for so long that the master server has removed numbered journals that would be needed to catch up (typically several days or weeks depending on the KEEPJNLS setting). If this happens, the p4d server logs of all impacted servers will clearly indicate the root cause, so long p4d versions are 2019.2 or later.

If set to `nomandatory` then there is no risk of delaying downstream replicas, however there is no guarantee that they will be able to switch seamlessly over to the new server in event of an unscheduled failover.

We recommend creating `mandatory` standby replica(s) if the server is local to its commit server. We also recommend active monitoring in place to quickly detect replication lag or other issues.

To change a server spec to be `mandatory` or `nomandatory`, modify the server spec with a command like `p4 server p4d_ha_bos` to edit the form, and then change the value in the `Options:` field to be as desired, `mandatory` or `nomandatory`, and the save and exit the editor.

5.2.5. Server host naming conventions

This is recommended, but not a requirement for SDP scripts to implement failover.

- Use a name that does not indicate switchable roles, e.g. don't indicate in the name whether a host is a master/primary or backup, or edge server and its backup. This might otherwise lead to confusion once you have performed a failover and the host name is no longer appropriate.
- Use names ending numeric designators, e.g. -01 or -05. The goal is to avoid being in a post-failover situation where a machine with `master` or `primary` is actually the backup. Also, the assumption is that host names will never need to change.
- While you don't want switchable roles baked into the hostname, you can have static roles, e.g. use p4d vs. p4p in the host name (as those generally don't change). The p4d could be primary, standby, edge, edge's standby (switchable roles).
- Using a short geographic site is sometimes helpful/desirable. If used, use the same site tag used in the ServerID, e.g. aus.

  Valid site tags should be listed in: `/p4/common/config/SiteTags.cfg` - see Section 5.3.4.1, "SiteTags.cfg"

- Using a short tag to indicate the major OS version is **sometimes** helpful/desirable, eg. c7 for CentOS 7, or r8 for RHEL 8. This is based on the idea that when the major OS is upgraded, you either move to new hardware, or change the host name (an exception to the rule above about never changing the hostname). This option maybe overkill for many sites.
- End users should reference a DNS name that may include the site tag, but would exclude the number, OS indicator, and server type (`p4d`/`p4p`/`p4broker`), replacing all that with just `perforce` or optionally just `p4`. General idea is that users needn't be bothered by under-the-covers tech of whether something is a proxy or replica.
- For edge servers, it is advisable to include `edge` in both the host and DNS name, as users and admins needs to be aware of the functional differences due to a server being an edge server.

Examples:

- `p4d-aus-r7-03`, a master in Austin on RHEL 7, pointed to by a DNS name like `p4-aus`.
- `p4d-aus-03`, a master in Austin (no indication of server OS), pointed to by a DNS name like `p4-aus`.
- `p4d-aus-r7-04`, a standby replica in Austin on RHEL 7, not pointed to by a DNS until failover, at which point it gets pointed to by `p4-aus`.
- `p4p-syd-r8-05`, a proxy in Sydney on RHEL 8, pointed to by a DNS name like `p4-syd`.
- `p4d-syd-r8-04`, a replica that replaced the proxy in Sydney, on RHEL 8, pointed to by a DNS name like `p4-syd` (same as the proxy it replaced).
- `p4d-edge-tok-s12-03`, an edge in Tokyo running SuSE12, pointed to by a DNS name like `p4edge-tok`.
- `p4d-edge-tok-s12-04`, a replica of an edge in Tokyo running SuSE12, not pointed to by a DNS name until failover, at which point it gets pointed to by `p4edge-tok`.

FQDNs (fully qualified DNS names) of short DNS names used in these examples would also exist, and would be based on the same short names.

5.3. Full One-Way Replication

Perforce supports a full one-way replication of data from a master server to a replica, including versioned files. The p4 pull command is the replication mechanism, and a replica server can be configured to know it is a replica and use the replication command. The p4 pull mechanism requires very little configuration and no additional scripting. As this replication mechanism is simple and effective, we recommend it as the preferred replication technique. Replica servers can also be configured to only contain metadata, which can be useful for reporting or offline checkpointing purposes. See the Distributing Perforce Guide for details on setting up replica servers.

If you wish to use the replica as a read-only server, you can use the P4Broker to direct read-only commands to the replica or you can use a forwarding replica. The broker can do load balancing to a pool of replicas if you need more than one replica to handle your load.

5.3.1. Replication Setup

To configure a replica server, first configure a machine identically to the master server (at least as regards the link structure such as `/p4`, `/p4/common/bin` and `/p4/instance/*`), then install the SDP on it to match the master server installation. Once the machine and SDP install is in place, you need to configure the master server for replication.

Perforce supports many types of replicas suited to a variety of purposes, such as:

- Real-time backup,
- Providing a disaster recovery solution,
- Load distribution to enhance performance,
- Distributed development,
- Dedicated resources for automated systems, such as build servers, and more.

We always recommend first setting up the replica as a read-only replica and ensuring that everything is working. Once that is the case you can easily modify server specs and configurables to change it to a forwarding replica, or an edge server etc.

5.3.2. Replication Setup for Failover

This is just a special case of replication, but implementing Section 5.2.3, “What is a Failover Replica?”

Please note the section below Section 5.3.4, “Using mkrep.sh” which implements many details.

5.3.3. Pre-requisites for Failover

These are vital as part of your planning.

- Obtain and install a license for your replica(s)

  Your commit or standard server has a license file (tied to IP address), while your replicas do not require one to function as replicas.

  However, in order for a replica to function as a replacement for a commit or standard server, it must have a suitable license installed.

  This should be requested when the replica is first created. See the form: https://www.perforce.com/support/duplicate-server-request

- Review your authentication mechanism (LDAP etc) - is the LDAP server contactable from the replica machine (firewalls etc configured appropriately).
- Review all your triggers and how they are deployed - will they work on the failover host?

  Is the right version of Perl/Python etc correctly installed and configured on the failover host with all imported libraries?

TEST, TEST, TEST!!! It is important to test the above issues as part of your planning. For peace of mind you don't want to be finding problems at the time of trying to failover for real, which may be in the middle of the night!

On Linux:

- Review the configuration of options such as Section 7.1, "Ensure Transparent Huge Pages (THP) is turned off" and also Section 7.2, "Putting server.locks directory into RAM" are correctly configured for your HA server machine - otherwise you **risk reduced performance** after failover.

5.3.4. Using mkrep.sh

The SDP `mkrep.sh` script should be used to expand your Helix Topology, e.g. adding replicas and edge servers.

When creating server machines to be used as Helix servers, the server machines should be named following a well-designed host naming convention. The SDP has no dependency on the convention used, and so any existing local naming convention can be applied. The SDP includes a suggested naming convention in Section 5.2.5, "Server host naming conventions"

Usage

```
USAGE for mkrep.sh v3.1.3:

mkrep.sh -t <Type> -s <Site_Tag> -r <Replica_Host> [-f <From_ServerID>] [-os] [-p] [-N
<N>] [-i <SDP_Instance>] [-L <log>] [-v<n>] [-n] [-D]

or
```



```
mkrep.sh [-h|-man|-V]


DESCRIPTION:
    This script simplifies the task of creating Helix Core replicas and
    edge servers, and helps ensure they are setup with best practices.

    This script executes as two phases. In Phase 1, this script does all
    the metadata configuration to be executed on the master server that
    must be baked into a seed checkpoint for creating the replica/edge.
    This essentially captures the planning for a new replica, and can be
    done before the physical infrastructure (e.g. hardware, storage, and
    networking) is ready. Phase 1, fully automated by this script, takes
    only seconds to run.

    In Phase 2, this script provides information for the manual steps
    needed to create, transfer, and load seed checkpoints onto the
    replica/edge. The guidance is specific to type of replica created,
    based on the command line flags provided to this script. This
    processing can take a while for large data sets, as it involves
    creating and transporting checkpoints.

    Before using this script, a set of geographic site tags must be defined.
    See the FILES: below for details on a site tags.

    This script adheres to the these SDP Standards:
    * Server Spec Naming Standard:
https://swarm.workshop.perforce.com/projects/perforce-software-
sdp/view/main/doc/SDP_Guide.Unix.html#_server_spec_naming_standard
    * Journal Prefix Standard: https://swarm.workshop.perforce.com/projects/perforce-
software-sdp/view/main/doc/SDP_Guide.Unix.html#_the_journalprefix_standard

    In Phase 1, this script does the following to help create a replica or
    edge server:
    * Generates the server spec for the the replica.
    * Generates a server spec for master server (if needed).
    * Sets configurables ('p4 configure' settings) for replication.
    * Selects the correct 'Services' based on replica type.
    * Creates service user for the replica, and sets a password.
    * Creates service user for the master (if needed), and sets a password.
    * Adds newly created service users to the group 'ServiceUsers'.
    * Verifies the group ServiceUsers is granted super access in the
    protections table (and with the '-p', updates Protections).

    After these steps are completed, in Phase 2, detailed instructions are
    presented to guide the user through the remaining steps needed to complete
    the deployment of the replica.  This starts with creating a new
    checkpoint to capture all the metadata changes made by this
    script in Phase 1.

SERVICE USERS:
```

```
    Service users created by this type are always of type 'service',
    and so will not consume a licensed seat.

    Service users also have an 'AuthMethod' of 'perforce' (not
    'ldap') as is required by 'p4d' for 'service' users.  Passwords
    set for service users are long 32 character random strings
    that are not stored, as they are never needed.  Login tickets for
    service users are generated using: p4login -service -v

OPTIONS:
 -t <Type>[N]
    Specify the replica type tag. The type corresponds to the 'Type:' and
    'Services:' field of the server spec, which describes the type of services
    offered by a given replica.

    Valid type values are:
    * ha:   High Availability standby replica, for 'p4 failover' (P4D 2018.2+)
    * ham:  High Availability metadata-only standby replica, for 'p4 failover' (P4D
2018.2+)
    * ro:   Read-Only standby replica. (Discouraged; Use 'ha' instead for 'p4
failover' support.)
    * rom:  Read-Only standby replica, Metadata only. (Discouraged; Use 'ham' instead
for 'p4 failover' support.)
    * fr:   Forwarding Replica (Unfiltered).
    * fs:   Forwarding Standby (Unfiltered).
    * frm:  Forwarding Replica (Unfiltered, Metadata only).
    * fsm:  Forwarding Standby (Unfiltered, Metadata only).
    * ffr:  Filtered Forwarding Replica.  Not a valid failover target.
    * edge: Edge Server. Filtered by definition.

    Replicas with 'standby' are always unfiltered, and use the 'journalcopy'
    method of replication, which copies a byte-for-byte verbatim journal file
    rather than one that is merely logically equivalent.

    The tag has several purposes:
    1. Short Hand. Each tag represents a combination of 'Type:' and fully
    qualified 'Services:' values used in server specs.

    2. Distillation. Only the most useful Type/Services combinations have a
    shorthand form

    3. For forwarding replicas, the name includes the critical distinction of
    whether any replication filtering is used; as filtering of any kind disqualifies
    a replica from being a potential failover target. (No such distinction is
    needed for edge servers, which are filtered by definition).

 -s <Site_Tag>
    Specify a geographic site tag indicating the location and/or data center where
    the replica will physically be located. Valid site tags are defined in the site
    tags file:
```

```
    /p4/common/config/SiteTags.cfg

    A sample SiteTags.cfg file that is here:

    /p4/common/config/SiteTags.cfg.sample

 -r <Replica_Host>
    Specify the DNS name of the server machine on which the new replica will
    run. This is used in the 'ExternalAddress:' field of the replica's
    ServerID, and also used in instructions to the user for steps after
    metadata configuration is done by this script.

 -f <From_ServerID>
    Specify ServerID of the P4TARGET server from which we are replicating.
    This is used to populate the 'ReplicatingFrom' field of the server
    spec. The value must be a valid ServerID.

    This option should be used if the target is something other than the
    master. For example, to create an HA replica of an edge server, you might
    specify something like '-f p4d_edge_syd'.

 -os    Specify the '-os' option to overwrite an exising server spec.  By
    default, this script will abort of the server spec to be generated
    already exists on the Helix Core server.  Specify this option to
    overwrite the existing server spec.

 -p This script performs a check to ensure that the Protections table grants
    super access to the group ServiceUsers.

    By default, an error is displayed if the check fails, i.e. if super user
    access for the group ServiceUsers cannot be verified. This is
    because, by default, we want to avoid making changes to the Protections
    table. Some sites have local policies or custom automation that requires
    site-specific procedures to update the Protections table.

    If '-p' is specified, an attempt is made to append the Protections table
    an entry like:

    super group ServiceUsers * //...

-N <N>
    Specify '-N <N>', where N is an integer.  This is used to indicate that
    multiple replicas of the same type are to be created at the same site.
    The value specified with '-N' must be a numeric value. Left-padding with
    zeroes is allowed. For example, '-N 04' is allowed, and 'N A7' is not
    (as it is not numeric).

    This affects the ServerID to ee generated.  For example, the options
    '-t edge -s syd' would result in a ServerID of p4d_edge_syd.  To
    create a second edge in the same site, use '-t edge -s syd -N 2' to
    generate p4d_edge2_syd.
```

```
 -i <SDP_Instance>
    Specify the SDP Instance. If not specifed and the SDP_INSTANCE environment
    is defined, that value is used. If SDP_INSTANCE is not defined, the
    '-i <SDP_Instance>' argument is required.

 -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. By default, all output (stdout and stderr) goes in the logs
    directory referenced by $LOGS environment variable, in a file named
    mkrep.<timestamp>.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.'

 -n No-Op. Prints commands instead of running them.

 -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.

FILES:
    This Site Tags file defines the list of valid geographic site tags:
    /p4/common/config/SiteTags.cfg

    The contains one-line entries of the form:

    <tag>: <description>

    where <tag> is a short alphanumeric tag name for a geographic location,
    data center, or other useful distinction. This tag is incorporated into
        the ServerID of replicas or edge servers created by this script.  Tag
    names should be kept short, ideally no more than about 5 characters in
    length.

    The <description> is a one-line text description of what the tag
    refers to, which may contain spaces and ASCII punctuation.

    Blank lines and lines starting with a '#' are considered comments
    and are ignored.

REPLICA SERVER MACHINE SETUP:
    The replica/edge server machine must be have the SDP structure installed,
    either using the mkdirs.sh script included in the SDP, or the Helix
    Installer for 'green field' installations.
```

```
    When setting up an edge server, a replica of an edge server, or filtered
    replica, confirm that the JournaPrefix Standard (see URL above) structure
    has the separate checkpoints folder as identified in the 'Second Form' in
    the standard.  A baseline SDP structure can typically be extended by running
    commands like like these samples (assuming a ServerID of p4d_edge_syd or
    p4d_ha_edge_syd):

       mkdir /hxdepots/p4/1/checkpoints.edge_syd
       cd /p4/1
       ln -s /hxdepots/p4/1/checkpoints.edge_syd

CUSTOM PRE- AND POST- OPERATION AUTOMATION HOOKS:
    This script can execute custom pre- and post- processing scripts. This
    can be useful to incorporate site-specifc elements of replica setup.

    If the file /p4/common/site/mkrep/pre-mkrep.sh exists and is
    executable, it will be executed before mkrep.sh processing. If the file
    /p4/common/site/mkrep/post-mkrep.sh exists and is executable,
    it will be executed after mkrep.sh processing.

    Pre- and post- processing scripts are called with the same command line
    arguments passed to this mkrep.sh script.

    The pre- and post- processing scripts can use or ignore arguments as
    needed, though it is required to implement the '-n' flag to operate in
    preview mode, taking no actions that affect data (just as this script
    behaves).

    Pre- and post- processing scripts are expected to exit with a zero exit
    code to indicate success, and non-zero to indicate failure.

    The custom pre-processing script is executed after standard preflight
    checks complete successfully.  If a custom pre-processing script
    indicates a failure, processing is aborted before standard mkrep.sh
    processing occurs.

    The post-processing custom script is executed after the standard
    mkrep.sh processing is successful.  If a post-processing custom script
    is detected, the instructions that would be provided to the user in
    Phase 2 are not displayed, as it is expected that the custom post-
    processing will alter or handle these steps.

    Success or failure of pre- and post- processing scripts is reported in
    the log.  These scripts do not require independent logging, as all
    standard and error output is captured in the log of this mkrep.sh
    script.

    TIP: Be sure to fully test custom scripts in a test environment
    before incorporating them into production systems.
```



```
EXAMPLES:
    EXAMPLE 1 - Set up a High Availability (HA) Replica of the master.

    Add an HA replica to instance 1 to run on host bos-helix-02:
    mkrep.sh -i 1 -t ha -s bos -r bos-helix-02

    EXAMPLE 2 - Add an Edge Server to the topology.

    Add an Edge server to instance acme to run on host syd-helix-04:

    mkrep.sh -i acme -t edge -s syd -r syd-helix-04

    EXAMPLE 3 - Setup an HA replica of an edge server.

    Add a HA replica of the edge server to instance acme to run on host syd-helix-05:

    mkrep.sh -i acme -t ha -f p4d_edge_syd -s syd -r syd-helix-05

    EXAMPLE 4 - Add a second edge server in the same site as another edge.

    mkrep.sh -i acme -t edge -N 2 -s syd -r syd-helix-04
```

5.3.4.1. SiteTags.cfg

The `mkrep.sh` documentation references a SiteTags.cfg file used to register short tag names for geographic sites. Location is: `/p4/common/config/SiteTags.cfg`

Your tags should use abbreviations that are meaningful to your organization.

Example/Format

```
# Valid Geographic site tags.

# Each is intended to indicate a geography, and optionally a specific Data
# Center (or Computer Room, or Computer Closet) within a given geographic
# location.
#
# The format is:
# Name: Description
# The Name must be alphanumeric only. The Description may contain spaces.
# Lines starting with # and blank lines are ignored.

bej: Beijing, China
bos: Boston, MA, USA
blr: Bangalore, India
chi: Chicago greater metro area
cni: Chennai, India
pune: Pune, India
lv: Las Vegas, NV, USA
mlb: Melbourne, Australia
syd: Sydney, Australia
awsuseast1: AWS US-East-1
azuksouth: Azure UK South
```

A sample file exists `/p4/common/config/SiteTags.cfg.sample`.

5.3.4.2. Output of `mkrep.sh`

The output of `mkrep.sh` (which is also written to a log file in `/p4/<instance>/logs/mkrep.*`) describes a number of steps required to continue setting up the replica after the metadata configuration performed by the script is done.

5.3.5. Addition Replication Setup

In addition to steps recommended by `mkrep.sh`, there are other steps to be aware of to prepare a replica server machine.

5.3.6. SDP Installation

The SDP must first be installed on the replica server machine. If SDP already exists on the machine but not for the current instance, then `mkdirs.sh` must be used to add a new instance to the machine.

5.3.6.1. SSH Key Setup

SSH keys for the `perforce` operating system user should be setup to allow the `perforce` user to `ssh` and `rsync` among the Helix server machines in the topology. If no `~perforce/.ssh` directory exist on a machine, it can be created with this command:


```
ssh-keygen -t rsa -b 4096
```

5.4. Recovery Procedures

There are three scenarios that require you to recover server data:

Metadata	Depotdata	Action required
lost or corrupt	Intact	Recover metadata as described below
Intact	lost or corrupt	Call Perforce Support
lost or corrupt	lost or corrupt	Recover metadata as described below. Recover the hxdepots volume using your normal backup utilities.

Restoring the metadata from a backup also optimizes the database files.

5.4.1. Recovering a master server from a checkpoint and journal(s)

The checkpoint files are stored in the `/p4/instance/checkpoints` directory, and the most recent checkpoint is named `p4_instance.ckp.number.gz`. Recreating up-to-date database files requires the most recent checkpoint, from `/p4/instance/checkpoints` and the journal file from `/p4/instance/logs`.

To recover the server database manually, perform the following steps from the root directory of the server (/p4/instance/root).

Assuming instance 1:

1. Stop the Perforce Server by issuing the following command:

   ```
   /p4/1/bin/p4_1 admin stop
   ```

2. Delete the old database files in the `/p4/1/root/save` directory
3. Move the live database files (db.*) to the save directory.
4. Use the following command to restore from the most recent checkpoint.

   ```
   /p4/1/bin/p4d_1 -r /p4/1/root -jr -z /p4/1/checkpoints/p4_1.ckp.####.gz
   ```

5. To replay the transactions that occurred after the checkpoint was created, issue the following command:

```
/p4/1/bin/p4d_1 -r /p4/1/root -jr /p4/1/logs/journal
```

6. Restart your Perforce server.

If the Perforce service starts without errors, delete the old database files from `/p4/instance/root/save`.

If problems are reported when you attempt to recover from the most recent checkpoint, try recovering from the preceding checkpoint and journal. If you are successful, replay the subsequent journal. If the journals are corrupted, contact Perforce Technical Support. For full details about backup and recovery, refer to the Perforce System Administrator's Guide.

5.4.2. Recovering a replica from a checkpoint

This is very similar to creating a replica in the first place as described above.

If you have been running the replica crontab commands as suggested, then you will have the latest checkpoints from the master already copied across to the replica through the use of Section 8.6.31, "sync_replica.sh".

See the steps in the script Section 8.6.31, "sync_replica.sh" for details (note that it deletes the state and rdb.lbr files from the replica root directory so that the replica starts replicating from the start of a journal).

Remember to ensure you have logged the service user in to the master server (and that the ticket is stored in the correct location as described when setting up the replica).

5.4.3. Recovering from a tape backup

This section describes how to recover from a tape or other offline backup to a new server machine if the server machine fails. The tape backup for the server is made from the hxdepots volume. The new server machine must have the same volume layout and user/group settings as the original server. In other words, the new server must be as identical as possible to the server that failed.

To recover from a tape backup, perform the following steps (assuming instance 1):

1. Recover the hxdepots volume from your backup tape.
2. Create the `/p4` convenience directory on the OS volume.
3. Create the directories `/hxmetadata/p4/1/db1/save` and `/hxmetadata/p4/1/offline_db`.
4. Create the directories `/hxmetadata/p4/1/db2/save` and `/hxmetadata/p4/2/offline_db`.
5. Change ownership of these directories to the OS account that runs the Perforce processes.
6. Switch to the Perforce OS account, and create a link in the `/p4` directory to `/hxdepots/p4/1`.
7. Create a link in the `/p4` directory to `/hxdepots/p4/common`.
8. As a super-user, reinstall and enable the Systemd service files or or SysV init scripts.
9. Find the last available checkpoint, under `/p4/1/checkpoints`

10. Recover the latest checkpoint by running:

```
/p4/1/bin/p4d_1 -r /p4/1/root -jr -z <last_ckp_file>
```

11. Recover the checkpoint to the offline_db directory (assuming instance 1):

```
/p4/1/bin/p4d_1 -r /p4/1/offline_db -jr -z <last_ckp_file>
```

12. Reinstall the Perforce server license to the server root directory.
13. Start the perforce service by running 1/p4/1/bin/p4d_1_init start`
14. Verify that the server instance is running.
15. Reinstall the server crontab or scheduled tasks.
16. Perform any other initial server machine configuration.
17. Verify the database and versioned files by running the `p4verify.sh` script. Note that files using the `+k` file type modifier might be reported as BAD! after being moved. Contact Perforce Technical Support for assistance in determining if these files are actually corrupt.

5.4.4. Failover to a replicated standby machine

See SDP Failover Guide (PDF) or SDP Failover Guide (HTML) for detailed steps.

Chapter 6. Upgrades

This section describes both upgrades of the SDP itself, as well as upgrades of Helix software such as p4d, p4broker, p4p, and the the p4 command line client in the SDP structure.

6.1. Upgrade Order: SDP first, then Helix P4D

The SDP should normally be upgraded prior to the upgrade of Helix Core (P4D). If you are upgrading P4D to or beyond P4D 2019.1 from a prior version of P4D, you *must* upgrade the SDP first. If you run multiple instances of P4D on a given machine (potentially each running different versions of P4D), upgrade the SDP first before upgrading any of the instances.

The SDP should also be upgraded before upgrading other Helix software on machines using the SDP, including p4d, p4p, p4broker, and p4 (the command line client).

Upgrading a Helix Core server instance in the SDP framework is a simple process involving a few steps.

6.2. SDP and P4D Version Compatibility

Starting with the SDP 2020.1 release, the released versions of SDP match the released versions of P4D. So SDP r20.1 is guaranteed to work with P4D r20.1. In addition, the SDP Release Notes clarify all the specific versions of P4D supported.

