

Perforce P4 Server Deployment Package (for UNIX/Linux)

Perforce Professional Services

Version v2025.2, 2026-07-21

Table of Contents

Preface	1
1. Overview	2
1.1. Using this Guide	2
1.2. Getting the SDP	3
1.3. Checking the SDP Version	3
2. Setting up the SDP	4
2.1. Terminology Definitions	4
2.1.1. Process	4
2.1.2. Instance	4
2.1.3. Server machine	5
2.1.4. Server spec	5
2.1.5. Server	5
2.2. Pre-Requisites	5
2.3. Volume Layout and Hardware	6
3. Maintaining the SDP	7
3.1. The Offline Database	7
3.2. Scripts for Maintaining the <code>offline_db</code>	8
3.3. Parallel Checkpoints	8
3.4. Backup procedures	9
3.4.1. Metadata checkpoints	9
3.4.2. Backup of the partition containing depots, checkpoints, and the SDP configuration	9
3.5. Notifications	10
3.5.1. Configuration	10
3.5.2. Notifications to monitor	10
3.5.2.1. Daily Checkpoint	10
3.5.2.2. Verify	10
3.5.2.3. Sync Replica	10
3.6. Disk usage	11
4. Installing the SDP	12
4.1. Using <code>install_sdp.sh</code>	12
4.1.1. Planning	12
4.1.2. STEP 1: Configure storage.	12
4.1.3. STEP 2: Download the <code>install_sdp.sh</code> script	13
4.1.4. STEP 3: Generate install configuration file.	13
4.1.5. STEP 4: Modify install configuration file.	13
4.1.6. STEP 5: Install SDP (Dry Run).	14
4.1.7. STEP 6: Install SDP (Live Run)..	14
4.1.8. STEP 7: Install a license file.	14

4.1.9. Start Your Helix Core Server	15
4.2. Using makedirs.sh	15
4.2.1. Use of SSL	15
4.2.1.1. Changing SSL Certificates	15
4.2.1.2. Configuration script makedirs.cfg	16
4.2.2. SDP Init Scripts	17
4.2.2.1. Configuring systemd	18
Configuring systemd for p4d	18
Configuring systemd for p4p	19
Configuring systemd for p4dtg	19
Configuring systemd p4broker - multiple configs	19
4.2.2.2. Enabling systemd under SELinux	22
4.2.2.3. Configuring SysV Init Scripts	23
4.2.3. Configuring Automatic Service Start on Boot	23
4.2.3.1. Automatic Start for Systems using systemd	24
4.2.3.2. For systems using the SysV init mechanism	24
4.2.4. SDP Crontab Templates	24
4.2.5. Completing Your Server Configuration	24
4.2.6. Validating your SDP installation	25
4.3. Local SDP Configuration	26
4.3.1. Load Order	27
4.4. Setting your login environment for convenience	27
4.5. Configuring protections, file types, monitoring and security	27
4.6. Operating system configuration	28
4.6.1. Configuring email for notifications	28
4.6.2. Swarm Email Configuration	29
4.6.3. Configuring PagerDuty for notifications	29
4.6.3.1. Prerequisites	29
4.6.3.2. SDP Configuration	30
4.6.3.3. Optional variables	30
Example Additional Context Configuration	30
4.6.4. Configuring AWS Simple Notification Service (SNS) for notifications	31
4.6.4.1. Prerequisites	31
4.6.4.2. SDP Configuration	31
4.6.4.3. Example IAM Policy	31
4.7. Other server configurables	32
4.8. Archiving configuration files	32
4.9. Installing P4 Code Review Triggers	32
5. Backup, Recovery, and Replication	35
5.1. Backup Mechanisms	35
5.1.1. SDP Backups in SDP OS Package Structure	35

5.1.2. SDP Recovery in SDP OS Package Structure	35
5.1.3. SDP Backups in SDP Classic Structure	36
5.2. Planning for HA and DR	36
5.2.1. Creating a Failover Replica for Commit or Edge Server	37
5.2.2. What is a Failover Replica?	38
5.2.3. Mandatory vs Non-mandatory Standbys	38
5.3. Full One-Way Replication	39
5.3.1. Replication Setup	39
5.3.2. Replication Setup for Failover	40
5.3.3. Pre-requisites for Failover	40
5.3.4. Using mkrep.sh	40
5.3.4.1. SiteTags.cfg	41
5.3.4.2. Output of mkrep.sh	41
5.3.5. Additional Replication Setup	41
5.3.6. SDP Installation	42
5.3.6.1. SSH Key Setup	42
5.4. Recovery Procedures	42
5.4.1. Recovering a master server from a checkpoint and journal(s)	42
5.4.2. Recovering a replica from a checkpoint	43
5.4.3. Recovering from a tape backup	44
5.4.4. Failover to a replicated standby machine	44
6. Upgrades	45
6.1. Upgrade Order: SDP first, then Helix P4D	45
6.2. SDP and P4D Version Compatibility	45
6.3. Upgrading the SDP	45
6.3.1. Sample SDP Upgrade Procedure	46
6.3.1.1. Sample SDP Upgrade in Classic Structure	46
6.3.1.2. Sample SDP Upgrade in OS Package Structure	47
6.3.2. SDP Upgrades in the Future	47
6.3.3. SDP Legacy Upgrade Procedure	48
6.4. Upgrading Helix Software with the SDP	48
6.4.1. Get Latest Helix Binaries	48
6.4.2. Upgrade Each Instance	48
6.4.3. Global Topology Upgrades - Outer to Inner	48
7. Maximizing Server Performance	51
7.1. Ensure Transparent Huge Pages (THP) is turned off	51
7.2. Putting server.locks directory into RAM	52
7.3. Installing monitoring packages	54
7.4. Optimizing the database files	54
7.5. P4V Performance Settings	55
7.6. Proactive Performance Maintenance	55

7.6.1. Limiting large requests	55
7.6.2. Offloading remote syncs	55
8. Tools and Scripts	56
8.1. General SDP Usage	56
8.1.1. Linux	56
8.1.2. Monitoring SDP activities	57
8.2. Upgrade Scripts	57
8.2.1. get_helix_binaries.sh	57
8.2.2. upgrade.sh	61
8.2.3. sdp_upgrade.sh	72
8.3. Legacy Upgrade Scripts	80
8.3.1. clear_depot_Map_fields.sh	81
8.4. Core Scripts	82
8.4.1. p4_vars	83
8.4.2. p4_<instance>.vars	83
8.4.3. p4master_run	83
8.4.4. daily_checkpoint.sh	84
8.4.5. keep_offline_db_current.sh	84
8.4.6. live_checkpoint.sh	85
8.4.6.1. live_checkpoint.sh on a commit server	85
8.4.6.2. live_checkpoint.sh on an edge server	86
8.4.7. mkrep.sh	86
8.4.8. p4verify.sh	93
8.4.9. p4login	107
8.4.10. p4d_<instance>_init	110
8.4.11. recreate_offline_db.sh	110
8.4.12. refresh_P4ROOT_from_offline_db.sh	111
8.4.13. run_if_master.sh	111
8.4.14. run_if_edge.sh	111
8.4.15. run_if_replica.sh	111
8.4.16. run_if_master/edge/replica.sh	112
8.4.17. sdp_health_check.sh	112
8.5. More Server Scripts	113
8.5.1. p4.crontab	113
8.5.2. verify_sdp.sh	113
8.6. SDP Trigger Scripts	119
8.6.1. enforce_change_type.sh	120
8.6.2. pull.sh	120
8.6.3. pull_test.sh	121
8.6.4. sdp_info.sh	121
8.6.5. SetWsOptionsAndView.py	122

8.6.6. SetWsOptions.py	122
8.7. Other Scripts and Files	123
8.7.1. backup_functions.sh	123
8.7.2. broker_rotate.sh	123
8.7.3. ccheck.sh	123
8.7.4. edge_dump.sh	129
8.7.5. edge_vars	129
8.7.6. edge_shelf_replicate.sh	129
8.7.7. load_checkpoint.sh	129
8.7.8. gen_default_broker_cfg.sh	140
8.7.9. gen_sudoers.sh	140
8.7.10. journal_watch.sh	142
8.7.11. kill_idle.sh	143
8.7.12. mkdirs.sh	143
8.7.13. opt_perforce_sdp_backup.sh	152
8.7.14. p4d_base	155
8.7.15. p4broker_base	155
8.7.16. p4ftpd_base	155
8.7.17. p4p_base	156
8.7.18. p4pcm.pl	156
8.7.19. p4review2.py	157
8.7.20. proxy_rotate.sh	157
8.7.21. p4sanity_check.sh	158
8.7.22. p4dstate.sh	158
8.7.23. ps_functions.sh	158
8.7.24. purge_revisions.sh	159
8.7.25. recover_edge.sh	160
8.7.26. replica_cleanup.sh	160
8.7.27. replica_status.sh	161
8.7.28. request_replica_checkpoint.sh	161
8.7.29. rotate_journal.sh	161
8.7.30. submit.sh	162
8.7.31. submit_test.sh	163
8.7.32. sync_replica.sh	163
8.7.33. templates directory	164
8.7.34. update_limits.py	164
9. Sample Procedures	166
9.1. Installing Python3 and P4Python	166
9.2. Installing CheckCaseTrigger.py	167
9.3. Swarm JIRA Link	168
9.4. Reseeding an Edge Server	169