The SDP is often forward- and backward-compatible with P4D versions, but for best results they should be kept in sync by upgrading SDP before P4D. This is partly because the SDP contains logic that helps upgrade P4D, which can change as P4D evolves (most recently for 2019.1).

The SDP is aware of the P4D version, and has backward-compatibility logic to support older versions of P4D. This is guaranteed for supported versions of P4D. Backward compatibility of SDP with older versions of P4D may extend farther back, though without the "officially supported" guarantee.

6.3. Upgrading the SDP

Starting with this SDP 2021.1 release, upgrades of the SDP from 2020.1 and later use a new mechanism. The SDP upgrade procedure starting from 2020.1 and later uses the `sdp_upgrade.sh` script. Some highlights of the new upgrade mechanism:

- **Automated**: Upgrades from SDP 2020.1 are automated with `sdp_upgrade.sh` provided with each new version of the SDP.
- **Continuous**: Each new SDP version, starting from SDP 2021.1, will maintain the capability to upgrade from all prior versions, so long as the starting version is SDP 2020.1 or later.
- **Independent**: SDP upgrades will enable upgrades to new Helix Core versions, but will not directly cause Helix Core upgrades to occur immediately. Each Helix Core instance can be upgraded independently of the SDP on its own schedule.

6.3.1. Sample SDP Upgrade Procedure

For complete information, see: Section 8.2.3, “sdp_upgrade.sh”.

A basic set of commands is:

```
cd /hxdepots
[[ -d downloads ]] || mkdir downloads
cd downloads
[[ -d new ]] && mv new old.$(date +'%Y%m%d-%H%M%S')
[[ -e sdp.Unix.tgz ]] && mv sdp.Unix.tgz sdp.Unix.old.$(date +'%Y%m%d-%H%M%S')
curl -L -s -O https://swarm.workshop.perforce.com/projects/perforce-software-
sdp/download/downloads/sdp.Unix.tgz
ls -l sdp.Unix.tgz
mkdir new
cd new
tar -xzf ../sdp.Unix.tgz
```

After extracting the SDP tarball, cd to the directory where the `sdp_ugprade.sh` script resides, and execute it from there:

```
cd /hxdepots/downloads/new/sdp/Server/Unix/p4/common/sdp_upgrade
./sdp_upgrade.sh -man
```

If the `curl` command cannot be used (perhaps due to lack of outbound internet access), replace that step with some other means of acquiring the SDP tarball such that it lands as `/hxdepots/downloads/sdp.Unix.tgz`, and then proceed from that point forward.

What if there is no `/hxdepots` ?

If the existing SDP does not have a `/hxdepots` directory, find the correct value with this command:

```
bash -c 'cd /p4/common; d=$(pwd -P); echo ${d%/p4/common}'
```

This can be run from any shell (bash, tcsh, zsh, etc.)

6.3.2. SDP Legacy Upgrade Procedure

If your current SDP is older than the 2020.1 release, see the SDP Legacy Upgrade Guide (for Unix) for information on upgrading SDP to SDP 2020.1 from any prior version (dating back to 2007).

6.4. Upgrading Helix Software with the SDP

The following outlines the procedure for upgrading Helix binaries using the SDP scripts.

6.4.1. Get Latest Helix Binaries

Acquire the latest Perforce Helix binaries to stage them for upgrade using the Section 8.2.1, “get_helix_binaries.sh” script.

If you have multiple server machines with SDP, staging can be done with this script on one machine first, and then the `/hxdepots/sdp/helix_binaries` folder can be rsync’d to other machines.

Alternately, this script can be run on each machine, but as patches can be released at any time, running it once and then distributing the helix_binaries directory internally via rsync is preferred to ensure all machines at your site deploy with the same binary versions.

See Section 8.2.1, “get_helix_binaries.sh”

6.4.2. Upgrade Each Instance

Use the SDP `upgrade.sh` script to upgrade each instance of Helix on the current machine, using the staged binaries. The upgrade process handles all aspects of upgrading, including adjusting the database structure, executing commands to upgrade the p4d database schema, and managing the SDP symlinks in `/p4/common/bin`.

Instances can be upgraded independently of each other.

See Section 8.2.2, “upgrade.sh”.

6.4.3. Global Topology Upgrades - Outer to Inner

For any given instance, be aware of the Helix topology when performing upgrades, specifically whether that instance has replicas and/or edge servers. When replicas and edge servers exist (and are active), the order in which `upgrade.sh` must be run on different server machines matters. Perform upgrades following an "outer to inner" strategy.

For example, say for SDP instance 1, your site has the following server machines:

- bos-helix-01 - The master (in Boston, USA)
- bos-helix-02 - Replica of master (in Boston, USA)
- nyc-helix-03 - Replica of master (in New York, USA)
- syd-helix-04 - Edge Server (in Sydney, AU)
- syd-helix-05 - Replica of Sydney edge (in Sydney)

Envision the above topology with the master server in the center, and two concentric circles.

The Replica of the Sydney edge would be done first, as it is by itself in the outermost circle.

The Edge server and two Replicas of the master are all at the next inner circle. So bos-helix-02, nyc-

helix-03, and syd-helix-04 could be upgraded in any order with respect to each other, or even simultaneously, as they are in the same circle.

The master is the innermost, and would be upgraded last.

A few standards need to be in place to make this super easy:

- The `perforce` operating system user would have properly configured SSH keys to allow passwordless ssh from the master to all other servers.
- The `perforce` user shell environment (~/.bash_profile and ~/.bashrc) ensured that the SDP shell environment automatically sourced

The Helix global topology upgrade could be done something like, starting as perforce@bos-helix-01:

```
cd /p4/sdp/helix_binaries
./get_helix_binaries.sh
rsync -a /p4/sdp/helix_binaries/ syd-helix-05:/p4/sdp/helix_binaries
rsync -a /p4/sdp/helix_binaries/ syd-helix-04:/p4/sdp/helix_binaries
rsync -a /p4/sdp/helix_binaries/ nyc-helix-03:/p4/sdp/helix_binaries
rsync -a /p4/sdp/helix_binaries/ bos-helix-02:/p4/sdp/helix_binaries
```

Then do a preview of the upgrade on all machines, in outer-to-inner order:

```
ssh syd-helix-05 upgrade.sh
ssh syd-helix-04 upgrade.sh
ssh nyc-helix-03 upgrade.sh
ssh bos-helix-02 upgrade.sh
ssh bos-helix-01 upgrade.sh
```

On each machine, check for a message in the output that contains `Success: Finished`. If that looks good, then proceed to execute the actual upgrades:

```
ssh syd-helix-05 upgrade.sh -y
ssh syd-helix-04 upgrade.sh -y
ssh nyc-helix-03 upgrade.sh -y
ssh bos-helix-02 upgrade.sh -y
ssh bos-helix-01 upgrade.sh -y
```

As with the preview, check for a message in the output that contains `Success: Finished`.

6.5. Database Modifications

Occasionally modifications are made to the Perforce database from one release to another. For example, server upgrades and some recovery procedures modify the database.

When upgrading the server, replaying a journal patch, or performing any activity that modifies the

db.* files, you must ensure that the offline checkpoint process is functioning correctly so that the files in the offline_db directory match the ones in the live server directory.

Normally upgrades to the offline_db after a P4D upgrade will be applied by rotating the journal in the normal way, and applying it to the offline_db.

In some cases it is necessary to restart the offline checkpoint process and the easiest way to is to run the live_checkpoint script after modifying the db.* files, as follows:

```
/p4/common/bin/live_checkpoint.sh 1
```

This script makes a new checkpoint of the modified database files in the live `root` directory, then recovers that checkpoint to the `offline_db` directory so that both directories are in sync. This script can also be used anytime to create a checkpoint of the live database.

Please note the warnings about how long this process may take at Section 8.4.6, “live_checkpoint.sh”

This command should be run when an error occurs during offline checkpointing. It restarts the offline checkpoint process from the live database files to bring the offline copy back in sync. If the live checkpoint script fails, contact Perforce Consulting at consulting@perforce.com.

Chapter 7. Maximizing Server Performance

The following sections provide some guidelines for maximizing the performance of the Perforce Helix Core Server, using tools provided by the SDP. More information on this topic can be found in the Knowledge Base.

7.1. Ensure Transparent Huge Pages (THP) is turned off

This is reference KB Article on Platform Notes

There is a (now deprecated) script in the SDP which will do this:

```
/p4/sdp/Server/Unix/setup/os_tweaks.sh
```

It needs to be run as `root` or using `sudo`. This will not persist after system is rebooted - and is thus no longer the recommended option.

We recommend the usage of `tuned` instead of the above, since it will persist after reboots.

Install as appropriate for your Linux distribution (so as `root`):

```
yum install tuned
```

or

```
apt-get install tuned
```

1. Create a customized `tuned` profile with disabled THP. Create a new directory in `/etc/tuned` directory with desired profile name:

   ```
   mkdir /etc/tuned/nothp_profile
   ```

2. Then create a new `tuned.conf` file for `nothp_profile`, and insert the new tuning info:

```
cat <<EOF > /etc/tuned/nothp_profile/tuned.conf
[main]
include= throughput-performance

[vm]
transparent_hugepages=never
EOF
```

3. Make the script executable

```
chmod +x /etc/tuned/nothp_profile/tuned.conf
```

4. Enable `nothp_profile` using the `tuned-adm` command.

```
tuned-adm profile nothp_profile
```

5. This change will immediately take effect and persist after reboots. To verify if THP are disabled or not, run below command:

```
cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
```

7.2. Putting server.locks directory into RAM

The `server.locks` directory is maintained in the $P4ROOT (so `/p4/1/root`) for a running server instance. This directory contains a tree of 0-length files (or 17 byte files in earlier p4d versions) used for lock coordination amongst p4d processes.

This directory can be removed every time the p4d instance is restarted, so it is safe to put it into a tmpfs filesystem (which by its nature does not survive a reboot).

Even on a large installation with many hundreds or thousands of users, this directory will be unlikely to exceed 64M. The files in this directory are 17 or 0 bytes depending on th p4d version; space is needed for inodes.

To do this, first determine if the setting will be global for all p4d servers at your site, or will be determined on a per-server machine basis. If set globally, the per-machine configuration described below MUST be done on all p4d server machines.

This should be done in a scheduled maintenance window.

For each p4d server machine (**all** server machines if you intend to make this a global setting), do the following as user `root`:

1. Create a local directory mount point, and change owner/group to `perforce:perforce` (or `$OSUSER`

if SDP config specifies a different OS user, and whatever group is used):

```
mkdir /hxserverlocks
chown perforce:perforce /hxserverlocks
```

2. Add a line to `/etc/fstab` (adjusting appropriately if `$OSUSER` and group are set to something other than `perforce:perforce`):

```
HxServerLocks   /hxserverlocks  tmpfs
uid=perforce,gid=perforce,size=64M,mode=0700 0 0
```

Note: The `64M` in the above example is suitable for many sites, including large ones. For servers with less available RAM, a smaller value is recommended, but no less than 128K.

If multiple SDP instances are operated on the machine, the value must be large enough for all instances.

1. Mount the storage volume:

```
mount -a
```

2. Check it is looking correct and has correct ownership (`perforce` or `$OSUSER`):

```
df -h
ls -la /hxserverlocks
```

As user `perforce` (or `$OSUSER`), set the configurable `server.locks.dir`. This will be set in one of two ways, depending on whether it was set globally, or on a per-server machine.

First, set the shell environment for your instance:

```
source /p4/common/bin/p4_vars N
```

Replacing `N` with your instance name; `1` by default.

To set `server.locks.dir` globally, do:

```
p4 configure set server.locks.dir="/hxserverlocks${P4HOME}/server.locks"
```

e.g.

```
p4 configure set ${SERVERID}#server.locks.dir=/hxserverlocks${P4HOME}/server.locks
```

If you set this globally (without `serverid#` prefix), then you must ensure that all server machines running p4d, including replicas end edge servers, have a similarly named directory available (or bad things will happen!)

Consider failover options. A failover will, by nature, change the ServerID on a given machine. If `server.locks.dir` is set globally, and all machines have the HxServerLocks configuration done as noted above, then the `server.locks.dir` setting is fully accounted for, and will not cause a problem in a failover situaion.

If `server.locks.dir` is set on a per-machine basis, then you should ensure that every standby server has the same configuration with respect to `server.locks.dir` and the HxServerLocks filesystem as its target server. So any standby servers replicating from a commit server should have the same configuration as the commit server, and any standby servers replicating from an edge server should have the same configuration as the target edge server. For simplicity, using a global setting should be considered.

If you are defining server machine templates (such as an AMI in AWS or with Terraform or similar), the HxServerLoccks configuration can and should be accounted for in the system template.

7.3. Installing monitoring packages

The `sysstat` and `sos` packages are recommended for helping investigate any performance issues on a server.

```
yum install sysstat sos
```

or

```
apt install sysstat sos
```

Then enable it:

```
systemctl enable --now sysstat
```

The reports are text based, but you can use kSar (https://github.com/vlsi/ksar) to visualize the data. If installed before `sosreport` is run, `sosreport` will include the `sysstat` data.

We also recommend `P4prometheus` - https://github.com/perforce/p4prometheus. See Automated script installer for SDP instances which makes it easy to install `node_exporter`, `p4prometheus` and monitoring scripts in the `crontab`

See an example of interpreting prometheus metrics

7.4. Optimizing the database files

The Perforce Server's database is composed of b-tree files. The server does not fully rebalance and compress them during normal operation. To optimize the files, you must checkpoint and restore the server. This normally only needs to be done very few months.

To minimize the size of back up files and maximize server performance, minimize the size of the db.have and db.label files.

7.5. P4V Performance Settings

These are covered in: https://community.perforce.com/s/article/2878

7.6. Proactive Performance Maintenance

This section describes some things that can be done to proactively to enhance scalability and maintain performance.

7.6.1. Limiting large requests

To prevent large requests from overwhelming the server, you can limit the amount of data and time allowed per query by setting the MaxResults, MaxScanRows and MaxLockTime parameters to the lowest setting that does not interfere with normal daily activities. As a good starting point, set MaxScanRows to MaxResults * 3; set MaxResults to slightly larger than the maximum number of files the users need to be able to sync to do their work; and set MaxLockTime to 30000 milliseconds. These values must be adjusted up as the size of your server and the number of revisions of the files grow. To simplify administration, assign limits to groups rather than individual users.

To prevent users from inadvertently accessing large numbers of files, define their client view to be as narrow as possible, considering the requirements of their work. Similarly, limit users' access in the protections table to the smallest number of directories that are required for them to do their job.

Finally, keep triggers simple. Complex triggers increase load on the server.

7.6.2. Offloading remote syncs

For remote users who need to sync large numbers of files, Perforce offers a proxy server. P4P, the Perforce Proxy, is run on a machine that is on the remote users' local network. The Perforce Proxy caches file revisions, serving them to the remote users and diverting that load from the main server.

P4P is included in the Windows installer. To launch P4P on Unix machines, copy the `/p4/common/etc/init.d/p4p_1_init script` to `/p4/1/bin/p4p_1_init`. Then review and customize the script to specify your server volume names and directories.

P4P does not require special hardware but it can be quite CPU intensive if it is working with binary files, which are CPU-intensive to attempt to compress. It doesn't need to be backed up. If the P4P

instance isn’t working, users can switch their port back to the main server and continue working until the instance of P4P is fixed.

Chapter 8. Tools and Scripts

This section describes the various scripts and files provided as part of the SDP package.

8.1. General SDP Usage

This section presents an overview of the SDP scripts and tools, with details covered in subsequent sections.

8.1.1. Linux

Most scripts and tools reside in `/p4/common/bin`. The `/p4/<instance>/bin` directory (e.g. `/p4/1/bin`) contains scripts or links that are specific to that instance such as wrappers for the p4d executable.

Older versions of the SDP required you to always run important administrative commands using the `p4master_run` script, and specify fully qualified paths. This script loads environment information from `/p4/common/bin/p4_vars`, the central environment file of the SDP, ensuring a controlled environment. The `p4_vars` file includes instance specific environment data from `/p4/common/config/p4_instance.vars` e.g. `/p4/common/config/p4_1.vars`. The `p4master_run script` is still used when running p4 commands against the server unless you set up your environment first by sourcing p4_vars with the instance as a parameter (for bash shell: `source /p4/common/bin/p4_vars 1`). Administrative scripts, such as `daily_checkpoint.sh`, no longer need to be called with `p4master_run` however, they just need you to pass the instance number to them as a parameter.

When invoking a Perforce command directly on the server machine, use the p4_***instance*** wrapper that is located in `/p4/instance/bin`. This wrapper invokes the correct version of the p4 client for the instance. The use of these wrappers enables easy upgrades, because the wrapper is a link to the correct version of the p4 client. There is a similar wrapper for the p4d executable, called p4d_***instance***.

This wrapper is important to handle case sensitivity in a consistent manner, e.g. when running a Unix server in case-insensitive mode. If you just execute `p4d` directly when it should be case-insensitive, then you may cause problems, or commands will fail.

Below are some usage examples for instance 1.

Example	*Remarks*
`/p4/common/bin/p4master_run 1 /p4/1/bin/p4_1 admin stop`	Run `p4 admin stop` on instance 1
`/p4/common/bin/live_checkpoint.sh 1`	Take a checkpoint of the live database on instance 1
`/p4/common/bin/p4login 1`	Log in as the perforce user (superuser) on instance 1.

Some maintenance scripts can be run from any client workspace, if the user has administrative access to Perforce.

8.1.2. Monitoring SDP activities

The important SDP maintenance and backup scripts generate email notifications when they complete.

For further monitoring, you can consider options such as:

- Making the SDP log files available via a password protected HTTP server.
- Directing the SDP notification emails to an automated system that interprets the logs.

8.2. Upgrade Scripts

8.2.1. get_helix_binaries.sh

Usage

```
USAGE for get_helix_binaries.sh v1.3.3:

get_helix_binaries.sh [-r <HelixMajorVersion>] [-b <Binary1>,<Binary2>,...] [-n] [-D]

   or

get_helix_binaries.sh -h|-man

DESCRIPTION:
    This script acquires Perforce Helix binaries from the Perforce FTP server.

    The four Helix binaries that can be acquired are:

    * p4, the command line client
    * p4d, the Helix Core server
    * p4p, the Helix Proxy
    * p4broker, the Helix Broker

    This script gets the latest patch of binaries for the current major Helix
    version.  It is intended to acquire the latest patch for an existing install,
    or to get initial binaries for a fresh new install.  It must be run from
    the /hxdepots/sdp/helix_binaries directory (or similar; the /hxdepots
    directory is the default but is subject to local configuration).

    The helix_binaries directory is used for staging binaries for later upgrade
    with the SDP 'upgrade.sh' script (documented separately).  This helix_binaries
    directory is used to stage binaries on the current machine, while the
    'upgrade.sh' script updates a single SDP instance (of which there might be
    several on a machine).

    The helix_binaries directory may not be in the PATH. As a safety feature,
    the 'verify_sdp.sh' will report an error if the 'p4d' binary is found outside
    /p4/common/bin in the PATH. The SDP 'upgrade.sh' check uses 'verify_sdp.sh'
    as part of its preflight checks, and will refuse to upgrade if any 'p4d' is
```

```
    found outside /p4/common/bin.

    When a newer major version of Helix binaries is needed, this script should not
    be modified directly. Instead, the recommended approach is to upgrade the SDP
    to get the latest version of SDP first, which will included a newer version of
    this script, as well as the latest 'upgrade.sh'.  The 'upgrade.sh' script
    is updated with each major SDP version to be aware of any changes in
    the upgrade procedure for the corresponding p4d version.  Upgrading SDP first
    ensures you have a version of the SDP that works with newer versions of p4d
    and other Helix binaries.

OPTIONS:
 -r <HelixMajorVersion>
    Specify the Helix Version, using the short form.  The form is rYY.N, e.g. r21.2
    to denote the 2021.2 release. The default: is r23.1

 -b <Binary1>[,<Binary2>,...]
    Specify a comma-delimited list of Helix binaries. The default is: p4 p4d p4broker
p4p

 -n Specify the '-n' (No Operation) option to show the commands needed
    to fetch the Helix binaries from the Perforce FTP server without attempting
    to execute them.

 -D Set extreme debugging verbosity using bash 'set -x' mode.

HELP OPTIONS:
 -h Display short help message
 -man   Display this manual page

EXAMPLES:
    Note: All examples assume the SDP is in the standard location, /hxdepots/sdp.

    Example 1 - Typical Usage with no arguments:

    cd /hxdepots/sdp/helix_binaries
    ./get_helix_binaries.sh

    This acquires the latest patch of all 4 binaries for the r23.1
    release (aka 2023.1).

    Example 2 - Specifying the major version:

    cd /hxdepots/sdp/helix_binaries
    ./get_helix_binaries.sh -r r21.2

    This gets the latest patch of for the 2021.2 release of all 4 binaries.

    Note: Only supported Helix binaries are guaranteed to be available from the
    Perforce FTP server.
```

```
    Note: Only the latest patch of any given binary is available from the Perforce
    FTP server.

    Example 3 - Sample getting r22.2 and skipping the proxy binary (p4p):

    cd /hxdepots/sdp/helix_binaries
    ./get_helix_binaries.sh -r r22.2 -b p4,p4d,p4broker

DEPENDENCIES:
    This script requires outbound internet access. Depending on your environment,
    it may also require HTTPS_PROXY to be defined, or may not work at all.

    If this script doesn't work due to lack of outbound internet access, it is
    still useful illustrating the locations on the Perforce FTP server where
    Helix Core binaries can be found.  If outbound internet access is not
    available, use the '-n' flag to see where on the Perforce FTP server the
    files must be pulled from, and then find a way to get the files from the
    Perforce FTP server to the correct directory on your local machine,
    /hxdepots/sdp/helix_binaries by default.

EXIT CODES:
    An exit code of 0 indicates no errors were encountered. An
    non-zero exit code indicates errors were encountered.
```

8.2.2. upgrade.sh

The `upgrade.sh` script is used to upgrade `p4d` and other Perforce Helix binaries on a given server machine.

The links for different versions of `p4d` are described in Section A.1.3, “P4D versions and links”

Usage

```
USAGE for upgrade.sh v4.10.7:

upgrade.sh <instance> [-p|-I] [-M] [-Od] [-Osp] [-c] [-y] [-L <log>] [-d|-D]

or

upgrade.sh [-h|-man]


DESCRIPTION:

    This script upgrades the following Helix Core software:

    * p4d, the Perforce Helix Core server
    * p4broker, the Helix Broker server
    * p4p, the Helix Proxy server
    * p4, the command line client
```



```
    Details of each upgrade are described below. Prior to executing any upgrades, a
    preflight check is done to help ensure upgrades will go smoothly.  Also, checks
    are done to determine what (if any) of the above software products need to be
    updated.

    To prepare for an upgrade, new binaries must be update in the
/p4/sdp/helix_binaries
    directory.  This is generally done using the get_helix_binaries.sh script in
    that directory.  Binaries in this directory are not referenced by live running
    servers, and so it is safe to upgrade files in this directory to stage for a
    future upgrade at any time. Also, the SDP standard PATH does not include this
    directory, as verified by the verify_sdp.sh script.

THE INSTANCE BIN DIR

    The 'instance bin' directory, /p4/<instance>/bin, (e.g. /p4/1/bin for instance
    1), is expected to contain *_init scripts for services that operate on the
    given machine.

    For example, a typical master machine for instance 1 might have the following
    in /p4/1/bin:

    * p4broker_1_init script
    * p4broker_1 symlink
    * p4d_1_init script
    * p4d_1 symlink or script
    * p4_1 symlink (a reference to the 'p4' command line client)

    A server machine for instance 1 that runs only the proxy server would have the
    following in /p4/1/bin:

    * p4p_1_init script
    * p4p_1 symlink
    * p4_1 symlink

    The instance bin directory is never modified by the 'upgrade.sh' script.
    The addition of new binaries and update of symlinks occur in .

    The existence of *_init scripts for any given binary determines whether this
    script attempts to manage the service on a given machine, stopping it before
    upgrades, restarting it afterward, and other processing in the case of p4d.

    Note that Phase 2, adding new binaries and updating symlinks, will occur for
    all binaries for which new staged versions are available, regardless of
    whether they are operational on the given machine.

THE COMMON DIR

    This script performs it operations in the SDP common bin dir, .
```

```
    Unlike the instance bin directory, the  directory is expected
    to be identical across all machines in a topology.  Scripts and symlinks
    should always be the same, with only temporary differences while global
    topology upgrades are in progress.

    Thus, all binaries available to be upgraded will be upgraded in Phase 2, even
    if the binary does not operate on the current machine. For example, if a new
    version of 'p4p' binary is available, a new version will be copied to
     and symlink references updated there. However, the p4p binary will
    not be stopped/started.

GENERAL UPGRADE PROCESS

    This script determines what binaries need to be upgraded, based on what new
    binaries are available in the /p4/sdp/helix_binaries directory compared to what
binaries
    the current instance uses.

    There are 5 potential phases. Which phases execute depend on the set of binaries
    being upgraded.  The phases are:

    * PHASE 1 - Establish a clean rollback point.
    This phase executes on the master if p4d is upgraded.

    * PHASE 2 - Install new binaries and update SDP symlinks in .
    This phase executes for all upgrades.

    * PHASE 3 - Stop services to be upgraded.
    This phase executes for all upgrades involving p4d, p4p, p4broker.
    Only a 'p4' client only upgrade skips this phase.

    * PHASE 4 - Perforce p4d schema upgrades
    This step involves the 'p4d -xu' processing. It executes if p4d is upgraded
    to a new major version, and occurs on the master as well as all replicas/edge
    servers. The behavior of 'p4d -xu' differs depending on whether the server is
    the master or a replica.

    This phase is skipped if upgrading to a patch of the same major version, as
    patches do not require 'p4d -xu' processing.

    * PHASE 5 - Start upgraded services.
    This phase executes for all upgrades involving p4d, p4p, p4broker.
    Only a 'p4' client only upgrade skips this phase.

SPECIAL CASE - To OR THRU P4D 2019.1

    If you are upgrading from a version that is older than 2019.1, services
    are NOT restarted after the upgrade in Phase 5, except on the master.
    Services must be restarted manually on all other servers.

    For these 'to-or-thru' 2019.1 upgrades, after ensuring all replicas/edges
```

```
    are caught up (per 'p4 pull -lj'), shutdown all servers other than the
    master.

    Proceeding outer-to-inner, execute this script like so on all machines
    except the master:

    1. Deploy new executables in /p4/sdp/helix_binaries
    2. Stop p4d.
    3. Run 'verify_sdp.sh -skip cron,version'; fix problems if needed until it reports
clean.
    4. Run 'upgrade.sh -M' to update symlinks.
    5. Do the upgrade manually with:  p4d -xu
    6. Leave the server offline.

    On the master, execute like this:

    1. Deploy new executables in /p4/sdp/helix_binaries
    2. Run 'verify_sdp.sh -skip cron,version'; fix problems if needed until it reports
clean.
    3. upgrade.sh

    When the script completes (it will wait for 'p4 storage' upgrades),
    restart services manually after the upgrade in the 'inner-to-outer'
    direction.  Restart services on replicas/edges going inner-to-outer

    This procedure requiring extra steps is specific to 'to-or-thru' P4D 2019.1
    upgrades.  For upgrades starting from P4D 2019.1 or later, things are simpler.

UPGRADES FOR P4D 2019.1+

    For upgrades where the P4D start version is 2019.1 and going to any subsequent
    version, run this script going outer-to-inner. On each machine, it leaves the
    services online and running.  Going in the outer-to-inner direction an all
    servers, do:

    1. Deploy new executables in /p4/sdp/helix_binaries
    2. Run 'verify_sdp.sh -skip cron,version'; fix problems if needed until it reports
clean.
    3. upgrade.sh

UPGRADE PREPARATION

   The steps for deploying new binaries to server machines and running verify_sdp.sh
   (and potentially correcting any issues it discovers) can and should be done before
   the time or even day of any planned upgrade.

UPGRADING HELIX CORE - P4D

    The p4d process, the Perforce Helix Core Server, is the center of the Perforce
    Helix universe, and the only server with a significant database component.
    Most of the upgrade phases above are about performing the p4d upgrade.
```

```
    This 'upgrade.sh' script requires that the 'p4d' service be running at the
    beginning of processing if p4d is to be upgraded, and will abort if p4d is
    not running.

ORDER OF UPGRADES

    Any given Perforce Helix installation will have least one p4d master server, and
    may have several other p4d servers deployed on different machines as replicas and
    edge servers.  When upgrading multiple p4d servers for any given instance (i.e.
    any given data set, with a unique set of changelist numbers and users), the order
    in which upgrades are performed matters.  Upgrades must be done in "outer to
    inner" order.

    The master server, at the center of the topology, is the innermost server and
    must be upgraded last. Any replicas or edge servers connected directly to the
    master constitute the next outer circle.  These can be upgraded in any order
    relative to each other, but must be done before the master and after any
    replicas farther out from the master in the topology.  So this 'upgrade.sh'
    script should be run first on the server machines that are "outermost" from
    the master from a replication perspective, and moving inward. The last run is
    done on the master server machine.

    Server machines running only proxies and brokers do not have a strict order
    dependency for upgrades. These are commonly done in the same "outer to inner"
    methodology as p4d for process consistency rather than strict technical need.

    See the SDP_Guide.Unix.html for more information related to performing global
    topology upgrades.

MASTER JOURNAL ROTATIONS

    This script helps minimize downtime for upgrades by taking advantage of the SDP
    offline checkpoint mechanism.  Rather than wait for a full checkpoint, a journal
    is rotated and replayed to the offline_db.  This typically takes very little
    time compared to a checkpoint, reducing downtime needed for the overall upgrade.

    When the master server is upgraded, two rotations of the master server's journal
    occur during processing.  The first journal rotation occurs before any upgrade
    processing occurs, i.e. before the new binaries are added and symlinks are
    updated. This gives a clean rollback point.

    Later, after the p4d has started and p4d performs its journaled upgrade
    processing, a second journal rotation occurs in Phase 5. This second journal
    rotation captures all upgrade-related processing in a separately numbered journal.

UPGRADING HELIX BROKER

    Helix Broker (p4broker) servers are commonly deployed on the same machine as a
    Helix Core server, and can also be deployed on stand-alone machines (e.g.
    deployed to a DMZ host to provide secure access outside a corporate firewall).
```

```
    Helix Brokers configured in the SDP environment can use a default configuration
    file, and may have other configurations. The default configuration is the done
    defined in /p4/common/config/p4_N.broker.cfg (or a host-specific override file
    if it exists named /p4/common/config/p4_N.broker.<short_hostname>.cfg).  Other
    broker configurations may exist, such as a DFM (Down for Maintenance) broker
    config /p4/common/config/p4_N.broker.dfm.cfg.

    During upgrade processing, this 'upgrade.sh' script only stops and restarts the
    broker with the default configuration. Thus, if coordinating DFM brokers, first
    manually shutdown the default broker and start the DFM brokers before calling
    this script. This script will leave the DFM brokers running while adding the
    new binaries and updating the symlinks.  (Note: Depending on how services
    are configured, this DFM configuration might not survive a machine reboot.
    typically the default broker will come online after a machine reboot).

    This 'upgrade.sh' script will stop the p4broker service if it is running at the
    beginning of processing.  If it was stopped, it will be restarted after the new
    binaries are in place and symlinks are updated. If p4broker was not running at the
    start of processing, new binaries are added and symlinks updated, but the
    p4broker server will not be started.

UPGRADING HELIX PROXY

    Helix Proxy (p4p) are commonly deployed on a machine by themselves, with no p4d
    and no broker. It may also be run on the same machine as p4d.

    This 'upgrade.sh' script will stop the p4p service if it is running at the
    beginning of processing.  If it was stopped, it will be restarted after the new
    binaries are in place and symlinks are updated. If p4p was not running at the
    start of processing, new binaries are added and symlinks updated, but the p4p
    server will not be started.

UPGRADING HELIX P4 COMMAND LINE CLIENT

    The command line client, 'p4', is upgraded in Phase 2 by addition of new
    binaries and updating of symlinks.

STAGING HELIX BINARIES

    If your server can reach the Perforce FTP server over the public internet, a
    script can be used from the /p4/sdp/helix_binaries directory to get the latest
    binaries:

    $ cd /p4/sdp/helix_binaries
    $ ./get_helix_binaries.sh

    If your server cannot reach the Perforce FTP server, perhaps due to outbound
    network firewall restrictions or operating on an "air gapped" network,
    use the '-n' option to see where Helix binaries can be acquired from:
```

```
    $ cd /p4/sdp/helix_binaries
    $ ./get_helix_binaries.sh -n

OPTIONS:
 <instance>
    Specify the SDP instance name to add.  This is a reference to the Perforce
    Helix Core data set. This defaults to the current instance based on the
    $SDP_INSTANCE shell environment variable. If the SDP shell environment is
    not loaded, this option is required.

 -p Specify '-p' to halt processing after preflight checks are complete,
    and before actual processing starts. By default, processing starts
    immediately upon successful completion of preflight checks.

 -Od
    Specify '-Od' to override the rule preventing downgrades.

    WARNING: This is an advanced option intended for use by or with the
    guidance of Perforce Support or Perforce Consulting.

 -Osp
    Specify '-Osp' to override the sudo preflight, skipping that check.

    WARNING: This is an advanced option intended for use by or with the
    guidance of Perforce Support or Perforce Consulting.

 -I Specify '-I' to ignore preflight errors. Use of this flag is STRONGLY
    DISCOURAGED, as the preflight checks are essential to ensure a safe
    and smooth migration. If used, preflight checks are still done so
    their errors are recorded in the upgrade log, and then the migration
    will attempt to proceed.

    WARNING: This is an advanced option intended for use by or with the
    guidance of Perforce Support or Perforce Consulting.

 -M Specify '-M' if you plan to do a manual upgrade. With this option,
    only Phase 2 processing, adding new staged binaries and updating
    symlinks, is done by this script.

    WARNING: This is an advanced option intended for use by or with the
    guidance of Perforce Support or Perforce Consulting.

 -c Specify '-c' to execute a command to upgrade the Protections table
    comment format after the p4d upgrade, by using a command like:

        p4 protect --convert-p4admin-comments -o | p4 -s protect -i

    By default, this Protections table conversion is not performed.  In some
    environments with custom policies related to update of the protections
    table, this command may not work.
```

```
    The new style of comments and the '--convert-p4admin-comments' option
    was introduced in P4D 2016.1.

 -L <log>
    Specify the path to a log file, or the special value 'off' to disable
    logging.  By default, all output (stdout and stderr) goes to this file
    in the /p4/N/logs directory (where N is the SDP instance name):

    upgrade.p4_N.<datestamp>.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.'

    Logging can only be disabled with '-L off' if the '-n' or '-p' flags
    are used. Disabling logging for actual upgrades is not allowed.

 -y Specify the '-y' option to confirm that the upgrade should be done.

    By default, this script operates in No-Op mode, meaning no actions
    that affect data or structures are taken.  Instead, commands that would
    be run are displayed.  This mode can be educational, showing various
    steps that will occur during an actual upgrade.

DEBUGGING OPTIONS:
 -d     Increase verbosity for debugging.

 -D     Set extreme debugging verbosity, using bash '-x' mode. Also implies -d.

HELP OPTIONS:
 -h Display short help message
 -man   Display man-style help message

EXAMPLES:
    EXAMPLE 1: Preflight Only

    To see if an upgrade is needed for this instance, based on binaries
    staged in /p4/sdp/helix_binaries, use the '-p' flag to execute only the preflight
    checks, and disable logging, as in this example:

    $ cd /p4/common/bin
    $ ./upgrade.sh 1 -p -L off

    EXAMPLE 2: Typical Usage

    Typical usage is with just the SDP instance name as an argument, e.g.
    instance '1', and no other parameters, as in this example:

    $ cd /p4/common/bin
    $ ./upgrade.sh 1
```

```
    This first runs preflight checks, and aborts if preflight checks
    detected any issues. The it gives a preview of the upgrade. A
    successful preview completes with a line near the end that looks like
    this sample:

    Success: Finished p4_1 Upgrade.

    If the preview is successful, then proceed with the real upgrade using
    the -y flag:

    $ ./upgrade.sh 1 -y

    EXAMPLE 3: Simplified

    If the standard SDP shell environment is loaded, upgrade.sh will be in
    the path, so the 'cd' command to /p4/common/bin is not needed. Also,
    the SDP_INSTANCE shell environment variable will be defined, so the
    'instance' parameter can be dropped, with simply a call to the script
    needed.  First do a preview:

    $ upgrade.sh

    Review the output of the preview, looking for the 'Success: Finished'
    message near the end of the output. If that exists, then execute again
    with the '-y' flag to execute the actual migration:

    $ upgrade.sh -y

CUSTOM PRE- AND POST- UPGRADE AUTOMATION HOOKS:
    This script can execute custom pre- and post- upgrade scripts. This
    can be useful to incorporate site-specifc elements of an upgrade.

    If the file /p4/common/site/upgrade/pre-upgrade.sh exists and is
    executable, it will be executed as a pre-upgrade script. If the file
    /p4/common/site/upgrade/post-upgrade.sh exists and is executable,
    it will be executed as a post-ugprade script.

    Pre- and post- upgrade scripts are called with an SDP instance
    parameter, and an optional '-y' flag to confirm actual processing is
    to be done.  Custom scripts are expected to operate in preview mode
    by default, taking no actions that affect data (just as this script
    behaves).  If this upgrade.sh script is given the '-y' flag, that
    option is passed to the custom script as well, indicating active
    processing should occur.

    Pre- and post- upgrade scripts are expected to exit with a zero exit
    code to indicate success, and non-zero to indicate failure.

    The custom pre-upgrade script is executed after standard preflight
    checks complete successfully.  If the '-I' flag is used to ignore the
```

```
    status of preflight checks, the custom pre-upgrade script is
    executed regardless of the status of preflight checks. Preflight
    checks are executed before actual upgrade processing commences. If a
    custom pre-upgrade script indicates a failure, the overall upgrade
    process aborts.

    The post-upgrade custom script is executed after the main upgrade
    is successful.

    Success or failure of pre- and post- upgrade scripts is reported in
    the log.  These scripts do not require independent logging, as all
    standard and error output is captured in the log of this upgrade.sh
    script.

    TIP: Be sure to fully test custom scripts in a test environment
    before incorporating them into an upgrade on production systems.

SEE ALSO:
    The /verify_sdp.sh script is used for preflight checks.

    The /p4/sdp/helix_binaries/get_helix_binaries.sh script acquires new binaries
    for upgrades.

    Both scripts sport the same '-h' (short help) and '-man' (full manual)
    usage options as this script.

LIMITATIONS:
    This script does not handle upgrades of 'p4dtg', Helix Swarm,
    Helix4Git, or any other software. It only handles upgrades of
    p4d, p4p, p4broker, and the p4 client binary on the SDP-managed
    server machine on which it is executed.
```

8.2.3. sdp_upgrade.sh

This script will perform an upgrade of the SDP itself - see Section 6.3, “Upgrading the SDP”

Usage

```
USAGE for sdp_upgrade.sh v1.7.6:

   sdp_upgrade.sh [-y] [-p] [-L <log>|off] [-D]

   or

sdp_upgrade.sh -h|-man

This script must be executed from the 'sdp_upgrade' directory in the extracted
SDP tarball.

Typical operation starts like this:
```

```
cd /hxdepots/downloads/new/sdp/Server/Unix/p4/common/sdp_upgrade
./sdp_upgrade.sh -h


DESCRIPTION:

    This script upgrades Perforce Helix Server Deployment Package (SDP) from
    SDP 2020.1 to the version included in the latest SDP version, SDP
    2022.2.

    == Pre-Upgrade Planning ==

    This script will upgrade the SDP if the pre-upgrade starting SDP version
    is SDP 2020.1 or later, including any/all patches of SDP
    2020.1.

    If the current SDP version is older than 2020.1, it must first be upgraded
    to SDP 2020.1 using the SDP Legacy Upgrade Guide.  For upgrading from
    pre-20.1 versions dating back to 2007, in-place or migration-style upgrades
    can be done.  See:

    https://swarm.workshop.perforce.com/projects/perforce-software-
sdp/view/main/doc/SDP_Legacy_Upgrades.Unix.html

    The SDP should always be upgraded to the latest version first before
    Helix Core binaries p4d/p4broker/p4p are upgraded using the SDP
    upgrade.sh script.

    Upgrading the SDP first ensures the version of the SDP you have
    is compatible with the latest versions of p4d/p4broker/p4p/p4, and
    will always be compatible with all supported versions of these
    Helix Core binaries.

    When this script is used, i.e. when the current SDP version is 2020.1
    or newer, the SDP upgrade procedure does not require downtime for any
    running Perforce Helix services, such as p4d, p4broker, or p4p.  This
    script is safe to run in environments where live p4d instances are running,
    and does not require p4d, p4broker, p4p, or any other services to be stopped
    or upgraded.  Upgrade of the SDP is cleanly separate from the upgrade the
    Helix Core binaries. The upgrade of the SDP can be done immediately prior to
    Helix Core upgrades, or many days prior.

    There can be multiple SDP instances on a given server machine. This script
    will upgrade the SDP on the machine, and thus after the upgrade all
    instances will immediately use new SDP scripts and updated instance
    configuration files, e.g. the /p4/common/config/p4_N.vars files.  However,
    all instances will continue running the same Helix Core binaries.  Any live
    running Helix Core server process on the machine are unaffected by the
    upgrade of SDP.
```

```
    This script will upgrade the SDP on a single machine. If your Perforce
    Helix topology has multiple machines, the SDP should be upgraded on all
    machines. The upgrade of SDP on multiple machines can be done in any order,
    as there is no cross-machine dependency requiring the SDP to be the same
    version.  (The order of upgrade of Helix Core services and binaries such as
    p4d in global topologies with replicas and edge servers does matter, but is
    outside the scope of this script).

    Planning Recap:
    1. The SDP can be upgraded without downtime when this script is used, i.e.
    when the starting SDP version is 2020.1 or later.
    2. Upgrade SDP on all machines, in any order, before upgrading p4d and other
    Helix binaries.

    == NFS Sharing of HxDepots ==

    In some environments, the HxDepots volume is shared across multiple server
    machines with NFS, typically mounted as /hxdepots.  This script updates the
    /hxdepots/p4/common and /hxdepots/sdp directories, both of which are on the
    NFS mount.  Thus upgrading SDP on a single machine will effectively and
    immediately upgrade the SDP on all machines that share /hxdepots from the
    same NFS-mounted storage. This is a safe and valid configuration, as
    upgrading the SDP does not affect any live running p4d servers.

    == Acquiring the SDP Package ==

    This script is part of the SDP package (tarball). It must be run from an
    extracted tarball directory.  Acquiring the SDP tarball is a manual operation.

    The SDP tarball must be extracted such that the 'sdp' directory appears as
    <HxDepots>/downloads/new/sdp, where <HxDepots> defaults to /hxdepots. To
    determine the value for <HxDepots> at your site you can run the following:

     bash -c 'cd /p4/common; d=$(pwd -P); echo ${d%/p4/common}'

   On this machine, that value is: /hxdepots

    Following are sample commands to acquire the latest SDP, to be executed
    as the user perforce:

      cd /hxdepots
      [[ -d downloads ]] || mkdir downloads
      cd downloads
      [[ -d new ]] && mv new old.$(date +'%Y%m%d-%H%M')
      curl -s -k -O https://swarm.workshop.perforce.com/projects/perforce-software-
sdp/download/downloads/sdp.Unix.tgz
      mkdir new
      cd new
      tar -xzf ../sdp.Unix.tgz

    After extracting the SDP tarball, cd to the directory where this
```

```
    sdp_upgrade.sh script resides, and execute it from there.

      cd /hxdepots/downloads/new/sdp/Server/Unix/p4/common/sdp_upgrade
      ./sdp_upgrade.sh -man

    == Preflight Checks ==

    Prior to upgrading, preflight checks are performed to ensure the
    upgrade can be completed successfully. If the preflight checks
    fail, the upgrade will not start.

    Sample Preflight Checks:

    * The existing SDP version is verified to be SDP 2020.1+.
    * Various basic SDP structural checks are done.
    * The /p4/common/bin/p4_vars is checked to confirm it can be upgraded.
    * All /p4/common/config/p4_N.vars files are checked to confirm they can be
upgraded.

    == Automated Upgrade Processing ==

    Step 1: Backup /p4/common.

    The existing <HxDepots>/p4/common structure is backed up to:

    <HxDepots>/p4/common.bak.<YYYYMMDD-hhmm>

    Step 2: Update /p4/common.

    The existing SDP /p4/common structure is updated with new
    versions of SDP files.

    Step 3: Generate the SDP Environment File.

    Regenerate the SDP general environment file,
    /p4/common/bin/p4_vars.

    The template is /p4/common/config/p4_vars.template.

    Step 4: Generate the SDP Instance Files.

    Regenerate the SDP instance environment files for all instances based on
    the new template.

    The template is /p4/common/config/instance_vars.template.

    For Steps 3 and 4, the re-generation logic will preserve current
    settings. If upgrading from SDP r20.1, any custom logic that
    exists below the '### MAKE LOCAL CHANGES HERE' tag will be
    split into separate files.  Custom logic in p4_vars will be moved
    to /p4/common/site/config/p4_vars.local. Custom logic in
```

```
p4_N.vars files will be moved to /p4/common/site/config/p4_N.vars.local.

Note: Despite these changes, the mechanism for loading the SDP shell
environment remains unchanged since 2007, so it looks like:

$ source /p4/common/bin/p4_vars N

Changes to the right-side of assignments for specific are preserved
for all defined SDP settings.  For p4_vars, preserved settings are:
  - OSUSER (determined by current owner of /p4/common)
  - KEEPLOGS
  - KEEPCKPS
  - KEEPJNLS

For instance_vars files, preserved settings are:
  - MAILTO
  - MAILFROM
  - P4USER
  - P4MASTER_ID
  - SSL_PREFIX
  - P4PORTNUM
  - P4BROKERPORTNUM
  - P4MASTERHOST
  - PROXY_TARGET
  - PROXY_PORT
  - PROXY_V_FLAGS
  - P4DTG_CFG
  - SNAPSHOT_SCRIPT
  - SDP_ALWAYS_LOGIN
  - SDP_AUTOMATION_USERS
  - SDP_MAX_START_DELAY_P4D
  - SDP_MAX_START_DELAY_P4BROKER
  - SDP_MAX_START_DELAY_P4P
  - SDP_MAX_STOP_DELAY_P4D
  - SDP_MAX_STOP_DELAY_P4BROKER
  - SDP_MAX_STOP_DELAY_P4P
  - VERIFY_SDP_SKIP_TEST_LIST
  - The 'umask' setting.
  - KEEPLOGS (if set)
  - KEEPCKPS (if set)
  - KEEPJNLS (if set)

Note that the above list excludes any values that are calculated.

Step 5: Remove Deprecated Files.

Deprecated files will be purged from the SDP structure.  The list of
files to be cleaned are listed in this file:

/hxdepots/downloads/new/sdp/Server/Unix/p4/common/sdp_upgrade/deprecated_files.txt
```

```
    Paths listed in this file are relative to the '/p4' directory (or
    more accurately the SDP Install Root directory, which is always
    '/p4' except in SDP test production environments).

    Step 6: Update SDP crontabs.

    No crontab updates are required for this SDP upgrade.

    == Post-Upgrade Processing ==

    This script provides guidance on any post-processing steps. For some
    releases, this may include upgrades to crontabs.

OPTIONS:
-y  Specify the '-y' option to confirm that the SDP upgrade should be done.

    By default, this script operates in No-Op mode, meaning no actions
    that affect data or structures are taken.  Instead, commands that would
    be run are displayed.  This mode can be educational, showing various
    steps that will occur during an actual upgrade.

 -p Specify '-p' to halt processing after preflight checks are complete,
    and before actual processing starts. By default, processing starts
    immediately upon successful completion of preflight checks.

 -L <log>
    Specify the log file to use.  The default is /tmp/sdp_upgrade.<timestamp>.log

    The special value 'off' disables logging to a file. This cannot be
    specified if '-y' is used.

 -d Enable debugging verbosity.

 -D Set extreme debugging verbosity.

HELP OPTIONS:
 -h Display short help message
 -man   Display man-style help message

FILES AND DIRECTORIES:
    Name: SDPCommon
    Path: /p4/common
    Notes: This sdp_upgrade.sh script updates files in and under this folder.

    Name: HxDepots
    Default Path: /hxdepots
    Notes: The folder containing versioned files, checkpoints, and numbered
    journals, and the SDP itself. This is commonly a mount point.

    Name: DownloadsDir
    Default Path: /hxdepots/downloads
```

```
    Name: SDPInstallRoot
    Path: /p4

EXAMPLES:
    This script must be executed from 'sdp_upgrade' directory in the extracted
    SDP tarball.  Typical operation starts like this:

    cd /hxdepots/downloads/new/sdp/Server/Unix/p4/common/sdp_upgrade
    ./sdp_upgrade.sh -h

    All following examples assume operation from that directory.

    Example 1: Prelight check only:

    sdp_upgrade.sh -p

    Example 2: Preview mode:

    sdp_upgrade.sh

    Example 3: Live operation:

    sdp_upgrade.sh -y

LOGGING:
    This script generates a log file, ~/sdp_upgrade.<timestamp>.log
    by default. See the '-L' option above.

CUSTOM PRE- AND POST- UPGRADE AUTOMATION HOOKS:
    This script can execute custom pre- and post- upgrade scripts. This
    can be useful to incorporate site-specific elements of an SDP upgrade.

    If the file /p4/common/site/upgrade/pre-sdp_upgrade.sh exists and is
    executable, it will be executed as a pre-upgrade script. If the file
    /p4/common/site/upgrade/post-sdp_upgrade.sh exists and is executable,
    it will be executed as a post-upgrade script.

    Pre- and post- upgrade scripts are passed the '-y' flag to confirm
    actual processing is to be done.  Custom scripts are expected to
    operate in preview mode by default, taking no actions that affect data
    (just as this script behaves).  If this sdp_upgrade.sh script is given
    the '-y' flag, that option is passed to the custom script as well,
    indicating active processing should occur.

    Pre- and post- upgrade scripts are expected to exit with a zero exit
    code to indicate success, and non-zero to indicate failure.
    The custom pre-upgrade script is executed after standard preflight
    checks complete successfully.  Preflight checks are executed before
    actual upgrade processing commences. If a custom pre-upgrade script
    indicates a failure, the overall upgrade process aborts.
```

```
    The post-upgrade custom script is executed after the main SDP upgrade
    is successful.

    Success or failure of pre- and post- upgrade scripts is reported in
    the log.  These scripts do not require independent logging, as all
    standard and error output is captured in the log of this sdp_upgrade.sh
    script.

    TIP: Be sure to fully test custom scripts in a test environment
    before incorporating them into an upgrade on production systems.

EXIT CODES:
    An exit code of 0 indicates no errors were encountered during the
    upgrade. A non-zero exit code indicates the upgrade was aborted
    or failed.
```

8.3. Legacy Upgrade Scripts

8.3.1. clear_depot_Map_fields.sh

The `clear_depot_Map_fields.sh` script is used when upgrading to SDP from versions earlier than SDP 2020.1. Its usage is discussed in SDP Legacy Upgrade Guide (for Unix).