9.5. Edge Reseed Scenario	170
9.5.1. Step 0: Preflight Checks	170
9.5.2. Step 1: Create New Edge Seed Checkpoint	171
9.5.3. Step 2: Transfer Edge Seed	171
9.5.4. Step 3: Reseed the Edge	172
Appendix A: SDP Package Contents and Planning	174
A.1. SDP Classic and OS Package Structures	174
A.2. SDP Runtime Structure	176
A.2.1. The Site Directory	177
A.3. P4D versions and links	177
A.4. Storage Volumes Layout	178
A.4.1. Storage Volumes for a Helix Core Server	178
A.4.2. More About HxDepsots	179
A.4.2.1. Using Multiple Depot Storage Volumes	180
A.4.3. More About HxCheckpoints	181
A.4.4. More About HxMetadata	182
A.4.5. More about HxLogs	182
A.4.6. Storage Volumes for a Helix Proxy	183
A.4.7. Storage Volumes for a Helix Broker	183
A.5. Memory and CPU	184
A.6. Case Insensitive P4D on UNIX/Linux	184
Appendix B: The journalPrefix Standard	186
B.1. SDP Scripts that set <code>journalPrefix</code>	186
B.2. First Form of <code>journalPrefix</code> Value	186
B.2.1. Detail on "Completely Unfiltered"	186
B.3. Second Form of <code>journalPrefix</code> Value	187
B.4. SDP Structure and <code>journalPrefix</code>	187
B.5. Replicas of Edge Servers	188
B.6. Goals of the <code>journalPrefix</code> Standard	188
Appendix C: Server Spec Naming Standard	189
C.1. General Form	189
C.1.1. Commit Server Spec	189
C.1.2. Helix Server Tags	190
C.1.3. Replica Type Tags	190
C.1.3.1. Replication Notes	191
C.1.4. Site Tags	191
C.2. Example Server Specs	192
C.3. Implications of Replication Filtering	192
C.4. Other Replica Types	192
C.5. The SDP <code>mkrep.sh</code> script	192
Appendix D: Frequently Asked Questions	193

D.1. How do I tell what version of the SDP I have?	193
D.2. How do I change the super user password?	193
D.3. How do I change from using cleartext to encoded passwords?	195
D.4. Can I remove the perforce user?	196
D.5. Can I clone a VM to create a standby replica?	196
Appendix E: Troubleshooting Guide	199
E.1. Daily_checkpoint.sh fails	199
E.1.1. Last checkpoint not complete. Check the backup process or contact support.	199
E.2. Replication appears to be stalled	199
E.2.1. Resolution	200
E.2.2. Make Replication Errors Visible	201
E.2.3. Remove state file	201
E.3. Archive pull queue appears to be stalled	202
E.3.1. Resolutions	202
E.3.1.1. Remove and re-queue	202
E.3.1.2. Check for verify errors on the parent server	203
E.4. Can't login to edge server	203
E.4.1. Resolution	203
E.5. Updating offline_db for an edge server	203
E.5.1. Resolution	203
E.6. Journal out of sequence in checkpoint.log file	204
E.7. Unexpected end of file in replica daily sync	205
E.8. Recovering from an interrupted checkpoint (causes replication to fail)	205
E.8.1. Resolution	205
E.9. Server won't start: Error: DB check with 'p4d -xvU' failed. Database integrity is in question.....	207
E.9.1. Resolution	208
Appendix F: Starting and Stopping Services	210
F.1. SDP Service Management with the systemd init mechanism	210
F.1.1. Brokers and Proxies	211
F.1.2. Root or sudo required with systemd	211
F.2. Force Start Mechanism for P4D	211
F.3. Journal Corruption Check	213
F.4. SDP Service Management with SysV init mechanism	213
Appendix G: Brokers in Stack Topology	215
Appendix H: SDP Health Checks	216
Appendix I: More Detail on install_sdp.sh	217
I.1. Sample configuration file <code>sdp_install.cfg</code>	217
I.2. install_sdp.sh	221
Appendix J: More Detail on mkdirs.sh	235

Preface

The Server Deployment Package (SDP) is the implementation of Perforce best practices 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 with:

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

This guide is intended to provide instructions for 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](#) or [Helix Core Documentation](#).

Related Guides:

- [SDP Release Notes](#)
- [SDP Failover Guide](#)
- [SDP Guide for Windows](#)

Please Give Us Feedback

Perforce welcomes feedback from our users. Please send any suggestions for improving this document or the SDP to p4consulting@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 3, *Maintaining the SDP*](#) covers administrative duties associated with keeping an installation of the SDP in good shape.

[Chapter 4, *Installing the SDP*](#) consists of what you need to know to install SDP and setup a Helix Core Server, Broker, or Proxy.

[\[*backup_replication_and_recovery*\]](#) 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 SDP as well as upgrading Helix Core binaries such as `p4d` and `p4p`.

[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 an 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://workshop.perforce.com/files/guest/perforce_software/sdp/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://workshop.perforce.com/download/guest/perforce_software/sdp/downloads/sdp.Unix.
tgz
```

or