Usage

```
USAGE for clear_depot_Map_fields.sh v1.2.0:

clear_depot_Map_fields.sh [-i <instance>] [-L <log>] [-v<n>] [-p|-n] [-D]

or

clear_depot_Map_fields.sh [-h|-man|-V]


DESCRIPTION:
    This script obsoletes the SetDefaultDepotSpecMapField.py trigger.

    It does so by following a series of steps.  First, it ensures that
    the configurable server.depot.root is set correctly, setting it
    if it is not already set.

    Next, the Triggers table is checked to ensure the call to the
    SetDefaultDepotSpecMapField.py is not called; it is deleted from
    the Triggers table if found.
    Last, it resets the 'Map:' field of depot specs for depot
    types where that is appropriate, setting it to the default value of
    '<DepotName>/...', so that it honors the server.depot.root
    configurable.  This is done for depots of these types:
```

```
    * stream
    * local
    * spec
    * unload
    * graph

    but not these:
    * archive
    * remote

    If an unknown depot type is encountered, the 'Map:' field is reset
    as well if it is set.

    This script does a preflight check first, reporting any cases
    where the starting conditions are not as expected.  These conditions
    are treated as Errors, and will abort processing:

    * Depot Map field set to something other than the default.
    * Configurable server.depot.root is set, but to something other
    than what it should be.

    The following are treated as Warnings, and will be reported but
    will not prevent processing.

    * Configurable server.depot.root is already set.
    * SetDefaultDepotSpecMapField.py not found in triggers.
    * Depot already has 'Map:' field set to the default value:
    <DepotName>/...

OPTIONS:
 -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.  By default, all output (stdout and stderr) goes to
    EDITME_DEFAULT_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.'

 -p Run preflight checks only, and then stop. By default, actual changes
    occur if preflight checks find no issues.

 -n No-Op.  No actions are taken that would affect data significantly;
    instead commands are displayed rather than executed.

 -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:
    A typical flow for this script is to do a preflight first, and then
    a live run, for any given instance:
    clear_depot_Map_fields.sh -i 1 -p
    clear_depot_Map_fields.sh -i 1

    Note that if using '-n', the '-v5' flag should also be used.
```

8.4. Core Scripts

The core SDP scripts are those related to checkpoints and other scheduled operations, and all run from `/p4/common/bin`.

If you `source /p4/common/bin/p4_vars <instance>` then the `/p4/common/bin` directory will be added to your $PATH.

8.4.1. p4_vars

The `/p4/common/bin/p4_vars` defines the SDP shell environment, as required by the Perforce Helix server process. This script uses a specified instance number as a basis for setting environment variables. It will look for and open the respective p4_<instance>.vars file (see next section).

This script also sets server logging options and configurables.

It is intended to be used by other scripts for common environment settings, and also by users for setting the environment of their Bash shell.

Usage

```
source /p4/common/bin/p4_vars 1
```

See also: Section 4.4, “Setting your login environment for convenience”

8.4.2. p4_<instance>.vars

Defines the environment variables for a specific instance, including P4PORT etc.

This script is called by Section 8.4.1, “p4_vars” - it is not intended to be called directly by a user.

For instance `1`:

```
p4_1.vars
```

For instance `art`:


```
p4_art.vars
```

Occasionally you may need to edit this script to update variables such as `P4MASTERHOST` or similar.

Location: /p4/common/config

8.4.3. p4master_run

The `/p4/common/bin/p4master_run` is a wrapper script to other SDP scripts. This ensures that the shell environment is loaded from `p4_vars` before executing the script. It provides a '-c' flag for silent operation, used in many crontab so that email is sent from the scripts themselves.

This is especially useful for calling scripts that do not set their own shell environment, such as Python or Perl scripts. Historically it was used as a wrapper for all SDP scripts.

Many of the bash shell scripts in the SDP set their own environment (by doing `source /p4/common/bin/p4_vars N` for their instance); those bash shell scripts do **not** need to be called with the `p4master_run` wrapper.

8.4.4. daily_checkpoint.sh

The `/p4/common/bin/daily_checkpoint.sh` script configured by default to run six days a week using crontab. The script:

- truncates the journal
- replays it into the `offline_db` directory
- creates a new checkpoint from the resulting database files
- recreates the `offline_db` database from the new checkpoint.

This procedure rebalances and compresses the database files in the `offline_db` directory.

These can be rotated into the live (`root`) database, by the script Section 8.4.10, “refresh_P4ROOT_from_offline_db.sh”

Usage

```
/p4/common/bin/daily_checkpoint.sh <instance>
/p4/common/bin/daily_checkpoint.sh 1
```

8.4.5. recreate_offline_db.sh

The `/p4/common/bin/recreate_offline_db.sh` recovers the offline_db database from the latest checkpoint and replays any journals since then. If you have a problem with the offline database then it is worth running this script first before running Section 8.4.6, “live_checkpoint.sh”, as the latter will stop the server while it is running, which can take hours for a large installation.

Run this script if an error occurs while replaying a journal during daily checkpoint process.

This script recreates offline_db files from the latest checkpoint. If it fails, then check to see if the most recent checkpoint in the `/p4/<instance>/checkpoints` directory is bad (ie doesn't look like the right size compared to the others), and if so, delete it and rerun this script. If the error you are getting is that the journal replay failed, then the only option may be to run Section 8.4.6, "live_checkpoint.sh" script.

Usage

```
/p4/common/bin/recreate_offline_db.sh <instance>
/p4/common/bin/recreate_offline_db.sh 1
```

8.4.6. live_checkpoint.sh

The `/p4/common/bin/live_checkpoint.sh` is used to initialize the SDP `offline_db`. It must be run once, typically manually during initial installation, before any other scripts that rely on the `offline_db` can be used, such as `daily_checkpoint.sh` and `rotate_journal.sh`.

This script can also be used in some cases to repair the `offline_db` if it has has become corrupt, e.g. due to a sudden power loss while checkpoint processing was running.

Be aware this script locks the live database for the duration of the checkpoint which can take hours for a large installation (please check the `/p4/1/logs/checkpoint.log` for the most recent output of `daily_checkpoint.sh` to see how long checkpoints take to create/restore).

Note that when a `live_checkpoint.sh` runs, the server will be unresponsive to users for a time. On a new installation this "hang time" will be imperceptible, but over time it can grow to minutes and eventually hours. The idea is that `live_checkpoint.sh` should be used only very sparingly, and is not scheduled as a routine operation.

This performs the following actions:

- Does a journal rotation, so the active P4JOURNAL file becomes numbered.
- Creates a checkpoint from the live database db.* files in the P4ROOT.
- Recovers the `offline_db` database from that checkpoint to rebalance and compress the files

Run this script when creating the server instance and if an error occurs while replaying a journal during the off-line checkpoint process.

Usage

```
/p4/common/bin/live_checkpoint.sh <instance>
/p4/common/bin/live_checkpoint.sh 1
```

8.4.7. p4verify.sh

The `/p4/common/bin/p4verify.sh` script verifies the integrity of the 'archive' files, all versioned files in your repository. This script is run by crontab on a regular basis, typically weekly.

It verifies both shelves and submitted archive files

Any errors in the log file (e.g. `/p4/1/logs/p4verify.log`) should be handled according to KB articles:

- MISSING! errors from p4 verify
- BAD! error from p4 verify

If in doubt contact support-helix-core@perforce.com

Our recommendation is that you should expect this to be without error, and you should address errors sooner rather than later. This may involve obliterating unrecoverable errors.

when run on replicas, this will also append the `-t` flag to the `p4 verify` command to ensure that MISSING files are scheduled for transfer. This is useful to keep replicas (including edge servers) up-to-date.

Usage

```
/p4/common/bin/p4verify.sh <instance>
/p4/common/bin/p4verify.sh 1
```

```
USAGE for  v5.13.3:

p4verify.sh [<instance>] [-N] [-nu] [-nr] [-ns] [-nS] [-a] [-nt] [-nz] [-o
BAD|MISSING] [-chunks <ChunkSize>|-paths <paths_file>] [-w <Wait>] [-q
<MaxActivePullQueueSize>] [-Q MaxTotalPullQueueSize] [-recent] [-dlf
<depot_list_file>] [-I|-ignores <regex_ignores_file>] [-Ocache] [-n] [-L <log>] [-v]
[-d] [-D]

   or

p4verify.sh -h|-man

DESCRIPTION:

    This script performs a 'p4 verify' of all submitted and shelved versioned
    files in depots of all types except 'remote' and 'archive' type depots.

    If run on a replica, it schedules archive failures for transfer to the
    replica.

OPTIONS:
<instance>
    Specify the SDP instance.  If not specified, the SDP_INSTANCE
    environment variable is used instead.  If the instance is not
    defined by a parameter and SDP_INSTANCE is not defined, p4verify.sh
    exists immediately with an error message.

 -N Specify '-N' (Notify Only On Failure) to disable the default behavior
```

```
    which will always send a notification which includes a report of the p4
    verify status.  Specifying '-N' which change the behavior to only send
    a notification if there is an error during the p4 verify execution.
    Notification methods are email, AWS SNS, and PagerDuty. Details on
    configuration can be found in the SDP documentation.  Providing the
    environment variable NOTIFY_ONLY_ON_FAILURE=1 is equivalent to the '-N'
    command line argument.

 -nu    Specify '-nu' (No Unload) to skip verification of the singleton depot
    of type 'unload' (if created).  The 'unload' depot is verified
    by default.

 -nr    Specify '-nr' (No Regular) to skip verification of regular submitted
    archive files.  The '-nr' option is not compatible with '-recent'.
    Regular submitted archive files are verified by default.

 -ns    Specify '-ns' (No Spec Depot) to skip verification of singleton depot
    of type 'spec' (if created). The 'spec' depot is verified by default.

 -nS    Specify '-nS' (No Shelves) to skip verification of shelved archive
    files, i.e. to skip the 'p4 verify -qS'.

 -a Specify '-a' (Archive Depots) to do verification of depots of type
    'archive'.  Depots of type 'archive' are not verified by default, as
    archive depots are often physically removed from the server's
    storage subsystem for long-term cold storage.

-nt Specify the '-nt' option to avoid passing the '-t' flag to 'p4 verify'
    on a replica. By default, p4verify.sh detects if it is running on a
    replica, and if so automatically applies the '-t' flag to 'p4 verify'.
    That causes the replica to attempt to self-heal, as files that fail
    verification are scheduled for transfer from the P4TARGET server. This
    default behavior results in 'Transfer scheduled' messages in the log,
    and MISSING/BAD files are listed as 'info:' rather than 'error:'. There
    is no clear indication of whether or which of the scheduled transfers
    complete successfully, and so there may be a mix of transient/correctable
    and "real"/persistent transfer errors for files that are also BAD/MISSING
    on the master server.  Specify '-nt' to ensure the log contains a list
    of files that currently fail a 'p4 verify' without attempting to transfer
    them from the master.

 -nz    Specify '-nt' to skip the gzip of the old log file.  By default, if a
    log with the default name or the name specified with '-L' exists at the
    start of processing, the old log is rotated and gzipped.  With this option
    the old log is not zipped when rotated.

 -o BAD|MISSING
    Specify '-o MISSING' to check only whether expected archive files exist
    or not, skipping the checksum calculation of existing files.  This results
    in dramatically faster, if less comprehensive, verification.  This
    is particularly well suited when verification is being used to schedule
```

```
    archive file transfers of missing files on replicas.  This translates into
    passing the '--only MISSING' option to 'p4 verify'.

    Specify '-o BAD' to check only for BAD revisions.  This translates into
    passing the '--only BAD' option to 'p4 verify'.

    This option requires p4d to be 2021.1 or newer. For older p4d versions,
    this option is silently ignored.

-chunks <ChunkSize>
    Specify the maximum amount of content by size to verify at once.  If
    this is specified, the depot_verify_chunks.py script is used to
    break up depots into chunks of a given size, e.g. 100M or 4G.

    The <ChunkSize> parameter must be a size value valid to pass to the
    depot_verify_chunks.py script with the '-m' option.  That is,
    specifying '-chunks 200M' translates to calling depot_chunks_verify.sh
    with '-m 200M'.

    This requires the perforce-p4python3 module to be installed and the
    python3 in the PATH must be the correct one that uses the P4 module.

    Using '-chunks' is likely to result in a significantlly slower overall
    verify operation, though chuking can make it less impactful when it
    runs.  Using the '-chunks' option may be necessary on very large data
    sets, e.g. if there insufficient resources to process the largest
    depots.

    The '-recent' and '-chunks' options are mutually exclusive.

    The '-chunks' and '-paths' options can be used together; see the
    description of the '-paths' option below.

    Chunking logic applies only in depots of type 'stream' or 'local'.

-paths <paths_file>

    Specify a file containing a list of depot paths to verify, with one
    line per entry.  Valid entries in the file start with '//', e.g.

    //mydepot/main/src/...

    In this example, when //mydepot depot is processed, only specified
    paths will be verified.  All other depots will be processed in full.
    To verify only specified paths, combine '-paths <paths_file>' with
    '-dlf <depot_list_file>' where the depot list file contains only
    'mydepot' (per the example above).

    The '-recent' and '-paths' options are mutually exclusive.

    The '-chunks' and '-paths' options can be used together for combined
```

```
    effects. If both options are specified, depots that contain specified
    paths are chunked based on the specified paths rather than the entire
    depot, and other paths in that depot are not processed.  Depots that
    do not have any specified paths listed in the <paths_file> are
    chunked at the top/depot level directory.

    Paths specified must be in depots of type 'stream' or 'local'.

 -w <Wait>
    Specify the '-w' option, where <Wait> is a positive integer
    indicating the number of seconds to sleep between individual calls
    to 'p4 verify' commands.  For example, specifying '-w 300' results
    in a delay of 5 minutes between verify commands.

    This can be used with '-chunks' to inject a delay between chunked
    depot paths. Otherwise, the delay is injected between each depot
    processed. This can significantly lengthen the overall duration
    of 'p4verify.sh' operation, but can also spread out the resource
    consumption load on a server machine.

    If shelves are processed (regardless of whether '-chunks' is used),
    the delay is injected between each individual shelved changelist, as
    shelved changes are verified one changelist at a time.  For data sets
    with a large number of shelves, it may be be wise to process shelves
    separately from submitted files if '-w' is used, a delay value to
    apply between depots may be different from that applied to
    individual changelists.

    See the '-q' option for a description of how '-q' and '-w' can be
    used together.

-q <MaxActivePullQueueSize>
    Specify the '-q' option, where <MaxActivePullQueueSize> is a positive
    integer indicating the maximum number of active pulls allowed before
    a 'p4 verify' command will be executed to transfer archives.

    The absolute maximum number of possible active pulls is affected by
    the number of 'startup.N' threads configured to pull archives files,
    and whether those threads indicate batching.

    The threads that pull archive files are those that configured to use
    the 'pull' command the '-u' option. Typically, a small number of pull
    threads are configured, between 2 and 10 or perhaps 20.

    If '-q 1' is specified, new 'p4 verify' commands will only be run
    when the active pull queue is quiet.  Specifying a too-high value,
    e.g. '-q 50' if only 3 'pull -u' archive pull threads are configured,
    will be ineffective, as the active pull threads will never exceed
    3 (let alone 50).

    The current active pull queue on a replica is reported by:
```



```
    p4 -ztag -F %replicaTransfersActive% pull -ls

    This option can be useful if using this p4verify.sh script to pull
    many or even all archives on a new replica server machine from its
    target server.  The injected delays can give the server time to transfer
    archives scheduled in one call to 'p4 verify' before calling it again
    with the goal of avoidng overloading the pull queue.

    If '-w' and '-q' options are both used, the delay specified by '-w'
    is ignored unless the active pull queue size is greater than or equal
    to the specified maximum active pull queue size.  The '-w' then
    essentially determines how frequently the 'p4 pull -ls' is run to
    check the active pull queue size.  A reasonable set of values might
    be '-q 1 -w 10'.

    The '-q' option in mutually exclusive with '-nt'.

    The '-q' option in mutually exclusive with '-Q'.

-Q <MaxTotalPullQueueSize>
    Specify the '-Q' option, where <MaxTotalPullQueueSize> is a positive
    integer indicating the maximum number of total pulls allowed before
    a 'p4 verify' command will be executed to transfer archives.

    In certain scenarios, the pull queue can become quite massive. For
    example, if a fresh standby replica is seeded from a checkpoint
    but has no archive files, and then a 'p4verify.sh' is run, the
    verify will schedule all files to be transferred, perhaps millions.

    If the pull queue gets too large, it can impact metadata replication.
    Setting this value may help mitigate issues related to scheduling
    too many archives pulls at once, by delaying scheduling new archive
    pulls until enough previously scheduled pulls are completed.

    This option can be useful in such scenarios, if this p4verify.sh script
    is used to pull many or even all archives on a new replica server machine
    from its target server.  The injected delays can give the server time to
    transfer archives scheduled in one call to 'p4 verify' before calling it
    again with the goal of avoidng overloading the pull queue.

    If individual depots contain large numbers of files, such that
    a verify on a single depot will schedule too many files to be
    transferred at once, it may be necessary to combine this option with
    the '-chunks' option to avoid overloading the transfer queue.

    **WARNING**: If there are files that cannot be tranferred from the
    replica's target server, the value of '-Q' must be set to higher than
    that number, or an infinite loop may occur.  For example, if there are
    500 permanent "legacy" verify errors on the commit server from 10
    years ago that have long since been abandoned, those files can never
```

```
   be transferred to any replica. Running p4verify.sh on the replica will
   cause those files to be scheduled, but as they cannot be pulled, they
   will land in the total pull queue.  In this scenario, the value set
   with '-Q' must be greater than 500, or an infinite loop is possible.

   Specify '-Q 0' to disable checking the total pull queue.

   The current total pull queue on a replica is reported by:

   p4 -ztag -F %replicaTransfersTotal% pull -ls

   This option can be useful if using this p4verify.sh script to pull
   many or even all archives on a new replica server machine from its
   target server.  The injected delays can give the server time to transfer
   archives scheduled in one call to 'p4 verify' before calling it again
   with the goal of avoidng overloading the pull queue.

   If '-w' and '-Q' options are both used, the delay specified by '-w'
   is ignored unless the total pull queue size is greater than or equal
   to the specified maximum total pull queue size.  The '-w' then
   essentially determines how frequently the 'p4 pull -ls' is run to
   check the total pull queue size.  A reasonable set of values might
   be '-q 50000 -w 10'.

   The '-Q' option in mutually exclusive with '-nt'.

   The '-Q' option in mutually exclusive with '-q'.

-recent
   Specify that only recent changelists should be verified.
   The $SDP_RECENT_CHANGES_TO_VERIFY variable defines how many
   changelists are considered recent; the default is 200.

   If the default is not appropriate for your site, add
   "export SDP_RECENT_CHANGES_TO_VERIFY" to /p4/common/config/p4_N.vars to
   change the default for an instance, or to /p4/common/bin/p4_vars to
   change it globally.  If $SDP_RECENT_CHANGES_TO_VERIFY is unset, the
   default is 200.

   When -recent is used, neither shelves nor files in the unload depot
   are verified.

-dlf <depot_list_file>
   Specify a file containing a list of depots to process in the desired
   order.  By default, all depots reported by 'p4 depots', which
   effectively results in depots processed in alphabetical order.

   This can be useful in time-sensitive situations where the order
   of processing can be prioritized, and/or to prevent processing
       certain depots.
```



```
    The format fo the depot list file is straighforward, one line per
    depot, without the '//' nor trailling /..., so a list might look
    like this sample:

    ProjA
    ProjB
    spec
    .swarm
    unload
    archive
    ProjC

    Blank lines and lines starting with a '#' are treated as comments and
    ignored.

    WARNING: This is not intended to be the primary method of verification,
    because it would be easy to forget to add new depots to the list file.

    If the depot list file is not readable, processing aborts.

  -ignores <regex_ignores_file>
    Specify the 'verify ignores' file, a file containing a series of
    regular expression patterns representing files or file revisions
    to ignore when scanning for verify errors.  Errors matching the
    pattern will be suppressed from the output captured in the log,
    and will not be considered a verification error.

    If the '-ignores' is not specified, the default verify ignores
    file is:

    /p4/common/config/p4verify.N.ignores

    where 'N' is the SDP instance name. If this file exists, it is
    considered the 'verify ignores' file.

    Specify '-ignores none' to avoid processing the standard ignores
    file.

    The patterns can be specific files, specific file paths, or broader
    patterns (e.g. in the case of entirely abandoned depots).  The file
    provided is passed as the '-f <file>' option to the 'grep' utility,
    and is expected to contain a series of one-line entries, each
    containing an expression to exclude from being considered as verify
    errors reported by this script.

    You can test your expression by first using it with grep to
    ensure it suppresses errors by using a command like this sample,
    providing an older log from this script that contains errors to
    be suppressed:

    grep -Ev -f /path/to/regex_file /path/to/old/p4verify.log
```

```
    If your server is case-sensitive, change that command to use '-i':

    grep -Evi -f /path/to/regex_file /path/to/old/p4verify.log

    This sample entry ignores a single file revision:

    //Alpha/main/docs/Expenses from February 1999.xls#3

    This sample entry ignores all revisions of a single file:

    //Alpha/main/docs/Expenses from February 1999.xls

    This sample entry ignores all entries in the spec depot
    related to client specs:

    //spec/client

    This sample uses the MD5 checksum from the verify error, just
    to illustrate that this can be used as an alternative to
    specifying file paths:

    D34989BFB8D9B0FB9866C4A604A05410

    This sample ignores BAD! (but not MISSING!) errors under the
    //Beta/main/src directory tree:

    //Beta/main/src/.* BAD!

    WARNING: Ensure that the regex file provided does NOT contain
    any blank lines or comments. The file should contain only tested
    regex patterns.

    This option is intended to provide a way to ignore unrecoverably lost
    file revisions from things like past infrastructure failures, for
    which search and recovery efforts have been abandoned. This option
    subtly changes the question answered by this script from "Are there any
    verify errors?" to "Are there any new verify errors, errors we don't
    already know about?"

    WARNING: This option is not intended to be incorporated into the primary
    method of verification, because ignoring archive errors in this script
    does not solve the problem at its source.  Ideally, the root cause of
    the verify errors should be addressed by recovering lost archives,
    injecting replacement content, or other means. So long as verify errors
    remain, even if ignored by this option, users attempting to access the
    revisions will still see Librarian errors, and replicas will encounter
    errors trying to pull the missing archives. This option could increase
    the risk that such revisions are never dealt with.

-Ocache
```

```
    Specify '-Ocache' to attempt a verification on a replica configured
    with a 'lbr.replication' replication configuration setting value
    of 'cache'.  By default, if the 'lbr.replication' configurable is
    set to 'cache', this script aborts, as replication of such a depot
    will schedule transfers that are likely unintended.  This is a
    safety feature.

    The 'cache' mode is generally used on replicas or edge servers with
    limited disk space.  Because running a verify will cause transfers
    of any missing files, this could result in filling up the disk.

    Use of '-Ocache' is strongly discouraged unless combined with
    other options to ensure that only targeted paths are scheduled
    for transfer.

 -v     Verbose.  Show output of verify attempts, which is suppressed by default.
    Setting SDP_SHOW_LOG=1 in the shell environment has the same effect as -v.

    The default behavior of this script is to generate no terminal output,
    but instead to write output into a log file -- see LOGGING below.  If
    '-v' is specified, the generated log is sent to stdout at the end of
    processing.  This flag is not recommended for routine cron operation or
    for large data sets.

    The -chunks and -recent options are mutually exclusive.

 -L <log>
    Specify the log file to use.  The default is /p4/N/logs/p4verify.log

    Log rotation and old log cleanup logic does not apply to log files
    specified with -L.  Thus, using -L is not recommended for routine
    scheduled operation, e.g. via crontab.

DEBUGGING OPTIONS:
 -n No-Operation (NO_OP) mode, for debugging.

    Display certain commands that would be executed without executing
    them. When '-n' is used, commands that might take a long time to
    run or affect data are only displayed.

    Even in '-n' mode, some information-gathering commands such as
    listing shelved CLs are executed, which may cause the script to take
    a bit of time to run on a large data set even in dry run mode.

 -d Specify that debug messages should be displayed.

 -D Use bash 'set -x' extreme debugging verbosity, and imply '-d'.

 -L off
    The special value '-L off' disables logging. This can only be used
    with '-n' for debugging.
```

```
HELP OPTIONS:
 -h Display short help message
 -man   Display man-style help message

EXAMPLES:
    Example 1: Full Verify

    This script is typically called via cron with only the instance
    parameter as an argument, e.g.:

    p4verify.sh 1

    Example 2: Fast Verify

    A "fast" verify is one in which only the check for MISSING archives
    is done, while the resource-intensive checksum calculation of
    potentially BAD existing archives is skipped.  This is especially
    useful when used on a replica.

    p4verify.sh 1 -o MISSING

    Example 3: Fast and Recent Verify

    The '-o MISSING' and '-recent' flags can be combined for a very
    fast check.  This check might be incorporated into a failover
    procedure.

    p4verify.sh 1 -o MISSING -recent

    Example 4: Submitted Files Only

    This will verify only use submitted files, ignoring shelves and the
    spec and unload depots, putting the results in a specified log:

    p4verify.sh 1 -ns -nS -nu -L  -L /p4/1/logs/p4verify.submitted.log

    Example 5: Shelved Files Only

    This will verify only use submitted files, ignoring shelves and the
    spec and unload depots, putting them in a specified log:

    p4verify.sh 1 -nr -ns -nu -L /p4/1/logs/p4verify.shelved.log

    Example 6: A Dry Run

    The '-n' option can be used for a dry run.  Output may also be
    displayed to the screen ('-v') for a dry run and the log file optionally
    discarded:

    p4verify.sh 1 -n -nS -L off -v
```

```
    Example 7: Archive File Load for New Replica

    The p4verify.sh script can be used to schedule transfers of a large
    number of files from a replica.  When doing so, however, overloading
    the new replicas pull queue with too many files may impact metadata
    replication.  This can be addressed by combining a variety of
    options, such as '-chunks' and '-Q'.  For example:

    p4verify.sh 1 -chunks 200M -Q 10000 -w 20 -o MISSING

NOHUP USAGE:
    Because archive verification is typically a long running task,
    it is advisable to use 'nohup' to call each command, and combine
    that by running the command as a background process.  Alternately,
    use 'screen' or similar.

    Any of the examples above can be used with 'nohup', without output
    redirected to /dev/null (i.e. to "the void", as this script handles
    logging and output redirection).

    To use 'nohup', start the command line with 'nohup', and then after
    the command, add this text exactly:

    < /dev/null > /dev/null 2>&1 &

    As a example, Example 2 above, called with nohup, would look like:

    nohup /p4/common/bin/p4verify.sh 1 -o MISSING < /dev/null > /dev/null 2>&1 &

    With the ampersand '&' at the end, the command will appear to return
    immediately as the process continues to run in the background.

    Then optionally monitor the log:

    tail -f /p4/1/logs/p4verify.log

LOGGING:
    This script generates no output by default.  All (stdout and stderr) is
    logged to /p4/N/logs/p4verify.log.

    The exception is usage errors, which result an error being sent to
    stderr followed usage info on stdout, followed by an immediate exit.

NOTIFICATIONS:
    In addition to logging, a short summary of the verify is sent as a
    notification.  The summary is reliably short even if the output of the
    verifications done by this script results in a large log file.

    There are two notification schemes with this script:
```

```
    * Email notification is always attempted.

    * AWS SNS notification is attempted if the SNS_ALERT_TOPIC_ARN custom
    setting is defined. This is typically set in:

    /p4/common/site/config/p4_N.vars.local

TIMING:
    The log file captures various timing information, including the
    time required to verify each depot, or each chunk or path if
    '-paths' or '-chunks' are used. The time to verify shelves
    in all depots is reported separately from submitted files.

    Timing indications all start with the text 'Time: ' on the beginning
    of a line of output in the log file, and can be extrated with a
    command like this example (adjusting the log file name as needed):

    grep ^Time: /p4/1/logs/p4verify.log

EXIT CODES:
    An exit code of 0 indicates no errors were encountered attempting to
    perform verifications, AND that all verifications attempted
    reported no problems.

    A exit status of 1 indicates that verifications could not be
    attempted for some reason.

    A exit status of 2 indicates that verifications were successfully
    performed, but that problems such as BAD or MISSING files
    were detected, or else system limits prevented verification.
```

8.4.8. p4login

The `/p4/common/bin/p4login` script is a convenience wrapper to execute a series of `p4 login` commands, using the administration password configured in `mkdirs.cfg` and subsequently stored in a text file: `/p4/common/config/.p4passwd .p4_<instance>.admin`.

Usage

```
USAGE for p4login v4.4.4:

p4login [<instance>] [-p <port> | -service] [-automation] [-all]

   or

p4login -h|-man

DESCRIPTION:

    In its simplest form, this script simply logs in P4USER to P4PORT
```

```
    using the defined password access mechanism.

    It generates a login ticket for the SDP super user, defined by
    P4USER when sourcing the SDP standard shell environment.  It is
    called from cron scripts, and so does not normally generate any
    output.

    If run on a replica with the -service option, the serviceUser defined
    for the given replica is logged in.

    The $SDP_AUTOMATION_USERS variable can be defined in
    /p4_N.vars. If defined, this should contain a
    comma-delimited list of automation users to be logged in when the
    -automation option is used.  A definition might look like:

    export SDP_AUTOMATION_USERS=builder,trigger-admin,p4review

    Login behavior is affected by external factors:
    1. P4AUTH, if defined, affects login behavior on replicas.

    2. The auth.id setting, if defined, affects login behaviors (and
    generally simplifies them).

    3. The $SDP_ALWAYS_LOGIN variable.  If set to 1, this causes p4login
    to always execute a 'p4 login' command to generate a login ticket,
    even if a 'p4 login -s' test indicates none is needed.  By default,
    the login is skipped if a 'p4 login  -s' test indicates a long-term
    ticket is available that expires 31+days in the future.
    Add "export SDP_ALWYAYS_LOGIN=1" to /p4_N.vars to
    change the default for an instance, or to /p4/common/bin/p4_vars to
    change it globally.  If unset, the default is 0.

    4. If the P4PORT contains an ssl: prefix, the P4TRUST relationship
    is checked, and if necessary, a p4 trust -f -y is done to establish
    trust.

OPTIONS:
<instance>
    Specify the SDP instances.  If not specified, the SDP_INSTANCE
    environment variable is used instead.  If the instance is not
    defined by a parameter and SDP_INSTANCE is not defined, p4login
    exists immediately with an error message.

 -service
    Specify -service when run on a replica or edge server to login
    the super user and the replication service user.

    This option is not compatible with '-p <port>'.

 -p <port>
    Specify a P4PORT value to login to, overriding the default
```

```
    defined by P4PORT setting in the environment.  If operating
    on a host other than the master, and auth.id is set, this
    flag is ignored; the P4TARGET for the replica is used
    instead.

    This option is not compatible with '-service'.

 -automation
    Specify -automation to login external automation users defined
    by the $SDP_AUTOMATION_USERS variable.

 -v     Show output of login attempts, which is suppressed by default.
    Setting SDP_SHOW_LOG=1 in the shell environment has the same
    effect as -v.

 -L <log>
    Specify the log file to use.  The default is /p4/N/logs/p4login.log

 -d     Set debugging verbosity.

 -D     Set extreme debugging verbosity.

HELP OPTIONS:
 -h Display short help message
 -man   Display man-style help message

EXAMPLES:
    1. Typical usage for automation, with instance SDP_INSTANCE defined
    in the environment by sourcing p4_vars, and logging in only the super
    user P4USER to P4PORT:
    source /p4/common/bin/p4_vars abc
    p4login

    Login in only P4USER to the specified port, P4MASTERPORT in this example:
    p4login -p $P4MASTERPORT

    Login the super user P4USER, and then login the replication serviceUser
    for the current ServerID:
    p4login -service

    Login external automation users (see SDP_AUTOMATION_USERS above):
    p4login -automation

    Login all users:
    p4login -all

    Or: p4login -service -automation

LOGGING:
    This script generates no output by default.  All (stdout and stderr) is
    logged to /p4/N/logs/p4login.log.
```

```
    The exception is usage errors, which result an error being sent to
    stderr followed usage info on stdout, followed by an immediate exit.

    If the '-v' flag is used, the contents of the log are displayed to
    stdout at the end of processing.

EXIT CODES:
    An exit code of 0 indicates a valid login ticket exists, while a
    non-zero exit code indicates a failure to login.
```

8.4.9. p4d_<instance>_init

Starts the Perforce server instance. Can be called directly or as describe in Section 4.2.3, "Configuring Automatic Service Start on Boot" - it is created by `mkdirs.sh` when SDP is installed.

Do not use directly if you have configured systemctl for systemd Linux distributions such as CentOS 7.x. This risks database corruption if `systemd` does not think the service is running when it actually is running (for example on shutdown systemd will just kill processes without waiting for them).

This script sources `/p4/common/bin/p4_vars`, then runs `/p4/common/bin/p4d_base` (Section 8.6.10, "p4d_base").

Usage

```
/p4/<instance>/bin/p4d_<instance>_init [ start | stop | status | restart ]
/p4/1/bin/p4d_1_init start
```

8.4.10. refresh_P4ROOT_from_offline_db.sh

The `/p4/common/bin/refresh_P4ROOT_from_offline_db.sh` script is intended to be used occasionally, perhaps monthly, quarterly, or on-demand, to help ensure that your live (`root`) database files are defragmented.

It will:

- stop p4d
- truncate/rotate live journal
- replay journals to offline_db
- switch the links between `root` and `offline_db`
- restart p4d

It also knows how to do similar processes on edge servers and standby servers or other replicas.

Usage

```
/p4/common/bin/refresh_P4ROOT_from_offline_db.sh <instance>
/p4/common/bin/refresh_P4ROOT_from_offline_db.sh 1
```

8.4.11. run_if_master.sh

The `/p4/common/bin/run_if_master.sh` script is explained in Section 8.4.14, “run_if_master/edge/replica.sh”

8.4.12. run_if_edge.sh

The `/p4/common/bin/run_if_edge.sh` script is explained in Section 8.4.14, “run_if_master/edge/replica.sh”

8.4.13. run_if_replica.sh

The `/p4/common/bin/run_if_replica.sh` script is explained in Section 8.4.14, “run_if_master/edge/replica.sh”

8.4.14. run_if_master/edge/replica.sh

The SDP uses wrapper scripts in the crontab: `run_if_master.sh`, `run_if_edge.sh`, `run_if_replica.sh`. We suggest you ensure these are working as desired, e.g.

Usage

```
/p4/common/bin/run_if_master.sh 1 echo yes
/p4/common/bin/run_if_replica.sh 1 echo yes
/p4/common/bin/run_if_edge.sh 1 echo yes
```

It is important to ensure these are returning the valid results for the server machine you are on.

Any issues with these scripts are likely configuration issues with `/p4/common/config/p4_1.vars` (for instance `1`)

8.5. More Server Scripts

These scripts are helpful components of the SDP that run on the server machine, but are not included in the default crontab schedules.

8.5.1. p4.crontab

Contains crontab entries to run the server maintenance scripts.

Location: /p4/sdp/Server/Unix/p4/common/etc/cron.d

8.5.2. verify_sdp.sh

The `/p4/common/bin/verify_sdp.sh` does basic verification of SDP setup.

Usage

```
USAGE for verify_sdp.sh v5.22.2:

verify_sdp.sh [<instance>] [-online] [-skip <test>[,<test2>,...]] [-warn
<test>[,<test2>,...]] [-si] [-L <log>|off ] [-D]

   or

verify_sdp.sh -h|-man

DESCRIPTION:

    This script verifies the current SDP setup for the specified instance,
    and also performs basic health checks of configured servers.

    This uses the SDP instance bin directory /p4/N/bin to determine
    what server binaries (p4d, p4broker, p4p) are expected to be configured
    on this machine.

    Existence of the '*_init' script indicates the given binary is
    expected. For example, for instance 1, if /p4/1/bin/p4d_1_init
    exists, a p4d server is expected to run on this machine.

    Checks may be executed or skipped depending on what servers are
    configured. For example, if a p4d is configured, the $P4ROOT/server.id
    file should exist. If p4p is configured, the 'cache' directory
    should exist.

OPTIONS:
 <instance>
    Specify the SDP instances.  If not specified, the SDP_INSTANCE
    environment variable is used instead.  If the instance is not
    defined by a parameter and SDP_INSTANCE is not defined,
    exits immediately with an error message.

 -online
    Online mode.  Does additional checks that requires p4d, p4broker,
    and/or p4p to be online. Any servers for which there are
    *_init scripts in the Instance Bin directory must be online.
    The Instance Bin directory is the /p4/N/bin directory, where N
    is the SDP instance name.

 -skip <test>[,<test2>,...]

    Specify a comma-delimited list of named tests to skip.
```

```
    Valid test names are:

    * cron|crontab: Skip crontab check. Use this if you do not expect crontab to
    be configured, perhaps if you use a different scheduler.
    * excess: Skip checks for excess copies of p4d/p4p/p4broker in PATH.
    * init: Skip compare of init scripts w/templates in /p4/common/etc/init.d
    * license: Skip license related checks.
    * masterid: Skip check ensuring ServerID of master starts with 'master'.
    * offline_db: Skip checks that require a healthy offline_db.
    * p4root: Skip checks that require healthy P4ROOT db files.
    * p4t_files: Skip checks for existence of P4TICKETS and P4TRUST files.
    * passwd|password: Skip SDP password checks.
    * version: Skip version checks.

    As an alternative to using the '-skip' option, the shell environment
    variable VERIFY_SDP_SKIP_TEST_LIST can be set to a comma-separated
    list of named tests to skip.  Using the command line parameter is the
    best choice for temporarily skipping tests, while specifying the
    environment variable is better for making permanent exceptions (e.g.
    always excluding the crontab check if crontabs are not used at this
    site).  The variable should be set in /p4/common/config/p4_N.vars.

    If the '-skip' option is provided, the VERIFY_SDP_SKIP_TEST_LIST
    variable is ignored (not appended to). So it may make sense to
    reference the variable on the command line. For example, if the
    value of the variable is 'crontab', to skip crontab and license
    checks, you could specify:

    -skip $VERIFY_SDP_SKIP_TEST_LIST,license

-warn <test>[,<test2>,...]
    Specify a comma-delimited list of named tests that will be reported
    as warnings rather than errors.

    The list of valid test names as the same as for the '-skip' option.

    As an alternative to using the '-warn' option, the shell environment
    variable VERIFY_SDP_WARN_TEST_LIST can be set to a comma-separated
    list of name tests to skip.  Using the command line parameter is the
    best choice for temporarily converting errors to warnings, while
    specifying the environment variable is better for making the
    conversion to warnings permanent.  The variable should be set in
    /p4/common/config/p4_N.vars file.

    If the '-warn' option is provided, the VERIFY_SDP_WARN_TEST_LIST
    variable is ignored (not appended to). So it may make sense to
    reference the variable on the command line. For example, if the
    value of the variable is 'crontab', to convert to warnings for
    crontab and excess binaries tests, you could specify:

    -warn $VERIFY_SDP_WARN_TEST_LIST,excess
```



```
 -si    Silent mode, useful for cron operation.  Both stdout and stderr
    are still captured in the log.  The '-si' option cannot be used
    with '-L off'.

 -L <log>
    Specify the log file to use.  The default is /p4/N/logs/verify_sdp.log
    The special value 'off' disables logging to a file.

    Note that '-L off' and '-si' are mutually exclusive.

 -D Set extreme debugging verbosity.

HELP OPTIONS:
 -h Display short help message
 -man   Display man-style help message

EXAMPLES:
    Example 1: Typical usage:

    This script is typically called after SDP update with only the instance
    name or number as an argument, e.g.:

    verify_sdp.sh 1

    Example 2: Skipping some checks.

    verify_sdp.sh 1 -skip crontab

    Example 3: Automation Usage

    If used from automation already doing its own logging, use -L off:

    verify_sdp.sh 1 -L off

LOGGING:
    This script generates a log file and also displays it to stdout at the
    end of processing.  By default, the log is:
    /p4/N/logs/verify_sdp.log.

    The exception is usage errors, which result an error being sent to
    stderr followed usage info on stdout, followed by an immediate exit.

    If the '-si' (silent) flag is used, the log is generated, but its
    contents are not displayed to stdout at the end of processing.

EXIT CODES:
    An exit code of 0 indicates no errors were encountered attempting to
    perform verifications, and that all checks verified cleanly.
```


8.6. Other Scripts and Files

The following table describes other files in the SDP distribution. These files are usually not invoked directly by you; rather, they are invoked by higher-level scripts.

8.6.1. backup_functions.sh

The `/p4/common/bin/backup_functions.sh` script contains Bash functions used in other SDP scripts.

It is **sourced** (`source /p4/common/bin/backup_functions.sh`) by other scripts that use the common shared functions.

It is not intended to be called directly by the user.

8.6.2. broker_rotate.sh

The `/p4/common/bin/broker_rotate.sh` rotates the broker log file. It is intended for use on a server machine that has only broker running. When a broker is run on a p4d server machine, the `daily_checkpoint.sh` take care of rotating the broker log.

It can be added to a crontab for e.g. daily log rotation.

Usage

```
/p4/common/bin/broker_rotate.sh <instance>
/p4/common/bin/broker_rotate.sh 1
```

8.6.3. edge_dump.sh

The `/p4/common/bin/edge_dump.sh` script is designed to create a seed checkpoint for an Edge server.

An edge server is naturally filtered, with certain database tables (e.g. db.have) excluded. In addition to implicit filtering, the server spec may specify additional tables to be excluded, e.g. by using the ArchiveDataFilter field of the server spec.

The script requires the SDP instance and the edge ServerID.

Usage

```
/p4/common/bin/edge_dump.sh <instance> <edge server id>
/p4/common/bin/edge_dump.sh 1 p4d_edge_syd
```

It will output the full path of the checkpoint to be copied to the edge server and used with Section 8.6.24, “recover_edge.sh”

8.6.4. edge_vars

The `/p4/common/bin/edge_vars` file is sourced by scripts that work on edge servers.

It sets the correct list db.* files that are edge-specific in the federated architecture. This version is dependent on the version of p4d in use; this script accounts for the P4D version.

It is not intended for users to call directly.

8.6.5. edge_shelf_replicate.sh

The `/p4/common/bin/edge_shelf_replicate.sh` script is intended to be run on an edge server and will ensure that all shelves are replicated to that edge server (by running `p4 print` on them).

Only use if directed to by Perforce Support or Perforce Consulting.

8.6.6. load_checkpoint.sh

The `/p4/common/bin/load_checkpoint.sh` script loads a checkpoint into `root` and `offline_db` for commit/edge/replica instance.

This script will replace your `/p4/<instance>/root` database files! **Be careful!**

If you want to create db files in `offline_db` then use Section 8.4.5, “recreate_offline_db.sh”.