```
wget
https://workshop.perforce.com/download/guest/perforce_software/sdp/downloads/sdp.Unix.
tgz
```

1.3. Checking the SDP Version

Once installed, the SDP `Version` file is stored 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.

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

Key terms are defined in this section.

2.1.1. Process

A *process* is a running operating system process with a process identifier (PID) known to the operating system. It should normally be qualified as to what type of process it is:

- **p4d process** - a running p4d process with its 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-standby**, **forwarding-replica**, **edge-server** etc (see **p4 server** and the **Services:** field in the [P4 Command Reference](#)).
- **p4p process** - proxy instance talking to a single upstream p4d instance
- **p4broker process** - p4broker talking to a single upstream p4d instance

2.1.2. Instance

An *instance* is a logically independent set of Helix Core data and metadata, represented by entities such as changelist numbers and depot paths, and existing on 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, file histories, and changelists.

Some facts about SDP instance names:

- The default SDP instance name is simply **1** (the digit 'one').
- Any alphanumeric name can be used (e.g. **internal**). It is mainly of interest to administrators, not regular users. Underscores are also allowed; dots should not be used in SDP instance names.
- As they are typed often in various admin operational tasks:
 - Instance names are best kept short. A length of 1-5 characters is recommended, with a maximum of 32 characters.
 - Lowercase letters are preferred and required at some sites, but not required by the SDP.
- SDP instance names can be any alphanumeric name. Underscores (**_**) and dashes (**-**) are also

allowed. Dots, spaces, and other special characters should not be used in SDP instance names.

- 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, edge and replica machines, the instance exists as `/p4/N`, where `N` is the instance name.
- There can be more than one instance on a machine.

2.1.3. Server machine

A *server machine* is a host machine (virtual or physical) with operating system and on which any number of p4d or other processes may be running.

2.1.4. Server spec

A *server spec* (or *server specification*) is the entity managed using the `p4 server` command (and the plural `p4 servers` to list all of them) - [P4 Command Reference](#).

2.1.5. Server

The term *server* may mean any one of:

- A Server machine.
- The p4d process. This is perhaps the most common usage - tend to assume this unless otherwise defined.
- A defined *server spec* within a p4d data set, as listed with the `p4 servers` command.
- Any other type of instance!



The phrase "p4d server" is unclear as to whether you are talking about a p4d process, or a server machine on which the p4d process runs, 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!

2.2. Pre-Requisites

1. The Helix Core binaries (p4d, p4, p4broker, p4p) have been downloaded (see [Chapter 4, Installing the SDP](#))
2. `sudo` access is required for initial installation (and at least partial sudo is required for actions such as starting and stopping systemd services)
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. Note prior versions are typically past their End of Life.
 - Ubuntu 20.04 LTS (focal)
 - Ubuntu 22.04 LTS (jammy)
 - Ubuntu 24.04 LTS (noble)
 - Red Hat Enterprise Linux (RHEL) 8.x (and compatible)
 - Rocky Linux 8.x
 - Alma Linux 8.x
 - Red Hat Enterprise Linux (RHEL) 9.x (and compatible)
 - Rocky Linux 9.x
 - Alma Linux 9.x
 - CentOS 8 (not recommended for production; Rocky Linux replaces CentOS 8)
 - SUSE Linux Enterprise Server 15

2.3. Volume Layout and Hardware

As can be expected from a version control system (which includes a database), good disk (storage) management is key to maximizing data integrity and performance. Perforce recommends 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 Core data 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, performant, and widely used.

See the [Section A.4, “Storage Volumes Layout”](#) for guidance on storage configuration.

Chapter 3. Maintaining the SDP

These are regular (automated) tasks that are setup to maintain your installation.

3.1. The Offline Database

The P4 Server supports maintenance of separate copy of the live database, referred to as the "offline database" and often referred to as `offline_db`, the name of the directory that contains the offline database.

The offline database is a tremendously valuable resource for providing data redundancy as well as reducing downtime in recovery scenarios. Further, it provides a set of files that can be scanned with long-running operations that will not impact the live running service, such as creating an edge server seed checkpoint.

The offline database is an already recovered data set that is kept warm and nearly current, typically no more than 24 hours out of date. To bring it from *nearly* current to *completely* current in a recovery situation, a relatively fast operation referred to as a "journal replay" can be done to bring the `offline_db` up to date. A goal with SDP design is to disconnect checkpoint duration times (which can grow to be hours long) from recovery times by having the `offline_db` ready to use in place of a checkpoint. An offline database is an already-replayed checkpoint, just waiting to be used, and is the exact equivalent of replaying from a checkpoint as far as the data goes.

The live database lives in the P4ROOT directory in the form of a set of many (over 100) database table files, `db.*` files, such as `db.domain`, `db.config`, etc. A P4 *checkpoint* (verb) operation produces a *checkpoint* (noun), which is a guaranteed transactionally consistent, point-in-time reference to the state of the entire database. A checkpoint is the primary digital asset that must be backed up.



Because the live database tables in P4ROOT can be written to at any time, they cannot safely be backed up directly. **DO NOT** backup the P4ROOT data set directly.

Part of the checkpoint operation involves ensuring that all database tables are locked for the duration of the operation, providing the point-in-time recovery capability. Locking the live database tables causes impact to users, and of course it is best to avoid impacting users. To obtain a checkpoint while avoiding impact to users requires an *offline database*.

A "live checkpoint" locks the live database tables in P4ROOT to produce a checkpoint, and then replays that new checkpoint into the `offline_db`. This replay operation is essentially exercising the same commands with the same data that could be used in a recovery situation. Ideally, it is done only once early in the life of a data set, when it takes a mere fraction of a second to run. As a data set grows over the years, checkpoint duration grows.

An "offline checkpoint" process takes advantage of the offline database created initially with the live checkpoint operation. This shifts the locking operation from the live database to the offline database. The operation can then run for a long period of time without directly impacting users.

This slide deck illustrates the technical process: [SDP Offline Checkpoint Illustration](#).



The `offline_db` directory should contain a small text file called `offline_db_usable.txt`. Before using an offline database, be sure that file exists and has a recent timestamp (e.g. less than 24 hours old). If the file is missing **DO NOT** attempt to use the offline database for recovery operations. If the file exists but is outdated, contact Perforce Technical Support for advice.

3.2. Scripts for Maintaining the `offline_db`

The following SDP scripts help maintain the `offline_db`:

- `daily_checkpoint.sh`: Creates daily checkpoints and maintains the `offline_db` on commit and edge servers. See: [Section 8.4.4, “daily_checkpoint.sh”](#).
- `keep_offline_db_current.sh`: An alternative to `sync_replica.sh` for use on standby replicas. See: [Section 8.4.5, “keep_offline_db_current.sh”](#).
- `live_checkpoint.sh`: Takes a checkpoint of live P4ROOT and seed or reset the `offline_db`. See: [Section 8.4.6, “live_checkpoint.sh”](#).
- `rotate_journal.sh`: Does a journal rotation and replay of outstanding journals to the `offline_db`. See: [Section 8.7.29, “rotate_journal.sh”](#).
- `sync_replica.sh`: Keeps checkpoints sync'd and `offline_db` current on replicas. See [Section 8.7.32, “sync_replica.sh”](#).

3.3. Parallel Checkpoints

To enable parallel checkpoints, edit the SDP Instance Vars file, `/p4/common/config/p4_N.vars`.

In this file:

Set `DO_PARALLEL_CHECKPOINTS=0` to disable parallel checkpoints.

Set `DO_PARALLEL_CHECKPOINTS=N` to enable parallel checkpoints, where `N` is a positive integer indicating the number of parallel threads to use. As a special case, setting `DO_PARALLEL_CHECKPOINTS=1` is the same as setting `DO_PARALLEL_CHECKPOINTS=4`. The `N` value is passed to the `p4d` with the `'-N'` parameter when doing checkpoint create, dump, and replay options with `'p4d -jcp[m]'`, `'p4d -jdp[m]'`, and `'p4d -jrp'`, respectively.

If parallel checkpoints are enabled: * The `live_checkpoint.sh` will create live checkpoints using `p4d -jcp[m]`, and replay using `p4d -jrp`. * The `daily_checkpoint.sh` will create offline checkpoints using `p4d -jdp[m]`, and replay using `p4d -jrp`. * The `recover_offline_db.sh` will look for checkpoint directories rather than singular checkpoint files, and replay with `p4d -jrp`. * The `refresh_P4ROOT_from_offline_db.sh` replays using `p4d -jrp`.

The `db.checkpoint.threads` configurable is ignored with the scripts, as the `'-N'` parameter overrides configurable.

Parallel checkpoints became available in `p4d` 2022.2. This setting is ignored if the server version is earlier than that.

3.4. Backup procedures

A P4 Server's purpose is to enable collaboration while maintaining long-running history of your development activity. As such, it is critical to take reliable backups to preserve your data integrity.

A critical concept is that SDP operations automate the *preparation* for backup, but does not actually perform a backup. The SDP makes backup straightforward by ensuring all critical data assets that must be backed up are stored on the HxDeps volume. However, to achieve the actual backup, this volume must be somehow preserved or copied. The actual means of backup is outside the scope of the SDP. Some common mechanisms are:

- If using a file server with snapshot capability, such as with NetApp (available for on-prem and in some public clouds), snapshots of the data volume on the filer can be performed.
- For virtualized environments, to include public and private clouds as well as on-prem virtualization, typically have machine-level backup capabilities. This can be used to backup the entire machine. Backups of metadata via full-machine snapshots are NOT suitable for recovery; they may be included in full-machine backups but only HxDeps data should be relied upon. In AWS, for example, an AWS Lifecycle Policy can be configured to backup the entire machine. In Azure, a Vault can be established to store machine backups.
- Backup utilities and third party solutions can (and should) backup the HxDeps volume.
- While suboptimal, backups via copy of HxDeps data to other machines is at least a backup of some kind. This type of solution generally does not support going back in time.

3.4.1. Metadata checkpoints

The SDP contains scripts and a default `crontab` which will create daily checkpoints with no downtime. The script [Section 8.4.4, “daily_checkpoint.sh”](#) accomplishes this by rotating the journal, replaying it into the `offline_db` directory, and checkpointing the `offline_db` directory. The resulting checkpoints, rotated journals, and checkpoint checksum files can be found in `/p4/<instance>/checkpoints` or `/p4/<instance>/checkpoints.<shortServerID>` (for edge servers and filtered replica servers).

It is difficult to overstate the importance of regular checkpoints! Perforce metadata (contained in the `db.*` files) is in a constant state of flux and being updated, and a checkpoint is the most reliable point of recovery for a commit server. Attempts to back up the `root` directory with `cp` or `rsync` will result in a metadata set that is probably inconsistent and corrupt. Simple backups of the root directory are insufficient.

3.4.2. Backup of the partition containing depots, checkpoints, and the SDP configuration

There are three important parts to an SDP installation of Perforce: Metadata checkpoints and numbered journals, archive storage (back-end version file storage), and configuration. A standard SDP installation will have all three of these on the `/hxdepts` partition or equivalent. Whatever your server backup strategy is, ensure that you are taking regular backups or snapshots of `/hxdepts`.

3.5. Notifications

The SDP contains the framework to allow your server to communicate its automated maintenance activities, both successes and failures. It is important to ensure that the SDP is properly configured to send emails to the right people, and that the right people are monitoring their emails. We also recommend the use of [P4Prometheus](#) and associated scripts and dashboards using Prometheus and Grafana, together with Alertmanager for best practice monitoring of your installation.

3.5.1. Configuration

Setting up mailx, postfix, mailutils, or s-nail will allow your server to send out emails to your administrative team. Details can be found in [Section 4.6.1, “Configuring email for notifications”](#).

To tell the SDP whom to mail, you will need to set that in the file `/p4/common/config/p4_<instance.vars>` on a per-instance basis. The relevant lines are:

```
export MAILTO=P4AdminList@p4demo.com
```

```
export MAILFROM=P4Admin@p4demo.com
```

The `MAILTO` value can be a distribution group like `administrators@company.net`, a single recipient like `bruno@company.net`, or a comma delimited list like `bruno@company.net,mary@company.net,pat@company.net`.

The `MAILFROM` value can be a valid email address, or a placeholder like `do-not-reply@company.net`.

3.5.2. Notifications to monitor

Your administrator should be aware of the emails that the SDP will be sending on a regular basis. Be careful to not simply redirect them into an unmonitored folder!

3.5.2.1. Daily Checkpoint

Probably the most important notification to follow, the daily checkpoint job lets you know that your metadata is backed up. Any error messages should be investigated.

3.5.2.2. Verify

By default, the SDP will run a verify on all your back-end versioned file storage on a weekly basis. It is possible that errors or warnings will creep into an instance as time goes on. These should be investigated, but they are often not mission-critical.

3.5.2.3. Sync Replica

If you are in a Helix topology that contains replicas or edges, those machines will have their own automated jobs that synchronize checkpoints from the commit server, and keep the metadata in sync. To maintain a healthy topology, these emails should also be investigated if they contain errors.

3.6. Disk usage

Running out of disk is never fun. You should keep an eye on your disk usage, expanding when needed. A default SDP instance has the following configurables set:

```
filesys.P4JOURNAL.min = 5G
```

```
filesys.P4ROOT.min = 5G
```

```
filesys.depot.min = 5G
```

These settings will cause Perforce to halt when they discover that free disk space is under 5G on the specified partition. This will spare you from corruption if Perforce tries to write to a database and isn't able to finish. *However*, there are some edge cases where disk usage can still be disruptive. If your total partition size is 5G or lower, Perforce will halt automatically even if 5G was your intended partition size. Monitoring and expanding your storage space is an important part of maintenance.

Chapter 4. Installing the SDP

If you are installing SDP on a fresh new server machine where SDP has not been installed previously, see [Section 4.1, “Using install_sdp.sh”](#). This applies whether you are setting up a Helix Core Server (P4D) of any kind (commit, standby, edge, etc.) as well as if you intend to install a standalone Helix Proxy or Helix Broker.

If you are adding a new SDP instance on a machine where SDP has already been installed, see [Section 4.2, “Using mkdirs.sh”](#). This applies if the `/p4/sdp` directory structure already exists on the machine and you intend to add a new data set with a new `/p4/N` folder, for operating a Helix Core Server (P4D) of any kind (commit, standby, edge, etc.) as well as if you intend to add a standalone Helix Proxy or Helix Broker to the existing machine.

For clarity on use of the term 'instance', see [Section 2.1.2, “Instance”](#) in the section [Section 2.1, “Terminology Definitions”](#).

4.1. Using install_sdp.sh

Use `install_sdp.sh` if you are installing SDP on a fresh new server machine where SDP has not been installed previously. This applies whether you are setting up a Helix Core Server (P4D) of any kind (commit, standby, edge, etc.) as well as if you intend to install a standalone Helix Proxy or Helix Broker.

Following are instructions for installing a Helix Core P4D Server. More details appear in the [Appendix I, More Detail on install_sdp.sh](#).

4.1.1. Planning

Review the sample configuration file [Section I.1, “Sample configuration file sdp_install.cfg”](#), which captures various aspects you need to consider before installing a new Helix Core server, such as the desired case sensitivity, SDP instance name, type, port numbers, use of SSL, etc. This same configuration file is used regardless of whether you are configuring a Helix Core Server, Helix Proxy, or Helix Broker.

If you are installing SDP on a machine that is intended to extend the topology of an existing SDP instance, for example by adding a standby replica, edge server, proxy, or broker, then make sure to use the same SDP instance name and case sensitivity as the data set you are extending.

4.1.2. STEP 1: Configure storage.

For a production install, storage must first be configured before this script can be run.



If you are installing a non-production installation, such as for demonstration or training with SDP, you can skip storage configuration. Storage will not be optimally configured if this step is skipped, but this may be appropriate for a purely demonstration install. Use the `-demo` option to `install_sdp.sh` to skip preflight checks verifying optimal storage configuration.

See [Section A.4, “Storage Volumes Layout”](#) for guidance on storage configuration. There are a variety of options and methods for installing and configuring storage. However accomplished, when storage is ready, the following, the directories must exist and must have storage mounted that is NOT on the OS root volume (this makes it much easier to extend those volumes as required):

- `/hxdepots`
- `/hxmetadata`
- `/hxlogs`

These paths are typical, but are configurable. More information is available in the install configuration file generated below ([Section I.1, “Sample configuration file `sdp\_install.cfg`”](#)).

When storage is properly configured, you will have a `/hxdepots` directory and possibly other `/hx*` directories.

4.1.3. STEP 2: Download the `install_sdp.sh` script.

Become the `root` user, and install the `install_sdp.sh` script in a directory under the root user’s home directory with these commands:

```
mkdir /root/sdp_install
cd /root/sdp_install
curl -L -O
https://workshop.perforce.com/download/guest/perforce_software/sdp/main/Server/Unix/se
tup/install_sdp.sh
chmod +x install_sdp.sh
```

Get help from the script with the `-h` (short usage synopsis) and `-man` (full manual) options:

```
./install_sdp.sh -h
./install_sdp.sh -man
```

4.1.4. STEP 3: Generate install configuration file.

This creates a sample configuration file for modification:

```
./install_sdp.sh -C > sdp_install.cfg
```

4.1.5. STEP 4: Modify install configuration file.

Edit the generated `sdp_install.cfg` using your preferred text editor, changing the values as desired. This file contains various settings with documentation for each setting.

```
vi sdp_install.cfg
```

Once settings are decided, save the file. The file contains lots of comments to explain required configuration choices.

4.1.6. STEP 5: Install SDP (Dry Run).

Call this script and reference the configuration file, as a dry run/preview:

```
./install_sdp.sh -c sdp_install.cfg -init
```

Review the generated log of the preview, and address any reported issues.

4.1.7. STEP 6: Install SDP (Live Run).

```
./install_sdp.sh -c sdp_install.cfg -init -y
```

This will install SDP per the command line and settings in the install configuration file. It performs actions such as:

- Create directory structure and links with appropriate ownership
- Install binaries
- Install crontab and systemd service files
- Initialize the repository as required

4.1.8. STEP 7: Install a license file.

Only certain Helix Core P4D Servers require license files. Helix Proxy and Helix Broker services never require a license file.

For Helix Core P4D Servers, only the commit server and replica servers intended as possible failover targets from the commit server require a license. Other types of replica, e.g. edge servers and standby servers that target edge or forwarding replica servers do not need license files.

For Helix Core P4D Servers that need a license, get the license from Perforce Sales (RevOps), and install Perforce Helix Core license file for the p4d server instance into the P4ROOT directory for your instance, e.g. `/p4/N/root/license` (where N is the instance name). Then restart the p4d_N service.



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 for each instance.



Rename the license file to simply the name `license` in the P4ROOT folder.

4.1.9. Start Your Helix Core Server

You are now 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, “Use of SSL”](#).

4.2. Using makedirs.sh

If you use **install_sdp.sh**, you do not need to call **makedirs.sh** yourself (though the **makedirs.sh** script is called by the **install_sdp.sh** during its processing).

Use the **makedirs.sh** script directly only when adding a new SDP instance, i.e. a new data set, on a server machine where SDP has already been installed.

More details appear in the [Appendix J, *More Detail on makedirs.sh*](#).

4.2.1. Use of SSL

If your configuration file indicated that SSL was to be used during installation, your P4PORT value will have a prefix of **ssl:**.

During installation, the install script initially uses self-signed certificates. If you prefer to use "full chain" certificates acquired from a public global Certificate Authority, referred to as "CA certs", you will need to replace the self-signed certificates in **/p4/ssl** after the SDP install.



Changing SSL certificates is disruptive and should be done in a scheduled maintenance window. Each server machine has certs in **/p4/ssl**, so certificate updates should be coordinated across the server fleet.

4.2.1.1. Changing SSL Certificates

If you choose to update the self-signed certificates, do like this example (for instance 1):

As user **perforce** (or the defined OSUSER) on each p4d, p4proxy, and p4p server machine in your fleet, run the following commands. Note that the first command uses an **rsync** to create a backup of your current certificates before proceeding.

```
sudo systemctl stop p4d_1 ## Stop p4d_1, p4broker_1 and/or p4p_1 as applicable.
source /p4/common/bin/p4_vars 1
rsync -a /p4/ssl/ /p4/ssl.backup.$(date +%Y-%m-%d-%H%M%S')
cd /p4/ssl
rm -f certificate.txt privatekey.txt
cp -f /p4/sdp/Server/Unix/p4/ssl/config.txt .
p4d -Gc
sudo systemctl start p4d_1 ## Start p4d_1, p4broker_1 and/or p4p_1 as applicable.
```

In order to validate that SSL is working correctly:

```
p4 set
```



If you are using full-chain CA certs, the trust is still necessary on the p4d server machine, which is referenced by a P4PORT value that does not include the DNS name used in the full chain certificate. However, with full-chain CA certs, end users can skip the trust.

Update the P4TRUST values:

```
p4 trust -y # Trusts the default port on localhost
p4 -p ssl:p4d.myco.com:1666 trust -y # Assuming correct port
p4 -p ssl:`hostname`:1666 trust -y # Trusts the IP address of current host
p4 -p $P4MASTERPORT trust -y
```

If this server must connect to other servers in the fleet, do **p4 trust** as needed to those server port values as well. For example, a p4d server should trust the **ExternalAddress** of all edge servers, every standby and edge server must trust its **P4TARGET** server.

Check the stored P4TRUST values:

```
p4 trust -l
```

You need to have an entry for the above for both loopback (**127.0.0.1** and the IP address of current machine)

Check you are not prompted for trust:

```
p4 login
p4 info
```

4.2.1.2. Configuration script makedirs.cfg

The **makedirs.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 **makedirs.instance.cfg** for this script before running it, and adjust the values of the variables as required. The important parameters are:

Parameter	Description
DB1	Local path for /hxmetadata1 (can be same as DB2)
DB2	Local path for /hxmetadata2 (can be same as DB1)
DD	Local path for /hxdepots
CD	Local path for /hxcheckpoints , usually same as DD.

Parameter	Description
LG	Local path for <code>/hxlogs</code> volume
SHAREDATA	TRUE or FALSE - whether sharing the <code>/hxdepots</code> volume with a replica - normally this is FALSE
ADMINUSER	P4USER value of a Perforce super user that operates SDP scripts, typically <code>performe</code> .
OSUSER	Operating system user that will run the Perforce instance, typically <code>performe</code> .
OSGROUP	Operating system group that OSUSER belongs to, typically <code>performe</code> .
CASE_SENSITIVE	Indicates if p4d server instance has special case sensitivity settings
SSL_PREFIX	Set if SSL is required so either "ssl:" or blank for no SSL
P4ADMINPASS	Password to use for Perforce superuser account - can be edited later in <code>/p4/common/config/.p4password.p4_1.admin</code>
P4MASTERHOST	Fully qualified DNS name of the Perforce commit server machine for this instance.

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, 8 or 9, 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 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 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 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 run 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 polycoreutils-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 as **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 **p4login** script, giving it the parameter **1** for the SDP Instance name (which calls the **p4 login** command using the `/p4/common/config/.p4passwd.p4_1.admin` file). That looks like this:

```
p4login 1
```

2. For new server instances, run this script, which sets several recommended configurables:

```
/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 suit 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
/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 grant 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/setup/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="$(jq -r '.compute.vmId' /tmp/azure_metadata)"
export REGION="$(jq -r '.compute.location' /tmp/azure_metadata)"
export AZURE_SUBSCRIPTION_ID="$(jq -r '.compute.subscriptionId' /tmp/azure_metadata)"
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-1FIRH0KEXTU"
```

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 P4 Code Review Triggers

For modern versions of P4 Code Review, both *extensions* and *triggers* are provided. Using *extensions* are recommended. If using Extensions, consult the documentation for P4 Code Review and ignore this section.

Machines that need triggers installed include the commit server (**NOT** the Swarm machine), all edge servers, and any standby servers of either the commit or any edge servers.

Get each server to have triggers installed setup to connect to the Perforce Package Repository (if not already done). See: <https://www.perforce.com/perforce-packages>

Install the `helix-swarm-triggers` OS package. As the `root` user:

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

Then (for SDP environments for ease):

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

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

```
vi /opt/performce/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 `/opt/performce/etc/swarm-triggers.conf` config file using the SDP `swarm_triggers_test.sh` script.

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

Do what is needed to get that script to display happy output. Error messages are helpful in indicating the problem. It may require iteration of the conf file, trigger install, etc.

Then install triggers on the server.

```
cd /p4/1/tmp
p4 triggers -o | sed '$d' > /tmp/triggers.old.p4s
cp /tmp/triggers.old.p4s /tmp/triggers.new.p4s

# Append Swarm triggers
/opt/performce/swarm-triggers/bin/swarm-trigger.pl -o >> /tmp/triggers.new.p4s

# Review the file. Adjust order of lines if desired.
vi /tmp/triggers.new.p4s

# Review the diffs.
diff /tmp/triggers.old.p4s /tmp/triggers.new.p4s
```

```
p4 triggers -i < /tmp/triggers.new.p4s
p4 triggers -o # Make sure the new P4 Code Review triggers are there along with any
pre-existing triggers.
```

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, Recovery, and Replication

P4 Server instances maintain *metadata* and *versioned files*. The metadata contains information about all the versioned files (often referred to as "archive files") in the depots. Metadata resides in database (*db.** files) in the server instance's P4ROOT 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. Backup Mechanisms

Depending on how the SDP was installed, the SDP has two structures, the SDP Classic structure and SDP OS Package structure.