Usage

```
USAGE for load_checkpoint.sh v2.8.1:

load_checkpoint.sh <checkpoint> [<jnl.1> [<jnl.2> ...]] [-k] [-i <instance>] [-s
<ServerID>] [-t <Type>] [-verify {default|"Verify Options"} [-delay <delay>]] [-c] [-
l] [-r] [-b] [-y] [-L <log>] [-si] [-v<n>] [-D]

or

load_checkpoint.sh [-h|-man|-V]


DESCRIPTION:
    This script loads a specified checkpoint into /p4/N/root and /p4/N/offline_db,
    where 'N' is the SDP instance name.

    At the start of processing, preflight checks are done. Preflight checks
    include:
    * The specified checkpoint and corresponding *.md5 file must exist.
    * All journal files to replay (if any were specified) must exist.
    * The $P4ROOT/server.id file must exist, unless '-s' is specified.
    * If the $P4ROOT/server.id file exists and '-s' is specified, the values
    must match.
    * The $P4ROOT/license file must exist, unless '-l' is specified or if
    the replica type does not require a license.
    * Basic SDP structure and key files must exist.

    If the preflight passes, the p4d_N service is shutdown, and also the
    p4broker_N service is shutdown if configured.
```

```
    If a P4LOG file exists, it is moved aside so there is a fresh p4d server
    log corresponding to operation after the checkpoint load.

    If a P4JOURNAL file exists, it is moved aside as the old journal data is
    no longer relevant after a checkpoint replay. (Exception: If the P4JOURNAL
    is speciffed in a list of journals to reply, then it is not moved aside).

    Next, any existing state* files in P4ROOT are removed.

    Next, any existing database files in P4ROOT are removed (or moved aside
    with '-k').

    Next, the specified checkpoint is loaded. Upon completion, the Helix Core
    server process, p4d_N, is started.

    If the server to be started is a replica, the serviceUser configured for
    the replica is logged into the P4TARGET server.  Any needed 'p4 trust' and
    'p4 login' commands are done to enable replication.

    Note that this part of the processing will fail if the correct super user
    password is not stored in the standard SDP password file,

    /p4/common/config/.p4passwd.p4_N.admin

    After starting the server, a local 'p4 trust' is done if needed, and then
    a 'p4login -service -v' and 'p4login -v'.

    By default, the p4d_N service is started, but the p4broker_N service is not.
    Specify '-b' to restart both services.

ARGUMENTS AND OPTIONS:
 <checkpoint>
    Specify the path to the checkpoint file to load. Exactly one checkpoint
    must be specified.

    The file may be a compressed or uncompressed checkpoint, and it may be a case
    sensitive or case-insensitive checkpoint.  The checkpoint file must have a
    corresponding *.md5 checksum file in the same directory, with one of two name
    variations: If the checkpoint file is /somewhere/foo.gz, the checksum file may
    be named /somewhere/foo.gz.md5 or /somewhere/foo.md5.

<jnl.1> [<jnl.2> ...]
    Specify the path to the one or more journal files to replay after the
    checkpoint, in the correct sequence order.

 -k Specify '-k' to keep db.* files in P4ROOT rather than removing them.  This may
    be desirable to preserve databases for investigation.  This option requires
    sufficient disk space to hold an extra copy of the db.* files.

    If -k specified, a folder named 'MovedDBs.<datestamp>' is created under the
```

```
    P4ROOT directory, and databases are moved there.

-i <instance>
    Specify the SDP instance. This can be omitted if SDP_INSTANCE is already
    defined.

-s <ServerID>
    Specify the ServerID.  This value is written into $P4ROOT/server.id file.

    If no $P4ROOT/server.id file exists, this flag is required.

    If the $P4ROOT/server.id file exists, this argument is not needed. If this
    '-s <ServerID>' is given and a $P4ROOT/server.id file exists, the value in
    the file must match the value specified with this argument.

-t <Type>
    Specify the replica type tag if the checkpoint to be loaded is for an edge
    server or replica. The set of valid values for the replica type
    tag are defined in the documentation for mkrep.sh.  See: mkrep.sh -man

    If the type is specified, the '-s <ServerID>' is required.

    If the SDP Server Spec Naming Standard is followed, the ServerID
    specified with '-s' will start with 'p4d_'. In that case, the
    value for '-t edge' value is inferred, and '-t' is not required.

    If the type is specified or inferred, certain behaviors change based
    on the type:
    * If the type is edge, only the correct edge-specific subset of database
    tables are loaded.
    * The P4ROOT/license file check is suppressed unless the type is
    ha, ham, fs, for fsm (standby replicas usable with 'p4 failover').

    Do not use this '-t <Type>' option if the checkpoint being loaded is
    for a master server.

    For an edge server, an edge seed checkpoint created with edge_dump.sh
    must be used if the edge is filtered, e.g. if any of the *DataFilter
    fields in the server spec are used. If the edge server is not filtered
    by means other than being an edge server (for which certain tables are
    filtered by nature), a standard full checkpoint from the master can be
    used.

    For a filtered forwarding replica, a proper seed checkpoint must be
    loaded.  This can be created on the master using key options to p4d,
    including '-P <ServerID> -jd <SeedCkp' on the master (possibly using
    the 'offline_db' to avoid downtime, similar to how edge_dump.sh
    works for edge servers).

    WARNING: While this script is useful for seeding a new edge server, this
    script is NOT to be used for recovering or reseeding an existing edge server,
```

```
   because all edge-local database tables (mostly workspace data) would be lost.
   To recover an existing edge server, see the recover_edge.sh script.

   Warning: If this option is specified with the incorrect type for the
   checkpoint specified, results will be unpredictable.

-verify default [-delay <delay>]
-verify "Verify Options" [-delay <delay>]

   Specify '-verify' to initiate a call to 'p4verify.sh' after the server
   is online. On a replica, this can be useful to cause the server to pull
   missing archive files from its P4TARGET server. If this load_checkpoint.sh
   script is used in a recovery situation for a master server, this '-verify'
   option can be used to discover if archive files are missing after the
   metadata is recovered.

   The 'p4verify.sh' script has a rich set of options; see 'p4verify.sh -man'
   for more info.  To options to pass to p4verify.sh can be passed in a
   quoted list, or the value 'verify default' can be used to indicate these
   options:

   -o MISSING

   By default, a fast verify is used if the p4d version is new enough (2021.1+).
   See 'p4verify.sh -man' for more information, specifically the description
   of the '-o MISSING' option.

   In all cases, p4verify.sh is invoked as a background process; this
   load_checkpoint.sh script does not wait for it to complete. The p4verify.sh
   script will email as per normal when it completes.

   The optional delay option specifies how long to wait until kicking off the
   p4verify.sh command, in seconds. The default is 600 seconds.
   This is intended to give the replica time get get caught up with metadata
   before the archive pulls are scheduled. The delay is a workaround for job079842.

-c Specify that SSL certificates are required, and not to be generated with
   'p4d_N -Gc'.

   By default, if '-c' is not supplied and SSL certs are not available, certs
   are generated automatically with 'p4d_N -Gc'.

-l Specify that the server is to start without a license file. By default, if
   there is no $P4ROOT/license file, this script will abort.  Note that if '-l'
   is specified and a license file is actually needed, the attempt this script makes
   to start the server after loading the checkpoint will fail.

   If -'t <type>' is specified, the license check is skipped unless the type is
   'ha' or 'ham', i.e. HA replicas that need a license file to support failover.

-r Specify '-r' to replay only to P4ROOT.  By default, this script replays both
```

```
    to P4ROOT and the offline_db.

 -b Specify '-b' to start the a p4broker process (if configured).  By default
    the p4d process is started after loading the checkpoint, but the p4broker
    process is not. This can be useful to ensure the human administrator has
    an opportunity to do sanity checks before enabling the broker to allow
    access by end users (if the broker is deployed for this usage).

 -y Use the '-y' flag to bypass an interactive warning and confirmation
    prompt.

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

 -L <log>
    Specify the path to a log file.  By default, all output (stdout and stderr)
    goes to:
    /p4/<instance>/logs/load_checkpoint.<timestamp>.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.

 -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:
    EXAMPLE 1: Non-interactive Usage

    Non-interactive usage (bash syntax) to load a checkpoint:

    nohup /load_checkpoint.sh /p4/1/checkpoints/p4_1.ckp.4025.gz -i 1 -y < /dev/null >
/dev/null 2>&1 &

    Then, monitor with:
    tail -f $(ls -t $LOGS/load_checkpoint.*.log|head -1)


    EXAMPLE 2: Checkpoint Load then Verify

    Non-interactive usage (bash syntax) to load a checkpoint followed by a full
    verify of recent archives files only with other options passed to verify.sh:

    nohup /load_checkpoint.sh /p4/1/checkpoints/p4_1.ckp.4025.gz -i 1 -verify -recent
-nu -ns -y < /dev/null > /dev/null 2>&1 &
```

```
    EXAMPLE 3: Load Checkpoint and Journals

    Non-interactive usage (bash syntax) to loading a checkpoint and subsequent
journals:

    nohup /load_checkpoint.sh /p4/1/checkpoints/p4_1.ckp.4025.gz
/p4/1/checkpoints/p4_1.jnl.4025 /p4/1/checkpoints/p4_1.jnl.4026 -i 1 -y < /dev/null >
/dev/null 2>&1 &

    Then, monitor with:
    tail -f $(ls -t $LOGS/load_checkpoint.*.log|head -1)


    EXAMPLE 4: Interactive usage.

    Interactive usage to load a checkpoint with no license file.

    /load_checkpoint.sh /p4/1/checkpoints/p4_1.ckp.4025.gz -i 1 -l

    With interactive usage, logging still occurs; all output to the screen is
    captured.

    Note that non-interactive usage with nohup is recommended for checkpoints
    with a long replay duration, to make operation more reliable in event of a
    shell session disconnect.  Alternately, running interactively in a 'screen'
    session (if 'screen' is available) provides similar protection against
    shell session disconnects.

    EXAMPLE 5: Seed New Edge

    Seeding a new edge server.

    nohup /load_checkpoint.sh /p4/1/checkpoints/p4_1.ckp.4025.gz -i 1 -s p4d_edge_syd
< /dev/null > /tmp/null 2>&1 &

    WARNING: While this script is useful for seeding a new edge server, this
    script is NOT to be used for recovering or reseeding an existing edge server,
    because all edge-local database tables (mostly workspace data) would be lost.
    To recover an existing edge server, see the recover_edge.sh script.

    EXAMPLE 6: Seed New Edge and Verify

    Seeding a new edge server and then do a verify with default options.

    nohup /load_checkpoint.sh /p4/1/checkpoints/p4_1.ckp.4025.gz -i 1 -s p4d_edge_syd
-verify default < /dev/null > /tmp/null 2>&1 &
```

8.6.7. gen_default_broker_cfg.sh

The `/p4/common/bin/gen_default_broker_cfg.sh` script generates an SDP instance-specific variant of the generic P4Broker config file. Display to standard output.

Usage:

```
cd /p4/common/bin
gen_default_broker_cfg.sh 1 > /tmp/p4broker.cfg.ToBeReviewed
```

The final p4broker.cfg should end up here:

```
/p4/common/config/p4_${SDP_INSTANCE}.${SERVERID}.broker.cfg
```

8.6.8. journal_watch.sh

The `/p4/common/bin/journal_watch.sh` script will check diskspace available to P4JOURNAL and trigger a journal rotation based on specified thresholds. This is useful in case you are in danger of running out of disk space and your rotated journal files are stored on a separate partition than the active journal.

This script is using the following external variables:

- SDP_INSTANCE - The instance of Perforce that is being backed up. If not set in environment, pass in as argument to script.
- P4JOURNALWARN - Amount of space left (K,M,G,%) before min journal space where an email alert is sent
- P4JOURNALWARNALERT - Send an alert if warn threshold is reached (true/false, default: false)
- P4JOURNALROTATE - Amount of space left (K,M,G,%) before min journal space to trigger a journal rotation
- P4OVERRIDEKEEPJNL - Allow script to temporarily override KEEPJNL to retain enough journals to replay against oldest checkpoint (true/false, default: false)

Usage

```
/p4/common/bin/journal_watch.sh <P4JOURNALWARN> <P4JOURNALWARNALERT> <P4JOURNALROTATE>
<P4OVERRIDEKEEPJNL (Optional)>
```

Examples

Run from CLI that will warn via email if less than 20% is available and rotate journal when less than 10% is available

```
./journal_watch.sh 20% TRUE 10% TRUE
```

Cron job that will warn via email if less than 20% is available and rotate journal when less than 10% is available

```
30 * * * * [ -e /p4/common/bin ] && /p4/common/bin/run_if_master.sh ${INSTANCE}
/p4/common/bin/journal_watch.sh ${INSTANCE} 20\% TRUE 10\% TRUE
```

8.6.9. kill_idle.sh

The `/p4/common/bin/kill_idle.sh` script runs `p4 monitor terminate` on all processes showing in the output of `p4 monitor show` that are in the IDLE state.

Usage

```
/p4/common/bin/kill_idle.sh <instance>
/p4/common/bin/kill_idle.sh 1
```

8.6.10. p4d_base

The `/p4/common/bin/p4d_base` script is the script to start/stop/restart the `p4d` instance.

It is called by `p4d_<instance>_init` script (and thus also `systemctl` on systemd Linux distributions). It is not intended to be called by users directly.

8.6.11. p4broker_base

The `/p4/common/bin/p4broker_base` script is very similar to Section 8.6.10, “p4d_base” but for the `p4broker` service instance.

See p4broker in SysAdmin Guide

8.6.12. p4ftpd_base

The `/p4/common/bin/p4ftpd_base` script is very similar to Section 8.6.10, “p4d_base” but for the `p4ftp` service instance. The p4ftp has been deprecated; this may be removed in a future SDP release.

This product is very seldom used these days!

See P4FTP Installation Guide.

8.6.13. p4p_base

The `/p4/common/bin/p4p_base` is very similar to Section 8.6.10, “p4d_base” but for the `p4p` (P4 Proxy) service instance.

See p4proxy in SysAdmin Guide

8.6.14. p4pcm.pl

The `/p4/common/bin/p4pcm.pl` script is a utility to remove files in the proxy cache if the amount of free disk space falls below the low threshold.

Usage

```
Usage:

   p4pcm.pl [-d "proxy cache dir"] [-tlow <low_threshold>] [-thigh <high_threshold>]
[-n]
or
   p4pcm.pl -h

This utility removes files in the proxy cache if the amount of free disk space
falls below the low threshold (default 10GB).  It removes files (oldest first)
until the high threshold is (default 20GB) is reached. Specify the thresholds
in kilobyte units (kb).

The '-d "proxy cache dir"' argument is required unless $P4PCACHE is defined,
in which case it is used.

The log is $LOGS/p4pcm.log if $LOGS is defined, else p4pcm.log in the current
directory.

Use '-n' to show what files would be removed.
```

8.6.15. p4review.py

The `/p4/common/bin/p4review.py` script sends out email containing the change descriptions to users who are configured as reviewers for affected files (done by setting the Reviews: field in the user specification). This script is a version of the `p4review.py` script that is available on the Perforce Web site, but has been modified to use the server instance number. It relies on a configuration file in `/p4/common/config`, called `p4_<instance>.p4review.cfg`.

This is not required if you have installed Swarm which also performs notification functions and is easier for users to configure.

Usage

```
/p4/common/bin/p4review.py  # Uses config file as above
```

8.6.16. p4review2.py

The `/p4/common/bin/p4review2.py` script is an enhanced version of Section 8.6.15, “p4review.py”

1. Run p4review2.py --sample-config > p4review.conf
2. Edit the file p4review.conf

3. Add a crontab similar to this:
    - * * * * python2.7 /path/to/p4review2.py -c /path/to/p4review.conf

Features:

- Prevent multiple copies running concurrently with a simple lock file.
- Logging support built-in.
- Takes command-line options.
- Configurable subject and email templates.
- Use P4Python when available and use P4 (the CLI) as a fallback.
- Option to send a *single* email per user per invocation instead of multiple ones.
- Reads config from a INI-like file using ConfigParser
- Have command line options that overrides environment variables.
- Handles unicode-enabled server **and** non-ASCII characters on a non-unicode-enabled server.
- Option to opt-in (--opt-in-path) reviews globally (for migration from old review daemon).
- Configurable URLs for changes/jobs/users (for swarm).
- Able to limit the maximum email message size with a configurable.
- SMTP auth and TLS (not SSL) support.
- Handles P4AUTH (optional; use of P4AUTH is no longer recommended).

8.6.17. proxy_rotate.sh

The `/p4/common/bin/proxy_rotate.sh` rotates the proxy log file. It is intended for use on a server machine that has only proxy running. When a proxy is run on a p4d server machine, the `daily_checkpoint.sh` script takes care of rotating the proxy log.

It can be added to a crontab for e.g. daily log rotation.

Usage

```
/p4/common/bin/proxy_rotate.sh <instance>
/p4/common/bin/proxy_rotate.sh 1
```

8.6.18. p4sanity_check.sh

The `/p4/common/bin/p4sanity_check.sh` script is a simple script to run:

- p4 set
- p4 info
- p4 changes -m 10

Usage

```
/p4/common/bin/p4sanity_check.sh <instance>
/p4/common/bin/p4sanity_check.sh 1
```

8.6.19. p4dstate.sh

The `/p4/common/bin/p4dstate.sh` is a trouble-shooting script for use when directed by support, e.g. in situations such as server hanging, major locking problems etc.

It is an "SDP-aware" version of the standard p4dstate.sh so that it only requires the SDP instance to be specified as a parameter (since the location of logs etc are defined by SDP).

Usage

```
sudo /p4/common/bin/p4dstate.sh <instance>
sudo /p4/common/bin/p4dstate.sh 1
```

8.6.20. ps_functions.sh

The `/p4/common/bin/ps_functions.sh` library file contains common functions for using 'ps' to check on process ids. It is not intended to be called by users.

```
get_pids ($exe)
```

Usage

```
Call with an exe name, e.g. /p4/1/bin/p4web_1
```

Examples

```
p4web_pids=$(get_pids $P4WEBBIN)
p4broker_pids=$(get_pids $P4BROKERBIN)
```

8.6.21. pull.sh

The `/p4/common/bin/pull.sh` is a reference pull trigger implementation for External Archive Transfer using pull-archive and edge-content triggers

It is a fast content transfer mechanism using Aspera (and can be adapted to other similar UDP based products.) An Edge server uses this trigger to pull files from its upstream Commit server. It replaces or augments the built in replication archive pull and is useful in scenarios where there are lots of large (binary) files and commit/edge are geographically distributed with high latency and/or low bandwidth between them.

See also companion trigger Section 8.6.29, “submit.sh”.

It is based around getting a list of files to copy from commit to edge, then doing the file transfer using `ascp` (Aspera file copy).

The configurable `pull.trigger.dir` should be set to a temp folder like `/p4/1/tmp`.

Startup commands look like:

```
startup.2=pull -i 1 -u --trigger --batch=1000
```

The trigger entry for the pull commands looks like this:

```
pull_archive pull-archive pull "/p4/common/bin/triggers/pull.sh %archiveList%"
```

There are some pull trigger options, but the are not necessary with Aspera. Aspera works best if you give it the max batch size of 1000 and set up 1 or more threads. Note, that each thread will use the max bandwidth you specify, so a single pull-trigger thread is probably all you will want.

The `ascp` user needs to have ssl public keys set up or export `ASPERA_SCP_PASS`.

The `ascp` user should be set up with the target as / with full write access to the volume where the depot files are located. The easiest way to do that is to use the same user that is running the p4d service.

ensure ascp is correctly configured and working in your environment: https://www-01.ibm.com/support/docview.wss?uid=ibm10747281 (search for "ascp connectivity testing")

Standard SDP environment is assumed, e.g P4USER, P4PORT, OSUSER, P4BIN, etc. are set, PATH is appropriate, and a super user is logged in with a non-expiring ticket.

Read the trigger comments for any customization requirements required for your environment.

See also the test version of the script: Section 8.6.22, “pull_test.sh”

See the `/p4/common/bin/triggers/pull.sh` script for details and to customize for your environment.

8.6.22. pull_test.sh

The `/p4/common/bin/pull_test.sh` script is a test script.

THIS IS A TEST SCRIPT - it substitutes for Section 8.6.21, “pull.sh” which uses Aspera’s `ascp` and replaces that with Linux standard `scp` utility. **IT IS NOT INTENDED FOR PRODUCTION USE!!!!**

If you don’t have an Aspera license, then you can test with this script to understand the process.

See the `/p4/common/bin/triggers/pull_test.sh` script for details.

There is a demonstrator project showing usage: https://github.com/rcowham/p4d-edge-pull-demo

8.6.23. purge_revisions.sh

The `/p4/common/bin/purge_revisions.sh` script will allow you to archive files and optionally purge files based on a configurable number of days and minimum revisions that you want to keep. This is useful if you want to keep a certain number of days worth of files instead of a specific number of revisions.

Note: If you run this script with purge mode disabled, and then enable it after the fact, all previously archived files specified in the configuration file will be purged if the configured criteria is met.

Prior to running this script, you may want to disable server locks for archive to reduce impact to end users.

See: https://www.perforce.com/perforce/doc.current/manuals/cmdref/Content/CmdRef/configurables.configurables.html#server.locks.archive

Parameters:

- SDP_INSTANCE - The instance of Perforce that is being backed up. If not set in environment, pass in as argument to script.
- P4_ARCHIVE_CONFIG - The location of the config file used to determine retention. If not set in environment, pass in as argument to script. This can be stored on a physical disk or somewhere in perforce.
- P4_ARCHIVE_DEPOT - Depot to archive the files in (string)
- P4_ARCHIVE_REPORT_MODE - Do not archive revisions; report on which revisions would have been archived (bool - default: true)
- P4_ARCHIVE_TEXT - Archive text files (or other revisions stored in delta format, such as files of type binary+D) (bool - default: false)
- P4_PURGE_MODE - Enables purging of files after they are archived (bool - default: false)

Config File Format

The config file should contain a list of file paths, number of days and minimum of revisions to keep in a tab delimited format.

```
<PATH>    <DAYS>  <MINIMUM REVISIONS>
```

Example:

```
//test/1.txt    10  1
//test/2.txt    1   3
//test/3.txt    10  10
//test/4.txt    30  3
//test/5.txt    30  8
```

Usage

```
/p4/common/bin/purge_revisions.sh <SDP_INSTANCE> <P4_ARCHIVE_CONFIG>
<P4_ARCHIVE_DEPOT> <P4_ARCHIVE_REPORT_MODE (Optional)> 4_ARCHIVE_TEXT (Optional)>
<P4_PURGE_MODE (Optional)>
```

Examples

Run from CLI that will archive files as defined in the config file

```
./purge_revisions.sh 1 /p4/common/config/p4_1.p4purge.cfg archive FALSE
```

Cron job that will will archive files as defined in the config file, including text files

```
30 0 * * * [ -e /p4/common/bin ] && /p4/common/bin/run_if_master.sh ${INSTANCE}
/p4/common/bin/purge_revisions.sh $INSTANCE} /p4/common/config/p4_1.p4purge.cfg
archive FALSE FALSE
```

8.6.24. recover_edge.sh

The `/p4/common/bin/recover_edge.sh` script is designed to rebuild an Edge server from a seed checkpoint from the master while keeping the existing edge specific data.

You have to first copy the seed checkpoint from the master, created with Section 8.6.3, “edge_dump.sh”, to the edge server before running this script. (Alternately, a full checkpoint from the master can be used so long as the edge server spec does not specify any filtering, e.g. does not use ArchiveDataFilter.)

Then run this script on the Edge server host with the instance number and full path of the master seed checkpoint as parameters.

Usage

```
/p4/common/bin/recover_edge.sh <instance> <absolute path to checkpoint>
/p4/common/bin/recover_edge.sh 1 /p4/1/checkpoints/p4_1.edge_syd.seed.ckp.9188.gz
```

8.6.25. replica_cleanup.sh

The `/p4/common/bin/replica_cleanup.sh` script performs the following actions for a replica:

- rotate logs
- remove old checkpoints and journals
- remove old logs

This should be used on replicas for which the `sync_replica.sh` is not used.

Usage

```
/p4/common/bin/replica_cleanup.sh <instance>
/p4/common/bin/replica_cleanup.sh 1
```

8.6.26. replica_status.sh

The `/p4/common/bin/replica_status.sh` script is regularly run by crontab on a replica or edge (using Section 8.4.13, “run_if_replica.sh”).

```
0 8 * * * [ -e /p4/common/bin ] && /p4/common/bin/run_if_replica.sh ${INSTANCE}
/p4/common/bin/replica_status.sh ${INSTANCE} > /dev/null
0 8 * * * [ -e /p4/common/bin ] && /p4/common/bin/run_if_edge.sh ${INSTANCE}
/p4/common/bin/replica_status.sh ${INSTANCE} > /dev/null
```

It performs a `p4 pull -lj` command on the replica to report current replication status, and emails this to the standard SDP administrator email on a daily basis. This is useful for monitoring purposes to detect replica lag or similar problems.

If you are using enhanced monitoring such as p4prometheus then this script may not be required.

Usage

```
/p4/common/bin/replica_status.sh <instance>
/p4/common/bin/replica_status.sh 1
```

8.6.27. request_replica_checkpoint.sh

The `/p4/common/bin/request_replica_checkpoint.sh` script is intended to be run on a standby replica. It essentially just calls 'p4 admin checkpoint -Z' to request a checkpoint and exits. The actual checkpoint is created on the next journal rotation on the master.

Usage

```
/p4/common/bin/request_replica_checkpoint.sh <instance>
/p4/common/bin/request_replica_checkpoint.sh 1
```

8.6.28. rotate_journal.sh

The `/p4/common/bin/rotate_journal.sh` script is a convenience script to perform the following

actions for the specified instance (single parameter):

- rotate live journal
- replay it to the `offline_db`
- rotate logs files according to the settings in `p4_vars` for things like `KEEP_LOGS`

It has several use cases:

- For sites with large, long-running checkpoints, it can be used to schedule journal rotations to occur more frequently than `daily_checkpoint.sh` is run.
- It can be used to trigger checkpoints to run on edge servers.

Usage

```
/p4/common/bin/rotate_journal.sh <instance>
/p4/common/bin/rotate_journal.sh 1
```

8.6.29. submit.sh

The `/p4/common/bin/submit.sh` script is an example submit trigger for External Archive Transfer using pull-archive and edge-content triggers

This is a reference edge-content trigger for use with an Edge/Commit server topology - the Edge server uses this trigger to transmit files which are being submitted to the Commit instead of using its normal file transfer mechanism. This trigger uses Aspera for fast file transfer, and UDP, rather than TCP and is typically much faster, especially with high latency connections.

Companion trigger/script to Section 8.6.21, “pull.sh”

Uses `fstat -Ob` with some filtering to generate a list of files to be copied. Create a temp file with the filename pairs expected by ascp, and then perform the copy.

This configurable must be set:

```
rpl.submit.nocopy=1
```

The edge-content trigger looks like this:

```
EdgeSubmit edge-content //... "/p4/common/bin/triggers/ascpSubmit.sh %changelist%"
```

The `ascp` user needs to have ssl public keys set up or export `ASPERA_SCP_PASS`. The `ascp` user should be set up with the target as / with full write access to the volume where the depot files are located. The easiest way to do that is to use the same user that is running the p4d service.

ensure `ascp` is correctly configured and working in your environment: https://www-01.ibm.com/support/docview.wss?uid=ibm10747281 (search for "ascp connectivity testing")

Standard SDP environment is assumed, e.g P4USER, P4PORT, OSUSER, P4BIN, etc. are set, PATH is appropriate, and a super user is logged in with a non-expiring ticket.

See the test version of this script below: Section 8.6.30, “submit_test.sh”

See the `/p4/common/bin/triggers/submit.sh` script for details and to customize for your environment.

8.6.30. submit_test.sh

The `/p4/common/bin/submit_test.sh` script is a test script.

THIS IS A TEST SCRIPT - it substitutes for Section 8.6.29, “submit.sh” (which uses Aspera) - and replaces `ascp` with Linux standard `scp`. IT IS NOT INTENDED FOR PRODUCTION USE!!!!

If you don’t have an Aspera license, then you can test with this script to understand the process.

See the `/p4/common/bin/triggers/submit_test.sh` for details.

There is a demonstrator project showing usage: https://github.com/rcowham/p4d-edge-pull-demo

8.6.31. sync_replica.sh

The `/p4/common/bin/sync_replica.sh` script is included in the standard crontab for a replica.

It runs `rsync` to mirror the `/p4/1/checkpoints` (assuming instance `1`) directory to the replica machine.

It then uses the latest checkpoint in that directory to update the local `offline_db` directory for the replica.

This ensures that the replica can be quickly and easily reseeded if required without having to first copy checkpoints locally (which can take hours over slow WAN links).

Usage

```
/p4/common/bin/sync_replica.sh <instance>
/p4/common/bin/sync_replica.sh 1
```

8.6.32. templates directory

This sub-directory of `/p4/common/bin` contains some files which can be used as templates for new commands if you wish:

- template.pl - Perl

- template.py - Python
- template.py.cfg - config file for python
- template.sh - Bash

They are not intended to be run directly.

8.6.33. update_limits.py

The `/p4/common/bin/update_limits.py` script is a Python script which is intended to be called from a crontab entry one per hour. It must be wrapped with the `p4master_run` script.

It ensures that all current users are added to the `limits` group. This makes it easy for an administrator to configure global limits on values such as MaxScanRows, MaxSearchResults etc. This can reduce load on a heavily loaded instance.

For more information:

- Maximizing Perforce Helix Core Performance
- Multiple MaxScanRows and similar values

Usage

```
/p4/common/bin/update_limits.py <instance>
/p4/common/bin/update_limits.py 1
```

Chapter 9. Sample Procedures

This section describes sample procedures using the SDP tools described above, given certain scenarios.

9.1. Installing Python3 and P4Python

Python3 and P4Python are useful for custom automation, including triggers.

Installing Python3 and P4Python is best done using packages. First, set up the machine to download packages from Perforce Software, following the guidance appropriate for your platform on the Perforce Packages page.

Then install Python3 and P4Python Packages with the command appropriate for your operating system. For RHEL/Rocky Linux family, use:

```
sudo yum install perforce-p4python3
```

For the Debian/Ubuntu family, use:

```
sudo apt update
sudo apt install perforce-p4python3
```

It is possible to have multiple versions of Python installed, possibly Python 2.7 (the end of the Python 2 line) and various Python 3.x versions, and possibly multiple versions either or both of Python 2 and Python 3. Whether having multiple versions is desirable or necessary depends on what software on the machine uses Python; that discussion is outside the scope of this document. However, being are of this possibility is important for installing in various existing environments.

The behaviors of the `perforce-python3` package install vary slighly depending on what is already installed, and are optimized to avoid disrupting existing software.

- If no prior version of Python 3 exists on the machine when the `perforce-p4python3` package is installed, then the newly installed Python 3 will be established as the default, such that calling `python3` (a symlink) will implicitly refer to the just-installed Python 3 version. **The P4Python module will be available by calling python3**.
- If Python 3.8 exists on the machine when the `perforce-p4python3` package is installed, P4Python wil be added to the existing Python 3.8 install. **The P4Python module will be available by calling python3**.
- If there is already some other version of Python 3.x installed but not 3.8, such as Python 3.6, installing the `perforce-p4python3` package will add a new Python 3.8 installation with the version of Python 3 it uses (e.g. `python3.8`), but it will **not** adjust the existing `python3` symlink. **The P4Python module will not P4Python module available with python3.** You can at that point decide to manually adjust the `python3` symlink to point to `python3.8`, though this has some risk of breaking other things (such as custom triggers) that require the other version of Python3

if it was actively used. Alternately, you can adjust the shebang lines of specific scripts that use P4Python to refer to `python3.8` specifically rather than just `python3`. In any case, avoid using `python2` or just `python`, both of which by convention Python 2.

9.2. Installing CheckCaseTrigger.py

This trigger is very useful to avoid people accidentally checking in files on a case-sensitive server which only differ in case from an existing file (or directory).

This trigger requires `python3` not `python2`

The trigger to install is part of the SDP but by default is in `/p4/sdp/Unsupported/Samples/triggers`.

To install:

1. Install p4python. See: Section 9.1, “Installing Python3 and P4Python”.
2. Copy the trigger and dependencies to approprpiate directory

```
mkdir -p /p4/common/site/bin/triggers
cp /p4/sdp/Unsupported/Samples/triggers/CheckCaseTrigger.py
/p4/common/site/bin/triggers/
cp /p4/sdp/Unsupported/Samples/triggers/P4Trigger.py /p4/common/site/bin/triggers/
```

3. Edit the `shebang` line (first line) at the start of the trigger if necessary, e.g. change to:

```
#!/bin/env python3
```

Usually `python3` is appropriate.

1. Test on an existing (small) changelist:

```
p4 changes -s submitted -m 9
```

   pick a suitable changelist no, e.g. 1234

```
/p4/common/site/bin/triggers/CheckCaseTrigger.py 1234
```

2. Test that it works
   a. Add appropriate line to triggers table:

```
check-case submit-change //test/...
"/p4/common/site/bin/triggers/CheckCaseTrigger.py %changelist%"
```

b. Create test workspace

c. Submit simple `Test.txt`

d. Attempt to submit `test.txt` and check for error

3. Change triggers table to valid version/path:

```
check-case submit-change //... "/p4/common/site/bin/triggers/CheckCaseTrigger.py
%changelist%"
```

9.3. Swarm JIRA Link

Here is an example of linking to cloud JIRA in `config.php`:

```
'jira' => array(
    'host' => 'https://example.atlassian.net/',
    'user' => 'p4jira@example.com',
    'password' => '<API-Token>',
    'link_to_jobs' => 'true',
 ),
```

No need to get complicated with .pem files or 'http_client_options' section. Just specify `https://` prefix as above.

Login to user account on Atlassian URL as above, and then create an API token by going to this URL:

https://id.atlassian.com/manage-profile/security/api-tokens

This curl request tested the API:

```
curl https://example.atlassian.net/rest/api/latest/project --user
p4jira@example.com:<API-TOKEN>
```

The above should list all active projects:

Example JSON response

```
{"expand":"description,lead,issueTypes,url,projectKeys,permissions,insight","self":"ht
tps://example.atlassian.net/rest/api/2/project/11904","id":"11904","key":"ULG","name":
"Ultimate Game"}
```

Check that the provided JIRA account has access to all required projects to be linked (and that it isn't missing some)! See below.

Example list of projects accessible to JIRA account

```
$ curl --user 'p4jira@example.com:<API-TOKEN>'
https://example.atlassian.net/rest/api/latest/project | jq > projects.txt

$ egrep "name|key" projects.txt
egrep "name|key" projects.txt
    "key": "PRJA",
    "name": "Project A",
    "key": "PRJB",
    "name": "Project B",
```

9.4. Reseeding an Edge Server

Perforce Helix Edge Servers are a form of replica that replicates "persistent history" data such as submitted changelists from the master server, while maintaining local databases for "work-in-progress" data, to include user workspaces, lists of files checked out in user workspaces, etc. This separation of persistent and work-in-progress data has significant benefits that make edge servers perform optimally for certain use cases.

When a new edge server is deployed for the first time, it is "seeded" with a special seed checkpoint from the master server. This is done using the SDP `edge_dump.sh` script.

Edge servers need to be reseeded in certain circumstances. When an edge server is reseeded, the latest persistent history from the master server is combined with the latest work-in-progress data from the edge server.

Some occasions that require reseeding include:

- When changing the scope of replication filtering, i.e. if the `*DataFilter` fields of the server spec are changed.
- In some recovery situations involving hardware or other infrastructure failure.
- When advised by Perforce Support.

An article Edge Server Metadata Recovery discusses the manual process in detail. The process outlined in this article is implemented in the SDP with two scripts, `edge_dump.sh` and `recover_edge.sh`.

Key aspects of this implementation:

- No downtime is required for the master server process.
- Downtime for the edge to be reseeded is required. This is kept to a minimum.

9.5. Edge Reseed Scenario

In this sample scenario, an edge server needs to be reseeded.

Sample details about this scenario:

- The SDP instance is `1`.
- The `perforce` operating system runs the p4d process on all machines.
- The `perforce` user's `~/.bashrc` ensures that the shell environment is set automatically on login, by doing: `source /p4/common/bin/p4_vars 1`
- The master server has a ServerID of `master.1` and runs on the machine `bos-helix-01`.
- The edge server has a ServerID of `p4d_edge_syd` and runs on the machine `syd-helix-04`.
- Both the master and edge server are online and actively in use at the start of processing.
- Users of the edge server to be reseeded have been notified about a planned outage.
- No outage is planned or necessary for the master server
- SSH keys are setup for the `perforce` user.

9.5.1. Step 0: Preflight Checks

Make sure the start state is healthy.

As `perforce@bos-helix-01` (the master):

```
verify_sdp.sh 1 -online
```

As `perforce@syd-helix-04` (the edge):

```
verify_sdp.sh 1
```

9.5.2. Step 1: Create New Edge Seed Checkpoint

On the master server, create a new edge seed checkpoint using `edge_dump.sh`. This will contain recent persistent history from the master.

This process uses the `offline_db` rather than P4ROOT, so no downtime is needed.

Creating an edge seed requires that the `offline_db` directory not be interfered with. The `daily_checkpoint.sh` script runs in the crontab of the `perforce` user on the master, and that script must not be run when `edge_dump.sh` runs. Ensure that `edge_dump.sh` is run at a time when it won't conflict with the operation of `daily_checkpoint.sh`. If checkpoints take many hours, consider disabling the crontab for `daily_checkpoint.sh` by commenting it out of the crontab until `edge_dump.sh` completes — but don't forget to re-enable it afterward!

Create the edge seed like so, as `perforce@bos-helix-01` (the master):

```
nohup /p4/common/bin/p4master_run 1 edge_dump.sh 1 p4d_edge_syd < /dev/null >
/p4/1/logs/dump.log 2>&1 &
```

Then monitor until completion with:

```
tail -f $(ls -t $LOGS/edge_dump.*.log | head -1)
```

The edge seed will appear as a file looking something like:

```
/p4/1/checkpoints/p4_1.edge_syd.seed.2035.gz
/p4/1/checkpoints/p4_1.edge_syd.seed.2035.gz.md5
```

When the `.md5` file appears, the edge seed checkpoint is complete.

Notes:

- The `nohup` at the beginning of the command and the `&` at the end ensure this process will continue to run even if the terminal window in which the command was executed disconnects.

9.5.3. Step 2: Transfer Edge Seed

Transfer the edge seed from the master to the edge like so, as `perforce@bos-helix-01` (the master):

```
scp -p /p4/1/checkpoints/p4_1.edge_syd.seed.2035.gz syd-helix-04:/p4/1/checkpoints/.
scp -p /p4/1/checkpoints/p4_1.edge_syd.seed.2035.gz.md5 syd-helix-
04:/p4/1/checkpoints/.
```

9.5.4. Step 3: Reseed the Edge

Reseed the edge. As `perforce@syd-helix-04` (the edge):

```
nohup /p4/common/bin/run_if_edge.sh 1 recover_edge.sh 1
/p4/1/checkpoints/p4_1.edge_syd.seed.2035.gz < /dev/null > /p4/1/logs/rec.log 2>&1 &
```

Notes:

- The `offline_db` of the edge server is removed at the start of processing, but is replaced at the end.
- It is safe for the p4d process of the edge server to be up and running when this process starts. It it is up at the start of processing, it will be shutdown by the `recovered_edge.sh`, but not immediately. The script allows the p4d service to remain in use while the edge seed checkpoint from the master is replayed into the `offine_db`.
- After the edge seed checkpoint has been replayed, the p4d service is shutdown, and then the process of combining persistent and work-in-progress data commences, the essense of the reseed operation.
- After the edge reseed is complete, the p4d process is started. It will then start replcating new data from the master since the time of the edge seed checkpoint creation. The p4d service may

hang and be unresponive for several minutes after it is started. If you choose to monitor closely, when a `p4 pull -lj` on the edge indicates it has caught up to the master, the service is safe to use again.

- The `recover_edge.sh` script continues to run after the service is back online, as it rebuilds the `offline_db` of the edge server.
- On the edge server, the edge server's regular checkpoints land in `/p4/1/checkpionts.edge_syd`. The `/p4/1/checkpoints` folder is used only for holding edge seed checkpoints transferred from the master.
- Typically, all steps described in the process are done on the same day. However, it is OK if the `edge_dump.sh`, seed checkpoint transfer, and `recover_edge.sh` with some time lag between the major steps, typically measured in journal rotations or simply days, with incremental impact on the duration of the recovery step, and so long as the edge seed is not so far behind that the master no longer has numbered journals to feed the edge once it starts.

Reseeding requires that the `offline_db` directory not be interfered with. The `daily_checkpoint.sh` script runs in the crontab of the `perforce` user on the edge server, and that script must not be run when `recover_edge.sh` runs. Ensure that `recover_edge.sh` is run at a time when it won't conflict with the operation of `daily_checkpoint.sh`. If checkpoints take many hours, consider disabling the crontab for `daily_checkpoint.sh` by commenting it out of the crontab until `recover_edge.sh` completes — but don't forget to re-enable it afterward!

This sample procedure does not illustrate using a p4broker service to broadcast a "Down for maintence" message on the edge server. If your SDP installation uses p4brokers on p4d server machines, they can be used to prevent regular users from attempting to access the edge server during the processing of `recover_edge.sh`. This can help prevent users from experiencing a hang, for example, in the time after the edge p4d process starts but before it catches up to the master.

Appendix A: SDP Package Contents and Planning

The directory structure of the SDP is shown below in Figure 1 - SDP Package Directory Structure. This includes all SDP files, including documentation and sample scripts. A subset of these files are deployed to server machines during the installation process.

```
sdp
    doc
    Server (Core SDP Files)
        Unix
            setup (Unix-specific setup)
            p4
                common
                    bin (Backup scripts, etc)
                        triggers (Example triggers)
                    config
                    etc
                        cron.d
                        init.d
                        systemd
                    lib
                    test
    setup (cross platform setup - typemap, configure, etc)
    test (automated test scripts)
```

Figure 1 - SDP Package Directory Structure

A.1. Volume Layout and Server Planning

Figure 2: SDP Runtime Structure and Volume Layout, viewed from the top down, displays a Perforce *application* administrator's view of the system, which shows how to navigate the directory structure to find databases, log files, and versioned files in the depots. Viewed from the bottom up, it displays a Perforce *system* administrator's view, emphasizing the physical volume where Perforce data is stored.

A.1.1. Memory and CPU

Make sure the server has enough memory to cache the **db.rev** database file and to prevent the server from paging during user queries. Maximum performance is obtained if the server has enough memory to keep all of the database files in memory. While the p4d process itself is frugal with system resources such as RAM, it benefits from an excess of RAM due to modern operating systems using excess RAM as file I/O cache. This is to the great benefit of p4d, even though the p4d process itself may not be seen as consuming much RAM directly.

Below are some approximate guidelines for allocating memory.

- 1.5 kilobyte of RAM per file revision stored in the server.
- 32 MB of RAM per user.

INFO: When doing detailed history imports from legacy SCM systems into Perforce, there may be many revisions of files. You want to account for `(total files) x (average number of revisions per file)` rather than simply the total number of files.

Use the fastest processors available with the fastest available bus speed. Faster processors are typically more desirable than a greater number of cores and provide better performance since quick bursts of computational speed are more important to Perforce's performance than the number of processors. Have a minimum of two processors so that the offline checkpoint and back up processes do not interfere with your Perforce server. There are log analysis options to diagnose underperforming servers and improve things. Contact Perforce Support/Perforce Consulting for details.

A.1.2. Directory Structure Configuration Script for Linux/Unix

This script describes the steps performed by the mkdirs.sh script on Linux/Unix platforms. Please review this appendix carefully before running these steps manually. Assuming the three-volume configuration described in the Volume Layout and Hardware section are used, the following directories are created. The following examples are illustrated with "1" as the server instance number.

Directory	*Remarks*
`/p4`	Must be under root (`/`) on the OS volume
`/hxdepots/p4/1/bin`	Files in here are generated by the mkdirs.sh script.
`/hxdepots/p4/1/depots`	
`/hxdepots/p4/1/tmp`	
`/hxdepots/p4/common/config`	Contains p4_<instance>.vars file, e.g. `p4_1.vars`
`/hxdepots/p4/common/bin`	Files from `$SDP/Server/Unix/p4/common/bin`.
`/hxdepots/p4/common/etc`	Contains `init.d` and `cron.d`.
`/hxlogs/p4/1/logs/old`	
`/hxmetadata2/p4/1/db2`	Contains offline copy of main server databases (linked by `/p4/1/offline_db`.
`/hxmetadata1/p4/1/db1/save`	Used only during running of `refresh_P4ROOT_from_offline_db.sh` for extra redundancy.

Next, `mkdirs.sh` creates the following symlinks in the `/hxdepots/p4/1` directory:

Link source	***Link target***	***Command***
`/hxmetadata1/p4/1/db1`	`/p4/1/root`	`ln -s /hxmetadata1/p4/1/root`

Link source	*Link target*	*Command*
/hxmetadata2/p4/1/db2	/p4/1/offline_db	ln -s /hxmetadata1/p4/1/offline_db
/hxlogs/p4/1/logs	/p4/1/logs	ln -s /hxlogs/p4/1/logs

Then these symlinks are created in the /p4 directory:

Link source	*Link target*	*Command*
/hxdepots/p4/1	/p4/1	ln -s /hxdepots/p4/1 /p4/1
/hxdepots/p4/common	/p4/common	ln -s /hxdepots/p4/common /p4/common

Next, `mkdirs.sh` renames the Perforce binaries to include version and build number, and then creates appropriate symlinks.

A.1.3. P4D versions and links

The versioned binary links in `/p4/common/bin` are as below.

For the example of <instance> `1` we have:

```
ls -l /p4/1/bin
p4d_1 -> /p4/common/bin/p4d_1_bin
```

The structure is shown in this example, illustrating values for two instances, with instance #1 using p4d release 2018.1 and instance #2 using release 2018.2.

In /p4/1/bin:

```
p4_1 -> /p4/common/bin/p4_1_bin
p4d_1 -> /p4/common/bin/p4d_1_bin
```

In /p4/2/bin:

```
p4_2 -> /p4/common/bin/p4_2
p4d_2 -> /p4/common/bin/p4d_2
```

In `/p4/common/bin`:

```
p4_1_bin -> p4_2018.1_bin
p4_2018.1_bin -> p4_2018.1.685046
p4_2018.1.685046
```



```
p4_2_bin -> p4_2018.2_bin
p4_2018.2_bin -> p4_2018.2.700949
p4_2018.2.700949
```

```
p4d_1_bin -> p4d_2018.1_bin
p4d_2018.1_bin -> p4d_2018.1.685046
p4d_2018.1.685046
```

```
p4d_2_bin -> p4d_2018.2_bin
p4d_2018.2_bin -> p4d_2018.2.700949
p4d_2018.2.700949
```

The naming of the last comes from:

```
./p4d_2018.2.700949 -V
```

```
Rev. P4D/LINUX26X86_64/2018.2/700949 (2019/07/31).
```

So we see the build number `p4d_2018.2.700949` being included in the name of the p4d executable.

Although this link structure may appear quite complex, it is easy to understand, and it allows different instances on the same server host to be running with different patch levels, or indeed different releases. And you can upgrade those instances independently of each other which can be very useful.

A.1.4. Case Insensitive P4D on Unix

By default `p4d` is case sensitive on Unix for filenames and directory names etc.

It is possible and quite common to run your server in case insensitive mode. This is often done when Windows is the main operating system in use on the client host machines.

In "case insensitive" mode, that means that you should ALWAYS execute `p4d` with the flag `-C1` (or you risk possible table corruption in some circumstances).

The SDP achieves this by executing a simple Bash script which (for instance `1`) is `/p4/1/bin/p4d_1` with contents:

```
#!/bin/bash
P4D="/p4/common/bin/p4d_1_bin"
exec $P4D -C1 "$@"
```

So the above will ensure that `/p4/common/bin/p4d_1_bin` (for instance `1`) is executed with the `-C1` flag.

As noted above, for case sensitive servers, `p4d_1` is normally just a link:

```
/p4/1/bin/p4d_1 -> /p4/common/bin/p4d_1_bin
```

Note for an instance `alpha` (not `1`), the file would be `/p4/alpha/bin/p4d_alpha` with contents:

```
#!/bin/bash
P4D="/p4/common/bin/p4d_alpha_bin"
exec $P4D -C1 "$@"
```

Appendix B: The journalPrefix Standard

The Perforce Helix configurable `journalPrefix` determines where the active journal is rotated to when it becomes a numbered journal file during the journal rotation process. It also defines where checkpoints are created.

In the SDP structure, the `journalPrefix` is set so that numbered journals and checkpoints land on the `/hxdepots` volume. This volume contains critical digital assets that should be reliably backed up and should have sufficient storage for large digital assets such as checkpoints.

B.1. SDP Scripts that set `journalPrefix`

The SDP `configure_new_server.sh`, which applies SDP standards to fresh new `p4d` servers, sets the `journalPrefix` for the master server according to this standard.

The SDP `mkrep.sh` script, which creates new replicas, sets `journalPrefix for replicas according to this standard.

The SDP `mkdirs.sh` script, which initializes the SDP structure, creates a directory structure for checkpoints based on the journalPrefix.

B.2. First Form of `journalPrefix` Value

The first form of the `journalPrefix` value applies to the master server's metadata set. This value is of this form, where `N` is replaced with the SDP instance name:

```
/p4/N/checkpoints/p4_N
```

If the SDP instance name is the default `1`, then files with a `p4_1` prefix would be stored in the `/p4/1/checkpoints` directory on the filesystem. Journal files in that directory would have names like `p4_1.320.jnl` and checkpoints would have names like `p4_1.320.ckp.gz`.

This `journalPrefix` value and the corresponding `/p4/1/checkpoints` directory should be used for the master server. It should also be used for any replica that is a valid failover target for the master server. This includes all *completely unfiltered* replicas of the master, such as `standby` and `forwarding-standby` replicas with a `P4TARGET` value referencing the master server.

A `standby` replica, also referred to as a `journalcopy` replica due to the underlying replication mechanisms, cannot be filtered. Standby replicas are commonly deployed for High Availability (HA) and Disaster Recovery (DR) purposes.

B.2.1. Detail on "Completely Unfiltered"

A "completely unfiltered" replica is one in which:

- None of the `*DataFilter` fields in the replica's server spec are used

- The `p4 pull` command configured to pull metadata from the the replica's `P4TARGET` server, as defined in the replica's `startup.N` configurable, does not use filtering options such as `-T`.
- The replica is not an Edge server (i.e. one with a `Services` value in the server spec of `edge-server`.) Edge servers are filtered by their vary nature, as they exclude various database tables from being replicated.
- The replica's seed checkpoint was created without the `-P ServerID` flag to `p4d`. The `-P` flag is used when creating seed checkpoints for filtered replicas and edge servers.
- The replicas `P4TARGET` server references something other than the master server, such as an edge server.

B.3. Second Form of `journalPrefix` Value

A second form of the `journalPrefix` is used when the replica is filtered, including edge servers. The second form of the `journalPrefix` value incorporates a shortened form of the *ServerID* to indicate that the data set is specific to that *ServerID*. Because the metadata differs from the master, checkpoints for edge servers and filtered replicas are stored in a different directory, and use a prefix that identifies them as separate and divergent from the master's data set. This second form allows checkpoints from multiple edge servers or filtered replicas to be stored on an shared (e.g. NFS-mounted) `/hxdepots` volume.

The second form of journalPrefix is also used if the `/hxdepots` volume, on which checkpoints are stored, is shared (as indicated when the replicas `lbr.replication` value is set to a value of `shared`).

Filtered replicas are a strict subset of the master server's metadata. Edge servers filter some database tables from the master, but also have their own independent metadata (mainly workspace metadata) that varies from the master server and is potentially larger than the master's data set for some tables.

The "shortened form" of the *ServerID* removes the `p4d_` prefix (per Appendix C, *Server Spec Naming Standard*). So, for example an edge server with a *ServerID*` of `p4d_edge_uk` would use just the `edge_uk` portion of the *ServerID* in the `journalPrefix`, which would look like:

```
/p4/N/checkpoints.edge_uk/p4_N.edge_uk
```

If the SDP instance name is the default `1`, then files with a `p4_1.edge_uk` prefix would be stored in the `/p4/1/checkpoints.edge_uk` directory on the filesystem. Journal files in that directory would have names like `p4_1.edge_uk.320.jnl` and checkpoints would have names like `p4_1.edge_uk.320.ckp.gz`.