5.1.1. SDP Backups in SDP OS Package Structure

If `/p4/common` is a symlink to somewhere under `/opt/perforce`, then you are using the SDP OS Package Structure. This structure was introduced as an option in SDP 2024.2 with the introduction of the `install_sdp.sh` script. This is intended to prepare for planned future changes to provide an option to install and upgrade SDP with standard OS package mechanisms.

With this mechanism, the core of the SDP lives in the `/opt/perforce` structure, which is typically on the OS root volume. When an installation is done with `install_sdp.sh`, a new systemd service name `opt_perforce_sdp_backup.service` is installed, and is called daily via the systemd `opt_perforce_sdp_backup.timer`.

This service backs up everything needed to restore the SDP to fully functional status, including: * All system `p4*.service` and `p4*.timer` files. * The limited sudoers files, e.g. `/etc/sudoers.d/perforce`. * Some or all of the OSUSER home directory; see the `HomeDirBackupMode` setting defined in the `sdp_install.cfg` file; see: [Appendix I, More Detail on install_sdp.sh](#). * All files and folders, including `/p4` and all `/p4/N` folders and files directly in those folders, such as `.p4tickets` and `.p4trust` files.

If SELinux is enabled in enforcing mode, the recovery may have additional steps involving `semanage` and `restorecon` normally performed only during initial installation. See: [Section 4.2.2.2, "Enabling systemd under SELinux"](#).

Backups are stored in an appropriate backup directory on the HxDepots volume, something like:

```
/hxdepots/backup/opt_perforce_helix-sdp.<ShortHostname>
```

5.1.2. SDP Recovery in SDP OS Package Structure

As the `opt_perforce_sdp_backup.service` runs each night, a recovery script is generated in the

backup directory named `recover_opt_perforce_sdp.sh`. This script is intended to be useful both in recovery situations, or situations in which Infrastructure as Code (IaC) deployments are done using existing data sets. In either situation, the starting state is that a machine is prepared with appropriate SDP storage mounted as it had been at the time of backup.

```
cd /hxdepots/backup/opt_perforce_helix-sdp.<ShortHostname>
./recover_opt_perforce_sdp.sh
```

For more info, see: [Section 8.7.13, “opt_perforce_sdp_backup.sh”](#).

5.1.3. SDP Backups in SDP Classic Structure

If the `/opt/perforce/helix-sdp` directory does not exist, then you are using the SDP Classic Structure. In this structure, the `/p4/common` is a symlink to somewhere on the HxDepots volume. In this structure, the core of the SDP itself lives on the HxDepots volume, and thus should be backed up along with your checkpoints and versioned files, either due to backups of that HxDepots volume specifically or whole machine backup methods.

Typically backup solutions backup the entire machine, including critical data on the data volumes as well as those SDP files that live locally on the machine, such as the `/p4` directory itself and tickets for the super user. In the event or recovery with only the critical data volumes, a full recovery of P4 Data is possible. However, the SDP structure (e.g. `/p4`) would need to be created using `mkdirs.sh`, and some files such as ticket files would need to be recreated.

5.2. Planning for HA and DR

While it is necessary to be *prepared* to do a traditional recovery from backup, a key goal of replication strategies is to avoid ever having to do that. We strongly advise that replication strategies be designed to *augment*, but not entirely *replace* traditional backup strategies. Real-time replication is the best defense against certain kinds of failures, but traditional backups are required for other kinds of failures. **The best practice is to maintain both real-time replication and traditional point-in-time backup and recovery options.**

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 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. 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.

5.2.2. 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”](#).

5.2.3. 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.

5.3. 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 5.2.2, “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. For the detailed usage statement, go to [Section 8.4.7, “mkrep.sh”](#)



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`.



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 server host naming convention: [P4 Server Host Naming Convention](#).

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. Additional Replication Setup

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

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 ed25519
```

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 Technical Support for guidance on depot recovery. Recovering the hxddepots volume using your normal recovery procedures may help.
Lost or Corrupt	Lost or Corrupt	Recover metadata as described below. Call Perforce Technical Support for guidance on depot recovery. Recovering the hxddepots volume using your normal recovery procedures may help.

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 8.7.32](#), “`sync_replica.sh`”.

See the steps in the script [Section 8.7.32](#), “`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, use the `mkdirs.sh` script to create missing elements of the SDP structure. This can be used in a recovery scenario where some folders exist and others do not. This script will safely create only the folders needed. However, it is necessary to first ensure storage is properly configured before calling `mkdirs.sh`. In particular, the `/hxdepots`, `/hxmetadata`, and `/hxlogs` volumes must be mounted properly.

Following is a sample procedure using `1` as the SDP instance name:

1. Recover the `/hxdepots` volume from your backup tape.
2. As root, do: `cd /hxdepots/sdp/Server/Unix/setup; ./mkdirs.sh 1`
3. Switch to the Perforce OS account.
4. Find the last available checkpoint, under `/p4/1/checkpoints/`
5. Reinstall the Perforce server license to the server P4ROOT directory.
6. Recover the latest checkpoint by running:

```
source /p4/common/bin/p4_vars N
nohup load_checkpoint.sh -latest < /dev/null > $LOGS/load.log 2>&1 &
```

7. Verify that the server instance is running.
8. Reinstall the server crontab or scheduled tasks.
9. Perform any other initial server machine configuration.
10. 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 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”](#).

6.3.1.1. Sample SDP Upgrade in Classic Structure

First, confirm that you are on the SDP Classic or SDP OS Package structure. If the `/opt/perforce/helix-sdp` directory does not exist, then you are using the SDP Classic Structure, and the sample instructions below apply. If that directory does exist, your installation was done with the `install_sdp.sh` script; to upgrade it see [Section 6.3.1.2, “Sample SDP Upgrade in OS Package Structure”](#).

For the SDP Classic structure (if `/opt/perforce/helix-sdp` does **not** exist), a sample set of commands to upgrade appears below, to be As the OS user that your p4d service runs as (typically `perforce`):

As `perforce`:

```
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 -O
https://workshop.perforce.com/download/guest/perforce_software/sdp/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_upgrade.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.)

6.3.1.2. Sample SDP Upgrade in OS Package Structure

If `/opt/perforce/helix-sdp` directory exists, you are using the SDP OS Package Structure. A basic set of commands to upgrade is as follows, which must be executed as the **root** user:

As **root**:

```
cd /opt/perforce/helix-sdp/downloads
[[ -e sdp.Unix.tgz ]] && mv -f sdp.Unix.tgz sdp.Unix.$(date +%Y-%m-%d-%H%M%S').tgz
curl -L -O
https://workshop.perforce.com/download/guest/perforce_software/sdp/downloads/sdp.Unix.
tgz
tar -tzf sdp.Unix.tgz 2>&1 | grep -q sdp/Version && echo OK
```

If this does not display an OK message, the tarball is not valid. Investigate and resolve this issue before proceeding.

As **root**:

```
cd /opt/perforce/helix-sdp
[[ -d backup ]] || mkdir backup
mv sdp backup/sdp.old.$(date +%Y-%m-%d-%H%M%S')
tar -xzf downloads/sdp.Unix.tgz
chown -R root:root sdp
```

```
cd /opt/perforce/helix-sdp/sdp/Server/Unix/p4/common/sdp_upgrade
./sdp_upgrade.sh
```

Ensure the output contains a Success message near the end before proceeding with the live upgrade:

```
./sdp_upgrade.sh -y
```

6.3.2. SDP Upgrades in the Future

When SDP OS Packages eventually become an option for install and upgrade, they will not be required. The current mechanism of installing with a script will be maintained indefinitely. This is needed to support Linux distros for which packages are not available, and for situations like air gap installations where OS package mechanisms may not be preferred or available. However, for those

that have the option, we expect package update mechanisms to be preferred.

Changing the SDP in place from the SDP Classic Structure to the SDP OS Package Structure will be a future option. As of SDP 2025.1 Patch 1, this transition is not automated nor documented.

6.3.3. 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**.

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/p4d_profile
```

2. Then create a new **tuned.conf** file for **p4d_profile**, and insert the new tuning info:

```
cat <<EOF > /etc/tuned/p4d_profile/tuned.conf
[main]
summary = Profile for Perforce server performance optimization
include = virtual-guest

[vm]
transparent_hugepages = never
```

```
EOF
```

3. Make the script executable

```
chmod +x /etc/tuned/p4d_profile/tuned.conf
```

4. Enable `p4d_profile` using the `tuned-adm` command.

```
tuned-adm profile p4d_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]
```

6. To validate a profile (after a reboot):

```
tuned-adm verify  
tail -40 /var/log/tuned/tuned.log
```

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 the 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 performce:performce /hxserverlocks
```

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

```
HxServerLocks /hxserverlocks tmpfs
uid=performce,gid=performce,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 (`performce` or `$OSUSER`):

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

As user `performce` (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 and 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 situation.

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 HxServerLocks 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 every 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://portal.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. In crontabs, the `p4master_run` has effectively been replaced by the more granular `run_if_master.sh`, `run_if_replica.sh`, and `run_if_edge.sh`. The `run_if_*.sh` and `p4master_run` scripts load shell 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 name to them as a parameter (and even that is optional if the `SDP_INSTANCE` variable has been set, which is commonly done on initial login).

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/Linux 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
<code>/p4/common/bin/p4master_run 1 /p4/1/bin/p4_1 admin stop</code>	Run <code>p4 admin stop</code> on instance 1
<code>/p4/common/bin/live_checkpoint.sh 1</code>	Take a checkpoint of the live database on instance 1

Example	Remarks
<code>/p4/common/bin/p4login 1</code>	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 checkpoint 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 v1.7.5:

get_helix_binaries.sh [-r <HelixMajorVersion>] [-b <Binary1>,<Binary2>,...] [-api] [-sbd <StageBinDir>] [-n] [-d|-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

    In addition, P4API, the C++ client API, can be downloaded.

    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 /p4/sdp/helix_binaries directory in order for the upgrade.sh script to find the downloaded binaries.
```

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 uses the downloaded binaries to upgrade a single SDP instance (of which there might be several on a machine).