B.4. Scripts for Maintaining the `offline_db`

The following SDP scripts help maintain the `offline_db`:

- `daily_checkpoint.sh`: The `daily_checkpoint.sh` is used on the master server. When run on the master server, this script rotates the active journal to a numbered journal file, and then maintains the master's `offline_db` using the numbered journal file immediately after it is

rotated.

The `daily_checkpoint.sh` is also used on edge servers and filtered replicas. When run on edge servers and filtered replicas, this script maintains the replica's `offline_db` in a manner similar to the master, except that the journal rotation is skipped (as that can be done only on the master).

- `sync_replica.sh`: The SDP `sync_replica.sh` script is intended to be deployed on unfiltered replicas of the master. It maintains the `offline_db` by copying (via rsync) the checkpoints from the master, and then replays those checkpoints to the local `offline_db`. This keeps the `offline_db` of the replica current, which is good to have should the replica ever need to take over for the master.

INFO: For HA/DR and any purpose where replicas are not filtered, replicas of type `standby` and `forwarding-standby` should displace replicas of type `replica` and `forwarding-replica`.

B.5. SDP Structure and `journalPrefix`

On every server machine with the SDP structure where a `p4d` service runs (excluding broker-only and proxy-only hosts), a structure like the following should exist for each instance:

- A `/hxdepots/p4/N/checkpoints` directory
- In `/p4/N`, and symlink `checkpoints` that links to `/hxdepots/p4/N/checkpoints`, such that it can be referred to as `/p4/N/checkpoints`.

In addition, edge servers and filtered replicas will also have a structure like the following for each instance that runs an edge server or filtered replica:

- A `/hxdepots/p4/N/checkpoints.ShortServerID` directory
- In `/p4/N`, and symlink `checkpoints.ShortServerID` that links to `/hxdepots/p4/N/checkpoints.ShortServerID`, such that it can be referred to as `/p4/N/checkpoints.ShortServerID`.

The SDP `mkdirs.sh` script, which sets up the initial SDP structure, initializes this structure on initial install.

B.6. Replicas of Edge Servers

As edge servers have unique data, they are commonly deployed with their own `standby` replica with a `P4TARGET` value referencing a given edge server rather than the master. This enables faster recovery option for the edge server.

As a special case, a `standby` replica of an edge server should have the same `journalPrefix` value as the edge server it targets. Thus, the *ServerID* baked into the journalPrefix of a replica of an edge is the ServerID of the target edge server, not the replica.

So for example, an edge server with a *ServerID* of `p4d_edge_uk` has a `standby` replica with a *ServerID* of `p4d_ha_edge_uk`. The journalPrefix of that edge should be the same as the edge server it targets, e.g.


```
/p4/1/checkpoints.edge_uk/p4_1.edge_uk
```

B.7. Goals of the `journalPrefix` Standard

Some design of goals this standard:

- Make it so the `/p4/N/checkpoints` folder is reserved to mean checkpoints created from the master server's full metadata set.
- Make the `/p4/N/checkpoints` folder be safe to rsync from the master to any machine in the topology (as may be needed in certain recovery situations for replicas and edge servers).
- Make it so the SDP `/hxdepots` volume can be NFS-mounted across multiple SDP machines safely, such that two or more edge servers (or filtered replicas) could share versioned files, while writing to separate checkpoints directories on a per-ServerID basis.
- Support all replication uses cases, including support for 'Workspace Servers', a name referring to a set of edge servers deployed in in the same location, typically sharing `/hxdepots` via NFS. Use of Workspace Servers can be used to scale Helix Core horizontally for massive user bases (typically several thousand users).

Appendix C: Server Spec Naming Standard

Perforce Helix server specs identify various Helix servers in a topology. Servers can be p4d servers (master, replicas, edges), p4broker, p4p, etc. This standard defines the standard for the server spec names.

C.1. General Form

The general form of a server spec name is:

```
<HelixServerTag>_<ReplicaTypeTag>[<N>]_<SiteTag>
```

C.1.1. Helix Server Tags

The HelixServerTag_ is one of:

- `p4d`: for a Helix Core server (including all distributed architecture usages such as master/replica/edge).
- `p4broker`: A Helix Broker
- `p4p`: A Helix Proxy
- `gconn`: Helix4Git (H4G) Connector
- `swarm`: Helix Swarm

As a special case, the *HelixServerTag* is omitted for the ServerID of the master server spec.

C.1.2. Replica Type Tags

The *ReplicaType* is one of:

- `master.<instance>`: The single master-commit server for a given SDP instance. SDP instance names are included in the ServerID for the master, as they intended to be unique within an enterprise. They must be unique to enable certain cross-instance sharing workflows, e.g. using remote depots and Helix native DVCS features.
- `ha`: High Availability. This indicates a replica that was specifically intended for HA purposes and for use with the `p4 failover` command. It further implies the following:
  - The Services field value is `standby`.
  - The `rpl.journalcopy.location=1` configurable is set, optimized for SDP deployment.
  - The replica is not filtered in any way: No usage of the `-T` flag to `p4 pull` in the replicas startup.*N* configurables, and no usage of `*DataFilter` fields in the server spec.
  - Versioned files are replicated (with an `lbr.replication` value of `readonly`).
  - An HA replica is assumed to be geographically near its P4TARGET server, which can be a master server or an edge server.

    - It may or may not use the `mandatory` option in the server spec. The `ha` tag does not indicate whether the `mandatory` option is used (as this is more transient thing not suitable for baking into a server spec naming standard).

- `ham`: A `ham` replica is the same as an `ha` replica except it does not replicate versioned files. Thus is a *metadata-only* replica that shares versioned files with its P4TARGET server (master or edge) with an `lbr.replication` value of `shared`.
- `fr`: Forwarding Replica (unfiltered) that replicates versioned files.
- `frm`: Forwarding replica (unfiltered) that shares versioned files with its target server rather than replicating them.
- `fs`: Forwarding Standby (unfiltered) that replicates versioned files. This is the same as an `ha` server, except that it is not necessarily expected to be physically near its P4TARGET server. This could be suited for Disaster Recovery (DR) purposes.
- `fsm`: Forwarding standby (unfiltered) that shares versioned files with its target server rather than replicating them. This is the same as a `ham`, except that it is not necessarily expected to be physically near its P4TARGET server.
- `ffr`: Filtered Forwarding Replica. This replica uses some of filtering, such as usage of `*DataFilter` fields of the server spec or `-T` flag to `p4 pull` in the replicas `startup.<N>` configurables. Filtered replicas are not viable failover targets, as the filtered data would be lost.
- `ro` - Read Only replica (unfiltered), replicating versioned files).
- `rom` - Read Only metadata-only replica (unfiltered, sharing versioned files).
- `edge` - Edge servers. (As edge servers are filtered by their nature, they are not valid failover targets).

C.1.2.1. Replication Notes

If a replica does not need to be filtered, we recommend using `journalcopy` replication, i.e. using a replica with a `Services:` field value of `standby` or `forwarding-standby`. Only use non-journalcopy replication when using filtered replicas (and edge servers where there is no choice).

Some general tips:

- The `ha`, `ham` replicas are preferred for High Availability (HA) usage.
- The `fs` and `ro` replicas are preferred for Disaster Recovery (DR) usage.
- Since DR implies the replica is far from its master, replication of archives (rather than sharing e.g. via NFS) may not be practical, and so `rom` replicas don't have common use cases.
- The `fr` type replica is obsolete, and should be replaced with `fs` (using `journalcopy` replication).

C.1.3. Site Tags

The site tag needs to distinguish the data centers used by a single enterprise, and so generally short tag names are appropriate. See Section 5.3.4.1, "SiteTags.cfg"

Each site tag may be understood to be a true data center (Tier 1, Tier 2, etc.), a computer room, computer closet, or reserved space under a developer's desk. In some cases organizations will

already have their own familiar site tags to refer to different sites or data centers; these can be used.

In public cloud deployments, the public cloud provider's region names can be used (e.g. `us-east-1`), or an internal short form (e.g. `awsnva1` for the AWS us-east-1 data center in Northern Virginia, USA.

As a special case, the `<SiteTag>` is omitted for the master server spec.

C.2. Example Server Specs

Here are some sample server spec names based on this convention:

- `master.1`: A master server for SDP instance 1.
- `p4d_ha_chi`: A High Availability (HA) server, suitable for use with `p4 failover`, located in Chicago, IL.
- `p4d_ha2_chi`: A second High Availability server, suitable for use with `p4 failover`, located in Chicago, IL.
- `p4d_ffr_pune`: A filtered forwarding replica in Pune, India.
- `p4d_edge_blr`: An edge server located in Bangalore, India.
- `p4d_ha_edge_blr`: An HA server with P4TARGET pointing to the edge server in Bangalore, India.
- `p4d_edge3_awsnva`: A 3rd edge server in AWS data center in the us-east-1 (Northern Virginia) region.

C.3. Implications of Replication Filtering

Replicas that are filtered in any way are not viable candidate servers to failover to, because any filtered data would be lost.

C.4. Other Replica Types

The naming convention intentionally does not account for all possible server specs available with p4d. The standard accounts only for the distilled list of server spec types supported by the SDP `mkrep.sh` script, which are the most useful and commonly used ones.

C.5. The SDP `mkrep.sh` script

The SDP script `mkrep.sh` adheres to this standard. For more information on creating replicas with this script. See: Section 5.3.4, "Using mkrep.sh".

Appendix D: Frequently Asked Questions

This FAQ lists common questions about the SDP with answers.

D.1. How do I tell what version of the SDP I have?

First, try the standard check. See: Section 1.3, “Checking the SDP Version”.

If that does not display the SDP version, as may happen with older SDP installations, run the SDP Health Check, which will report the correct version reliably. See: Appendix H, *SDP Health Checks*.

Appendix E: Troubleshooting Guide

This appendix lists problems sometimes encountered by SDP users, with guidance on how to analyize and resolve each issue.

Do not hesitate to contact consulting@perforce.com if additional assistance is required.

E.1. Daily_checkpoint.sh fails

1. Check the output of the log file and look for errors:

```
less /p4/1/logs/checkpoint.log
```

Possibilities include:

- Errors from `verify_sdp.sh` - should be self explanatory.
  - Note that it is possible to edit `/p4/common/config/p4_1.vars` and set the value of `VERIFY_SDP_SKIP_TEST_LIST` to include any tests you consider should be skipped - don't overdo this!
- See next section

E.1.1. Last checkpoint not complete. Check the backup process or contact support.

If this error occurs it means the script has found a "semaphore" file which is used to prevent multiple checkpoints running at the same time. This file is (for instance 1) `/p4/1/logs/ckp_running.txt`.

Check if there is a current process running:

```
ps aux | grep daily_checkpoint
```

If you are CERTAIN that there is no checkpoint process running, then you can delete this file and re-run `daily_checkpoint.sh` (or allow it to be run via nightly crontab). If in doubt, contact support!

E.2. Replication appears to be stalled

This can happen for a variety of reasons, most commonly:

- Service user is not logged in to the parent
  - Or there is a problem with ticket or ticket location
- Configurables are incorrect (`p4 configure show allservers`)

- Network connectivity to upstream parent

E.2.1. Resolution

1. Check the output of `p4 pull -lj`, e.g. this shows all is working well:

```
$ p4 pull -lj
Current replica journal state is:       Journal 1237,  Sequence 2680510310.
Current master journal state is:        Journal 1237,  Sequence 2680510310.
The statefile was last modified at:     2022/03/29 14:15:16.
The replica server time is currently:   2022/03/29 14:15:18 +0000 GMT
```

2. This example shows a password error for the service user:

```
$ p4 pull -lj
Perforce password (P4PASSWD) invalid or unset.
Perforce password (P4PASSWD) invalid or unset.
Current replica journal state is:       Journal 1237,  Sequence 2568249374.
Current master journal state is:        Journal 1237,  Sequence -1.
Current master journal state is:        Journal 0,      Sequence -1.
The statefile was last modified at:     2022/03/29 13:05:46.
The replica server time is currently:   2022/03/29 14:13:21 +0000 GMT
```

   a. In case of a password error, try logging in again:

```
p4login -v 1 -service
p4 pull -lj
```

   b. If the above reports an error, then copy and paste the command it shows as executing and try it manually, for example (adjust the server/user ids):

```
/p4/1/bin/p4_1 -p p4master:1664 -u p4admin -s login svc_p4d_edge_ldn
```

If the above is not successful:

3. Review output of `verify_sdp.sh`:

```
/p4/common/bin/verify_sdp.sh 1
grep Error /p4/1/logs/verify_sdp.log
```

   a. Check for errors in the resulting log file:

```
grep Error /p4/1/logs/verify_sdp.log
```

4. Check for errors in the p4d log file:

```
grep -A4 error: /p4/1/logs/log | less
```

5. Check permissions on the tickets file (env var `$P4TICKETS`):

```
ls -al $P4TICKETS
```

   e.g.

```
ls -al /p4/1/.p4tickets
```

If the above doesn't help, then make errors visible/easy to find, assuming instance **1** - run this **on the replica (not commit!)**:

```
sudo systemctl stop p4d_1
cd /p4/1/logs
mv log log.old
sudo systemctl start p4d_1
grep -A4 error: log | less
```

Due to shortened log file, any errors should be easily found. Ask for help (email `support-helix-core@perforce.com`) if not obvious.

E.3. Archive pull queue appears to be stalled

This manifests as the output of `p4 pull -ls` showing an unchanging number of files in the queue - no progress is being made.

```
$ p4 pull -ls
File transfers: 3 active/29 total, bytes: 2338 active/25579 total.
Oldest change with at least one pending file transfer: 1234.
```

This can happen for a variety of reasons, most commonly:

- Non-existent (purged) files (where filetype includes +Sn - where n is number of revisions to keep contents for)
- Non-existent (shelved) files
- Non-existent files with verify problem on master server
- Temporary file transfer problems which exceeded thresholds for auto-retry

E.3.1. Resolutions

1. Retry pull errors

```
p4 pull -R

<wait a short time>

p4 pull -ls
```

2. If the above doesn't fix things then we can check for errors:

```
p4 pull -l | grep -c failed
```

3. If the above is > 0 then we need to investigate in more detail.

E.3.1.1. Remove and re-queue

Save the list of files with errors to a file - like this to allow for spaces in filenames:

```
p4 -F "%rev% %file%" pull -l > pull.errs
cat pull.errs | while read -e r f; do p4 pull -d -r $r -f "$f"; done
```

Finally we can "re-queue" any for re-transfer (note this can take a while for files with many revs):

```
cut -d' ' -f 2,999 pull.errs | sort | uniq | while read -e f; do echo "$f" && p4
verify -qt --only MISSING "$f"; done
```

the `--only MISSING` option requires `p4d` version >= 2021.1 and is much faster - just remove that option with older versions of `p4d`

Then have another look:

```
p4 pull -l
```

E.3.1.2. Check for verify errors on the parent server

On the parent server, check the most recent `p4verify.log` file (typically runs Saturday morning via crontab).

Cross-check any entries in `pull.errs` above - if they are also verify errors on the parent server then you need to resolve that. Consider contacting helix-core-support@perforce.com if you need help. Resolutions may include obliterating lost revisions, or attempting to restore from backup.

E.4. Can't login to edge server

This can happen if the edge server replication has stalled as above.

E.4.1. Resolution

- Try the resolution steps for Section E.2, "Replication appears to be stalled"
- Restart edge server
- Monitor replication and check for any errors

E.5. Updating offline_db for an edge server

If your `daily_checkpoint.sh` jobs on the edge server are failing due to a problem with the `offline_db` or missing edge journals, AND the edge server is otherwise running fine, then consider this option.

Checkpointing the edge will take some time during which the edge will be locked! Schedule this for a convenient time!

E.5.1. Resolution

Assuming instance 1:

- ON EDGE SERVER:

```
source /p4/common/bin/p4_vars 1
p4 admin checkpoint -Z
```

- ON COMMIT SERVER (and at a convenient time to lock edge):

```
source /p4/common/bin/p4_vars 1
p4 admin journal
```

- Monitor edge server checkpoint being created (on EDGE SERVER):

```
p4 configure show journalPrefix
```

Using the output shown by the above command:

```
ls -lhtr /p4/1/checkpoints.<suffix>/*.ckp.*
```

Also you can check for edge being locked (the following may hang):

```
p4 monitor show -al
```

- Then replay the journal on the edge server to the `offline_db`:

```
cd /p4/1/offline_db
mv db.* save/
nohup /p4/1/bin/p4d_1 -r . -jr /p4/1/checkpoints.<suffix>/p4_1.ckp.NNNN.gz >
rec.out &
```

When the above has completed, mark as usable by creating semaphore file:

```
touch /p4/1/offline_db/offline_db_usable.txt
```

E.6. Journal out of sequence in checkpoint.log file

This error is encountered when the offline and live databases are no longer in sync, and will cause the offline checkpoint process to fail. Because the scripts will replay all outstanding journals, this error is much less likely to occur. This error can be fixed by:

- recreating the offline_db: Section 8.4.5, “recreate_offline_db.sh”
- alternatively if that doesn’t work - run the Section 8.4.6, “live_checkpoint.sh” script (note the warnings about locking live database)

E.7. Unexpected end of file in replica daily sync

Check the start time and duration of the Section 8.4.4, “daily_checkpoint.sh” cron job on the master. If this overlaps with the start time of the Section 8.6.31, “sync_replica.sh” cron job on a replica, a truncated checkpoint may be rsync’d to the replica and replaying this will result in an error.

Adjust the replica’s cronjob to start later to resolve this.

Default cron job times, as installed by the SDP are initial estimates, and should be adjusted to suit your production environment.

Appendix F: Starting and Stopping Services

There are a variety of *init mechanisms* on various Linux flavors. The following describes how to start and stop services using different init mechanisms.

F.1. SDP Service Management with the systemd init mechanism

On modern OS's, like RHEL7 & 8, Rocky Linux 8, and Ubuntu >=18.04, and SuSE >=12, the `systemd` init mechanism is used. The underlying SDP init scripts are used, but they are wrapped with "unit" files in `/etc/systemd/system` directory, and called using the `systemctl` interface as `root` (typically using `sudo` while running as the `perforce` user).

On systems where systemd is used, **the service can only be started using the `sudo systemctl` command**, as in this example:

```
sudo systemctl status p4d_N
sudo systemctl start p4d_N
sudo systemctl status p4d_N
```

Note that there is no immediate indication from running the start command that it was actually successful, hence the status command is run after. For best results, wait a few seconds after running the start command before running the status command. (If the start was unsuccessful, a good start to diagnostics would include running `tail /p4/N/logs/log` and `cat /p4/N/logs/p4d_init.log`).

The service should also be stopped in the same manner:

```
sudo systemctl stop p4d_N
```

Checking for status can be done using both the `systemctl` command, or calling the underlying SDP init script directly. However, there are cases where the status indication may be different. Calling the underlying SDP init script for status will always report status accurately, as in this example:

```
/p4/N/bin/p4d_N_init status
```

That works reliably even if the service was started with `systemctl start p4d_N`.

Checking status using the systemctl mechanism is done like so:

```
sudo systemctl start p4d_N
```

If this reports that the service is `active (running)`, such indication is reliable. However, the status indication may falsely indicate that the service is down when it is actually running. This could

occur with older init scripts if the underlying init script was used to start the server rather than using `sudo systemctl start p4d_N` as prescribed. The status indication would only indicate that the service is running if it was started using the systemctl mechanism. As of SDP 2020.1, a safety feature now assures that system is always used if configured.

F.1.1. Brokers and Proxies

In the above examples for starting, stopping, and status-checking of services using either the SysV or `systemd` init mechanisms, `p4d` is the sample service managed. This can be replaced with `p4p` or `p4broker` to manage proxy and broker services, respectively. For example, on a `systemd` system, the broker service, if configured, can be started like so:

```
sudo systemctl status p4broker_1
sudo systemctl start p4broker_1
sudo systemctl status p4broker_1
```

F.1.2. Root or sudo required with systemd

For SysV, having sudo is optional, as the underlying SDP init scripts can be called safely as `root` or `perforce`; the service runs as `perforce`.

If `systemd` is used, by default `root` access (often granted via `sudo`) is needed to start and stop the p4d service, effectively making sudo access required for the `perforce` user. The systemd "unit" files provided with the SDP handle making sure the underlying SDP init scripts start running under the correct operating system account user (typically `perforce`).

F.2. SDP Service Management with SysV init mechanism

On older OS's, like RHEL/CentOS 6, the SysV init mechanism is used. For those, you can the following example commands, replacing *N* with the actual SDP instance name

```
sudo service p4d_N_init status
```

The service can be checked for status, started and stopped by calling the underlying SDP init scripts as either `root` or `perforce` directly:

```
/p4/N/bin/p4d_N_init status
```

Replace `status` with `start` or `stop` as needed. It is common to do a `status` check immediately before and after a `start` or `stop`.

During installation, a symlink is setup such that `/etc/init.d/p4d_N_init` is a symlink to `/p4/N/bin/p4_N_init`, and the proper `chkconfig` commands are run to register the application as a service that will be started on boot and gracefully shutdown on reboot.

On systems using SysV, calling the underlying SDP init scripts is safe and completely interchangeable with using the `service` command being run as `root`. That is, you can start a service with the underlying SDP init script, and the SysV init mechanism will still safely detect whether the service is running during a system shutdown, and thus will perform a graceful stop if p4d is up and running when you go to reboot. The status indication of the underlying SDP init script is absolutely 100% reliable, regardless of how the service was started (i.e. calling the init script directly as `root` or `perforce`, or using the `service` call as `root`.

Appendix G: Brokers in Stack Topology

A preferred methodology is to deploy p4broker processes to control access to p4d servers. In a typical configuration, 100% of user activity gets to p4d thru a p4broker deployed in "stack topology", i.e. a p4broker exists on every machine where p4d is, and access to p4d on any given machine is only via the broker, with a typical setup using firewalls to enforce that concept. There are typically only 3 exceptions:

1. p4d-to-p4d communication (`p4 pull`, `p4 journalcopy`) bypasses the broker
2. Triggers called from p4d run 'p4' commands against the p4d port directly.
3. Admins running 'p4' commands while on the server machine can bypass the broker if they want.

Everything else (to include Proxies, Swarm, Jenkins, any systems integrations, etc.) must go thru the broker.

Using brokers like this makes it straightforward to implement the "Down for Maintenance" concept across an entire global topology. For example, when upgrade p4d services in a global topology, doing the outer-to-inner upgrade procedure, it is best to prevent users from loading the system during the upgrade process.

Using brokers in "stack topology" avoids the significant performance impact of brokers deployed on a different machine than the targeted p4d. While running on the same host, the impact of brokers is relatively small.

Brokers are preferred over p4d command triggers for certain use cases. They’re independent of p4d and can keep p4d safe from rogue usage patterns.

Appendix H: SDP Health Checks

If you need to contact Perforce Support to analyze an issue with the SDP on UNIX/Linux, you can use the `sdp_health_check.sh` script. This script is not included in the SDP, as it can be used with any and all versions of the UNIX/Linux SDP dating back to its origins 2007. This script is acquired with the procedure below.

If your Perforce Helix server machine has outbound internet access, execute the following while logged in as the operating system user that owns the `/p4/common/bin` directory (typically `perforce` or `p4admin`):

```
cd
```

```
curl -L -s -O https://swarm.workshop.perforce.com/projects/perforce-software-
sdp/download/tools/sdp_health_check.sh
chmod +x sdp_health_check.sh
```

```
./sdp_health_check.sh
```

If your Perforce Helix server machine does not have have outbound internet access, acquire the `sdp_health_check.sh` file from a machine that does have outbound internet access, and then somehow get that file to your Perforce Helix server machine.

If you have multiple server machines with SDP, possibly including machines running P4D replicas or edge servers, P4Proxy or P4Broker servers, run the health on al machines of interest.

The `sdp_health_check.sh` script will produce a log file that can be provided to Perforce Support to help diagnose configuration issues and other problems. The script has these characteristics:

- It is always safe to run. It does only analysis and reporting.
- It does only fast checks, and has no interactive prompts. Some log files are captured such as checkpoint.log, but not potentially large ones such as the p4d server log.
- It requires no command line arguments.
- It works for any and all UNIX/Linux SDP version since 2007.