The `helix_binaries` directory must 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 in the PATH outside `/p4/common/bin`.

When a newer major version of Helix binaries is needed, this script should not be modified directly. Instead, get the latest version of SDP first, which will include 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.

PLATFORM DETECTION

The `'uname'` command is used to determine the architecture for the current machine on which this script is run.

This script and supporting `P4*.json` release list files know what platforms for which builds are available for each Helix Core binary (`p4`, `p4d`, `p4broker`, `p4p`). If the `'jq'` utility is available, this script uses the `P4*.json` files to verify that a build is available for the current platform, and in some cases selects an alternate compatible platform if needed. For example, if the detected platform is for OSX 12+ for the `x86_64` architecture, no build is available for binaries such as `p4d` for that platform, so a compatible alternative is used instead, in this case `macosx1015x86_64`.

This script handles only the UNIX/Linux platforms (to include OSX).

RELEASE LIST FILES:

For each binary, there is a corresponding release list file (in json format) that indicates the platforms available for the given binary. These files are:

- `P4.json` (for the `'p4'` binary)
- `P4D.json` (for the `'p4d'` binary)
- `P4Broker.json` (for the `'p4broker'` binary)
- `P4Proxy.json` (for the `'p4p'` binary)

These `P4*.json` release list files are aware of a wide list of supported platforms for a range of Helix Core binaries.

These release list files are packaged with the SDP, and updated for each major release.

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 r25.2

The form of 'rYY.N', e.g. 'r25.1', is the default form of the version, matching what is used in URLs on the Perforce Helix FTP server. For flexibility, similar forms that clearly convent the intended version are also accepted. For example:

'-r 23.1' is implicitly converted to '-r r23.1'.

'-r 2023.1' is implicitly converted to '-r r23.1'.

-b <Binary1>[,<Binary2>,...]

Specify a comma-delimited list of Helix binaries. The default is: p4 p4d p4broker p4p

Alternately, specify '-b none' in conjunction with '-api' to download only APIs and none of the p4* binaries.

-api

Specify '-api' to download P4API, the C++ client API. This will acquire one or more client API tarballs, depending on the current platform. The API files will look something like these examples:

```
* p4api-glibc2.3-openssl1.1.1.tgz
* p4api-glibc2.3-openssl3.tgz
* p4api-glibc2.12-openssl1.1.1.tgz
* p4api-glibc2.12-openssl3.tgz
```

```
* p4api-openssl1.1.1.tgz
* p4api-openssl3.tgz
```

All binaries that match 'p4api*tgz' in the relevant directory on the Perforce FTP server for the current architecture and Helix Core version are downloaded.

Unlike binary downloads, the old versions are not checked, because file names are fixed as they are with binaries.

APIs are not needed for normal operations, and are only downloaded if requested with the '-api' option. They may be useful for developing custom automation such as custom triggers. Be warned, custom triggers are not supported by Perforce Support.

-sbd <StageBinDir>

Specify the staging directory to install downloaded binaries.

By default, this script downloads files into the current directory, which is expected and required to be /p4/sdp/helix_binaries. Documented workflows for using this script involve first cd'ing to that directory. Using this option disables the expected directory check and allows binaries to be installed in any directory, which may be useful if this script is used

as a standalone script outside the SDP (e.g. for setting up test environments or enabling Helix native DVCS features by installing binaries into /usr/local/bin on a non-SDP machine).

This option also sets the location in which this script searches for the P4*.json release list files.

-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 debugging verbosity.

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

HELP OPTIONS:

-h Display short help message
-man Display this manual page

EXAMPLES:

Example 1 - Typical Usage with no arguments:

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

This acquires the latest patch of all 4 binaries for the r25.2 release (aka 2025.2).

This will not download APIs, which are not needed for general operation.

Example 2 - Specify the major version:

```
cd /p4/sdp/helix_binaries
./get_helix_binaries.sh -r r23.2
```

This gets the latest patch of for the 2023.2 release of all 4 binaries.

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

Note: Only the latest patch for any given major release is available from the Perforce FTP server.

Example 3 - Get r24.2 and skip the proxy binary (p4p):

```
cd /p4/sdp/helix_binaries
./get_helix_binaries.sh -r r24.2 -b p4,p4d,p4broker
```

Example 4 - Download r25.1 binaries in a non-default directory.

```
cd /any/directory/you/want
```

```
./get_helix_binaries.sh -r r25.1 -sbd .
```

or:

```
./get_helix_binaries.sh -r r25.1 -sbd /any/directory/you/want
```

Example 5 - Download C++ client API only:

```
./get_helix_binaries.sh -r r24.1 -b none -api
```

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 Helix 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, /p4/sdp/helix_binaries by default.

EXIT CODES:

An exit code of 0 indicates no errors were encountered. A 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.3, “P4D versions and links”](#)

Usage

USAGE for upgrade.sh v4.14.1:

```
upgrade.sh <instance> [-p|-I] [-M] [-Od] [-Ov] [-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

The preferred process for using this script is to start with the services to be upgraded (p4d, p4broker, and/or p4p) up and running at the start of processing. The p4d service must be online if it is to be upgraded.

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 current machine.

For example, a typical commit server 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 its 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 - Perform 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 on 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.

COMMIT SERVER 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. It also prepares the offline_db in case a rollback is needed.

When the commit server is upgraded, two rotations of the commit server's journal occur during processing for major upgrades, and a single journal rotation is done for patch upgrades. 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. This journal is immediately replayed to the offline_db.

Later, after p4d has started performs its journaled upgrade processing, a second journal rotation occurs in Phase 5 if a major upgrade was done. This second journal rotation captures all upgrade-related processing in a separately numbered journal. This second journal is not applied to the offline_db by this script. Instead, the replay of the second journal to the offline_db will occur the next time a call is made to the daily_checkpoint.sh or rotate_journal.sh, e.g. via routine crontab. For a p4d patch upgrade, there will not be any upgrade processing.

In the very unlikely event that a rollback were to ever be needed, the offline_db is left in a state that it could be used for a fast rollback on the commit server.

MULTI-SERVER OUTER-TO-INNER UPGRADES

Before starting an outer-to-inner upgrade involving multiple p4d servers, (standby, edge, and other p4d replica servers), a manual journal rotation should be done on the commit server before starting to call upgrade.sh on each of the p4d servers in outer-to-inner order. Take note of the journal counter used for this pre-start journal rotation. It can be useful in event of a rollback. That journal may need to be replayed to the offline_db on all servers other than the commit in a rollback scenario.

In preparation in days or weeks before an upgrade, every p4d server in the topology should be checked to ensure its offline_db is healthy and current.

ROLLBACK

In the very unlikely event that a rollback is needed, bear in mind the following:

- * There is no standard procedure for rolling back, because a procedure would need to take into account the reason a decision was made to rollback. Presumably the decision would be driven by some kind of failure. A large factor in determining whether rollback is practical is the point in the process at which a rollback is needed. In some situations, a 'Fix and Roll Forward' approach may be more pragmatic than a rollback, and should always be considered.
- * This script and supporting documentation will help prepare your data for as smooth a rollback as possible should it ever become necessary.
- * To best prepare for a rollback, it is essential to manage user lockout as part of your overall maintenance procedure. Then let users back in after you have confirmed you are moving forward. User lockout is outside the scope of this script, but can be managed using several possible methods such as:
 - Crafting a special Protections table to be used during maintenance,
 - Using "Down for Maintenance" brokers,
 - Using network and/or on-host firewall rules,
 - Using temporary ports for maintenance.
- * If Phase 2 (update of symlinks and binaries) completed and must be undone, than can be achieved by putting the pre-upgrade binaries in

place in the directory /p4/sdp/helix_binaries, named simply p4, p4d, p4broker, and p4p. Then run a command like this example for Instance 1:

```
upgrade.sh 1 -M -I -y
```

This will change symlinks back to reference the older versions. The new binaries will still exist in /p4/common/bin, but will no longer be referenced for Instance 1.

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 one 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 gap" 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.

`-Ov`

Specify `'-Ov'` to override the check that prevents upgrading P4 if the P4 version is newer than the SDP version.

By default, if the P4 version being upgraded to is newer than the current SDP version, the upgrade is aborted with an error message. This flag allows the upgrade to proceed despite the version mismatch.

CAUTION: This is an advanced option. Proceeding with a P4 version newer than the SDP version may result in compatibility issues, as some P4 features may require a newer SDP to function properly, and SDP is not tested with P4 versions newer than the SDP version. It is best to upgrade SDP before P4.

Use this option only if you have reviewed the compatibility implications and understand the risks.

`-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.

If '-M' is used, this script does not check that services to be upgraded are online at the start of processing, nor does it attempt to start to stop services. If '-M' is used, the services should be stopped manually before calling this script, and then started manually after.

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.<timestamp>.log
```

NOTE: This script is self-logging. That is, output displayed on the screen is simultaneously captured in the log file. Redirection operators like '> log' and '2>&1' are not required, nor is '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-specific 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-upgrade 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` v2.3.9:

```
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, which can be in one of two locations. If the `/opt/perforce/helix-sdp/sdp` directory exists, start the upgrade process like this (as the 'root' user):

```
cd /opt/perforce/helix-sdp/sdp/Server/Unix/p4/common/sdp_upgrade
./sdp_upgrade.sh
```

Otherwise, start the upgrade like this, as the operating system user account under which the p4d service runs as (e.g. 'perforce' or 'p4admin' but never as 'root'):

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

Running `sdp_upgrade.sh` without `'-y'` will do a dry run. Review the output and check for a SUCCESS indication at the end. If you get the success message, proceed with:

```
./sdp_upgrade.sh -y
```

DESCRIPTION:

This script upgrades Perforce Helix Server Deployment Package (SDP) from SDP 2020.1 to the version included in the latest SDP version, SDP 2025.1.

== 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://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.

== Diretory Structure Changes for /p4/sdp and /p4/common ==

There is a structure change with SDP affecting where the /p4/sdp and /p4/common symlinks are targeted. This change is part of a phased rollout of a new structure to be used by a future helix-sdp OS package. Nothing in the structural changes affects behaviors of routine SDP daily scripts. The changes affect how SDP upgrades work, and on what volumes files like Helix Core server binaries exist on. The gist of the change is that /hxdepots/p4/common and /hxdepots/sdp folders (which are on NFS if /hxdepots is NFS-mounted) are changed from being actively used folders to become backup directories,

The active directories are moved to local storage on the machine in the new /opt/perforce/helix-sdp structure.

Prior to SDP 2024.2, this was the typical symlink structure is:

- * /hxdepots/sdp, symlinked as /p4/sdp
- * /hxdepots/p4/common, symlinked as /p4/common
- * /hxdepots/downloads, no symlink

If /hxdepots is NFS mounted, then the active folders are on NFS.

Starting with 2024.2, a new structure is available, and will be used if the install_sdp.sh script was used for the initial SDP install. In that structure, we have the following:

- * /opt/perforce/helix-sdp/p4/sdp, symlinked as /p4/sdp
- * /opt/perforce/helix-sdp/p4/common, symlinked as /p4/common
- * /opt/perforce/helix-sdp/downloads, no symlink
- * /opt/perforce/helix-sdp/sdp, immutable root-owned structure, updated only by SDP upgrades (i.e. this script).

During upgrades, the legacy structure is changed in ways that are safe even if /hxdepots is NFS-shared with machines not being upgraded. For safety, some structures are abandoned and untouched.

- * A new /hxdepots/sdp/backup folder is created, and contain /hxdepots/sdp/backup/opt/perforce/helix-sdp
- * Dirs other than backup under /helix/sdp are unused/untouched
- * /hxdepots/p4/common is unused/untouched
- * /hxdepots/downloads is moved to /hxdepots/sdp/backup/downloads,

and the /hxdepots/sdp becomes a symlink to /opt/perforce/helix-sdp/downloads.

== 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 - OS Package Structure ==

If the /opt/perforce/helix-sdp structure exists on your machine, then upgrade using the procedure in this section. Otherwise see the section below "Acquiring the SDP Package - Classic Structure".

Become the root user first:

```
sudo su -
cd /opt/perforce/helix-sdp/downloads
[[ -e sdp.Unix.tgz ]] && mv -f sdp.Unix.tgz sdp.Unix.$(date +%Y-%m-%d-%H%M%S').tgz
curl -L -O
https://workshop.perforce.com/download/guest/perforce\_software/sdp/downloads/sdp.Unix.tgz
tar -tzf sdp.Unix.tgz 2>&1 | grep -q sdp/Version && echo OK
```

If this does not display an OK message, the tarball is not valid. Investigate and resolve this issue before proceeding. As root:

```
cd /opt/perforce/helix-sdp
[[ -d backup ]] || mkdir backup
mv sdp backup/sdp.old.$(date +%Y-%m-%d-%H%M%S')
tar -xzf downloads/sdp.Unix.tgz

cd /opt/perforce/helix-sdp/sdp/Server/Unix/p4/common/sdp_upgrade
./sdp_upgrade.sh -man
```

== Acquiring the SDP Package - Classic Structure ==

If the /opt/perforce/helix-sdp structure exists on your machine, then upgrade using the procedure above in the OS Package Structure section.

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 performce:

```
cd /hxdepots
[[ -d downloads ]] || mkdir downloads
cd downloads
[[ -d new ]] && mv new old.$(date +%Y%m%d-%H%M')
curl -L -O
https://workshop.perforce.com/download/guest/perforce_software/sdp/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 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 <HxDpots>/p4/common structure is backed up to:

```
<HxDpots>/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_MON_LEVEL
- 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.
- KEEPPLOGS (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:

`/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.

-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.

`-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`

EDITME - Add new structure dirs `/opt/perforce/helix-sdp`

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 \(for UNIX/Linux\)](#).

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 seven days a week using crontab. It operates on commit and edge servers. When run on the commit server, this script rotates the active journal to a numbered journal file, and then maintains the commit server's `offline_db` using the numbered journal file immediately after it is rotated. If there are any other outstanding numbered journal files to be replayed into the `offline_db` (perhaps due to manual journal rotations), those are replayed to the `offline_db` as well.

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 commit server, except that the journal rotation is skipped (as that can be done only on the commit server).

As a recap, this script does the following daily:

- Requests p4d to do a journal rotation (if on the commit server).
- Deletes the `offline_db_usable.txt` semaphore file to indicate the `offline_db` is being operated on and is not usable.
- Replays whatever numbered journal files are needed to make the `offline_db` current, up to but not including the live P4JOURNAL.
- Creates a new checkpoint from the `offline_db`.
- Deletes the `db.*` files in `offline_db`.
- Recreates the `offline_db` database set from the newly created checkpoint.
- Writes a new `offline_db_usable.txt` semaphore file to indicate the `offline_db` is healthy and ready to use.
- Rotates logs files and handles checkpoint, journal, and log retention according to settings `$KEEPLOGS`, `$KEEPJNLS`, and `$KEEPCKPS`.

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 8.4.12, “refresh_P4ROOT_from_offline_db.sh”](#)

Usage

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

8.4.5. `keep_offline_db_current.sh`

The `/p4/common/bin/keep_offline_db_current.sh` script is for use only on a standby replica. It will not run on any other type of replica.

This script ensures the `offline_db` has the most current journals replayed.

It is intended for use on standby replicas as an alternative to `sync_replica.sh` or `replica_cleanup.sh`.

It is ideal for use in an environment where the checkpoints folder of the P4TARGET server is shared (e.g. via NFS) with this server. It is also suitable for use in Windows to Linux migrations for operation on the Linux standby of the Windows commit server, where the `rsync` commands in the `sync_replica.sh` script are unable to pull the checkpoints folder of the (Windows) P4TARGET server to the Linux standby. This `keep_offline_db_current.sh` should be run instead of `sync_replica.sh` until after the failover to Linux is done, at which point `daily_checkpoint.sh` should be run instead.

This script does NOT do full checkpoint operations, and requires that the `offline_db` be in a good state before starting. The health of the `offline_db` is verified with a call to `verify_sdp.sh`.

This uses `checkpoint.log` as its primary log. It is only intended for use on a machine where other scripts that update `checkpoint.log` don't run, e.g. `daily_checkpoint.sh`, `live_checkpoint.sh`, or `rotate_journal.sh`.

Usage

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

8.4.6. live_checkpoint.sh

The `/p4/common/bin/live_checkpoint.sh` script is used to initialize the SDP `offline_db`. It is typically run once during initial installation, before any other scripts that rely on the `offline_db` can be used, such as `daily_checkpoint.sh`, `edge_dump.sh`, and `rotate_journal.sh`.

This script can also be used in some cases to repair the `offline_db` if it has somehow 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 operation. This can take hours for a large installation. 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 in your environment.

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 (seconds), but over time it can grow to minutes and eventually hours. The idea is that `live_checkpoint.sh` should be used only very sparingly. It is not scheduled as a routine operation.



If you have a large set of database files and checkpoints take many hours, then `p4d >= 2022.2` offers parallel checkpointing options which can reduce the time substantially. See [Section 3.3, "Parallel Checkpoints"](#).

The `live_checkpoint.sh` script can operate on a P4D commit or edge servers, with slightly different behaviors on each.

8.4.6.1. live_checkpoint.sh on a commit server

When operating on a commit server, `live_checkpoint.sh` does the following:

- Creates a checkpoint from the live database `db.*` files in the P4ROOT. The checkpoint operation involves a journal rotation as part of the checkpoint operation done by `p4d`. The active P4JOURNAL file becomes numbered and a fresh journal is started.
- Deletes the `offline_db_usable.txt` semaphore file to indicate the `offline_db` is being operated on and is not usable.
- Deletes the `db.*` files in `offline_db`.
- Recreates fresh database files in the `offline_db` from the new checkpoint. This creates fresh `db.*` files that have internally balanced binary tree structures and are typically smaller than the original files, yet contain identical data.
- Writes a new `offline_db_usable.txt` semaphore file to indicate the `offline_db` is healthy and ready to use.
- Rotates logs files and handles checkpoint, journal, and log retention according to settings `$KEEPLOGS`, `$KEEPJNLS`, and `$KEEPCKPS`.

8.4.6.2. `live_checkpoint.sh` on an edge server

When operating on an edge server, `live_checkpoint.sh` does the following:

- Requests that the local edge server create a checkpoint when it detects a journal rotation from its target server.
- Requests that the upstream P4TARGET server do a journal rotation. This is forwarded ultimately to the commit server, which will initiate a journal rotation on the commit server.
- Waits for `p4d` to create a checkpoint from the edge's live database `db.*` files in the P4ROOT.
- Deletes the `offline_db_usable.txt` semaphore file to indicate the `offline_db` is being operated on and is not usable.
- Deletes the `db.*` files in `offline_db`.
- Recreates fresh database files in the `offline_db` from the new edge checkpoint. This creates fresh `db.*` files that have internally balanced binary tree structures and are typically smaller than the original files, yet contain identical data.
- Writes a new `offline_db_usable.txt` semaphore file to indicate the `offline_db` is healthy and ready to use.
- Rotates logs files and handles checkpoint, journal, and log retention according to settings `$KEEPLOGS`, `$KEEPJNLS`, and `$KEEPCKPS`.

Usage

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

8.4.7. `mkrep.sh`

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

Usage

USAGE for mkrep.sh v3.8.1:

```
mkrep.sh -t <Type> -s <Site_Tag> -r <Replica_Host> [-f <From_ServerID>] [-os] [-p] [-p4config <PathToFile>] [-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 new 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 user(s) to the group 'ServiceUsers'.
- * Verifies the group ServiceUsers is granted super access in the

protections table (and with '-p', also 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 script 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.
- * rom: Read-Only standby replica, Metadata only.
- * 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:' and 'Address:' fields 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 existing server spec. By default, this script will abort if 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 always 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 * //...
```

This option may not be suitable for use on servers that have custom automation managing the Protections table.

-p4config <PathToFile>

Use the '-p4config' option use this SDP mkrep.sh script to create a replica spec on an arbitrary p4d server. That arbitrary server can be any p4d version, operating on any platform, and need not be managed with SDP.

To use this option, first create a P4CONFIG file that defines settings needed to access the other server. As a convention, identify a short tag name for the other server to use in the P4CONFIG file. In the example below, we use 'mot' for "my other server". Create a P4CONFIG file text named /p4/common/site/config/.p4config.mot that contains these settings:

```
P4PORT=ssl:my_other_server:1666
P4USER=p4admin
P4TICKETS=/p4/common/site/config/.p4tickets.mot
P4TRUST=/p4/common/site/config/.p4trust.mot
```

The P4TRUST setting is only needed if the port is SSL-enabled. If it is enabled, next trust the port:

```
p4 -E P4CONFIG=/p4/common/site/config/.p4config.mot trust -y
```

Next, generate a ticket on that connection:

```
p4 -E P4CONFIG=/p4/common/site/config/.p4config.mot login -a
```

Provide the password if prompted.

Finally, call mkrep.sh and specify the config file. When using this option, using '-L' to specify a non-default log file name is useful to keep logs from external servers cleanly separated.

```
mkrep.sh -p4config /p4/common/site/config/.p4config.mot -L /mkrep.mot.log
```

This will run the mkrep against the server specify in that P4CONFIG file.

-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 be 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 specified 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. Using redirection operators like `> log` or `>&1` are not necessary, nor is using '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 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-specific 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
```

EXAMPLE 5 - Set up a High Availability (HA) Replica of the master *from* the replica server via -p4config.

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 -p4config
/p4/common/site/config/.p4config.mot -L /mkrep.mot.log
```

8.4.8. 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 p4verify.sh v5.26.1:

```
p4verify.sh [<instance>] [-N] [-nu] [-nr] [-ns] [-nS] [-a] [-nt] [-nz] [-no_z] [-o
BAD|MISSING] [-p4config <PathToFile>] [-chunks <ChunkSize>|-paths <paths_file>] [-w
<Wait>] [-q <MaxActivePullQueueSize>] [-Q MaxTotalPullQueueSize] [-recent|-recent=N|-
recent=head|-since <date>|-last <string>] [-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.

The singular Extensions depot is also verified, if present.

The singular Traits depot is also verified, if present.

Archive depots are not verified by default, but can be with the '-a' option.

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.

This option also causes Extension and Traits depots (if present) not to be verified.

- 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 '-nz' 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.
- no_z Specify '-no_z' to avoid passing the '-z' option 'p4 verify' commands. Typically, verifies are done with '-qz'; with this option, '-q' is used instead.

See 'p4 help verify' for more information.

-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.

`-p4config <PathToFile>`

Use the '`-p4config`' option use this SDP `p4verify.sh` script to verify an arbitrary p4d server. That arbitrary server can be any p4d version, operating on any platform, and need not be managed with SDP.

To use this option, first create a `P4CONFIG` file that defines settings needed to access the other server. As a convention, identify a short tag name for the other server to use in the `P4CONFIG` file. In the example below, we use '`mot`' for "my other server". Create a `P4CONFIG` file text named `/p4/common/site/config/.p4config.mot` that contains these settings:

```
P4PORT=ssl:my_other_server:1666
P4USER=p4admin
P4TICKETS=/p4/common/site/config/.p4tickets.mot
P4TRUST=/p4/common/site/config/.p4trust.mot
```

The `P4TRUST` setting is only needed if the port is SSL-enabled. If it is enabled, next trust the port:

```
p4 -E P4CONFIG=/p4/common/site/config/.p4config.mot trust -y
```

Next, generate a ticket on that connection:

```
p4 -E P4CONFIG=/p4/common/site/config/.p4config.mot login -a
```

Provide the password if prompted.

Finally, call `p4verify.sh` and specify the config file. When using this option, using '`-L`' to specify a non-default log file name is useful to keep logs from external servers cleanly separated.

```
p4verify.sh -p4config /p4/common/site/config/.p4config.mot -L /p4verify.mot.log
```

This will run the varify against the server specify in that `P4CONFIG` file.

`-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_verify_chunks.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 significantly slower overall verify operation, though chunking 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 '-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.

The '-paths' option can be combined with '-recent' to verify only recent changelists in the specified paths.

This option disables processing of the Extensions and Traits depots by default, though '-paths' can specify paths in those depots.

Paths specified must be in depots of type 'stream' or 'local', or the singular Extensions or Traits depots.

`-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 procossed (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 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 archive 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 avoiding 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 is mutually exclusive with '-nt'.

The '-q' option is 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 avoiding 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 transferred 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 avoiding 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 is mutually exclusive with '-nt'.

The '-Q' option is mutually exclusive with '-q'.

`-recent[=N]`

`-recent[=head]`

Specify that only recent changelists should be verified.

This can be specified as '-recent', '-recent=N', where N is an integer indicating the number of recent changelists to verify in each depot, or '-recent=head' to specify to verify the head revision of all files in a depot.

If '-recent' is used without the optional '=N' or '=head' syntax, 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/site/config/p4_N.vars.local` to change the default for an instance, or to `/p4/common/site/config/p4_vars.local` to change it globally. If `$SDP_RECENT_CHANGES_TO_VERIFY` is unset, the default is 200.

When `-recent` is used, it also applies to shelves; i.e. only the most recent shelves are verified. When applied to shelved changes, '-recent=head' means to verify only the most recent single shelf.

When `-recent` is used, files in the unload depot are not verified.

`-since <date>`

Specify that only changes since the date provided should be verified. the date must be in the form of 'YYYY/MM/DD:HH:MM:SS' (HH,MM, and SS are options)

`-last <string>`

Specify that only changes over the last period described by <string> should be verified.

<string> can take any form understood by date i.e. '1 month', '2 days', '3 years' (n.b. 'ago' will be appended to the string before it's passed to date)

`-dlf <depot_list_file>`

Specify a file containing a list of depots to process in the desired order. By default, all depots in the order reported by reported by 'p4 depots' are processed, which effectively results in depots being

processed in alphabetical order, with the singleton Extensions and Traits depots (if present) being processed after other depots.

This '-dlf' option 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 straightforward, 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.

This option disables processing of the singleton Extensions and Traits depots unless those depots are explicitly included in the depot list file.

-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/cfg/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 -a -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

USAGE TIPS:

On a p4d server machine on which this script runs, the P4USER usually has an unlimited ticket in the P4TICKETS file. If this is not the case, ensure that the ticket duration is sufficient for the verify operation to complete. If the '-p4config' option is used, ensure the defined P4USER references a P4TICKETS file with sufficiently far out expiration to prevent issues with ticket expiration.

Depending on scale of data and system resources, this p4verify.sh script may run for hours or even days. A ticket duration of less than a defined minimum results in an warning being displayed in the log (but does not prevent the script from attempting the verify).

The minimum ticket duration is 31 days 0 hours 0 minutes 0 seconds.

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: Verify Since Date

Verify only changes since a given date:

```
p4verify.sh 1 -since 2025/12/31:23:59:59
```

Example 5: Verify Last Period of Time

Verify Last 2 Weeks:

```
p4verify.sh 1 -last "2 weeks"
```

Example 6: 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 7: 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 8: 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 9: 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 extracted with a command like this example (adjusting the log file name as needed):

```
grep -a ^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.

8.4.9. 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 v4.6.4:

```
p4login [<instance>] [-p <port> | -service] [-automation] [-all] [-L <log>] [-d|-D]
```

or

```
p4login -h|-man|-V
```

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 `$SDP_ALWAYS_LOGIN` variable. If set to 1, this causes `p4login` to always execute a 'p4 login' command to generate a login ticket, even if a 'p4 login -s' test indicates none is needed. By default, the login is skipped if a 'p4 login -s' test indicates a long-term ticket is available that expires 31+days in the future. Add "export `SDP_ALWAYS_LOGIN=1`" to `/p4_N.vars` to change the default for an instance, or to `/p4_vars` to change it globally. If unset, the default is 0.
 4. If the `P4PORT` contains an SSL prefix, the `P4TRUST` relationship is checked, and if necessary, a `p4 trust -f -y` is done to establish trust.
 5. The password file, which can be either in cleartext or encoded form.
- If cleartext, the password file path is: `$SDP_ADMIN_PASSWORD_FILE`
- Otherwise, the password file path is: `${SDP_ADMIN_PASSWORD_FILE}.enc`
- Using encoded passwords can prevent passwords from being stored in cleartext on the server machine.
- Commonly passwords are stored in a password manager or vault.

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` exits immediately with an error message.

-service

Specify `-service` on a replica or edge server to login the super user and the replication service user to the port of the `P4TARGET` server.

Specify `-service` on the commit server to login the super user and the service user of the commit server to the `ExternalAddress` of each "active" edge server (those with an `IsAlive` value of 1 as reported by 'p4 -ztags servers -J' on the commit).

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 using bash 'set -x' mode.

HELP OPTIONS:

-h Display short help message
 -man Display man-style help message
 -V Display version info.

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_vars abc
p4login
```

Log 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.

8.4.10. `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 8.7.14, “p4d_base”](#)).

Usage

```
/p4/<instance>/bin/p4d_<instance>_init [ start | stop | status | restart ]
/p4/1/bin/p4d_1_init start
```

8.4.11. `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”](#) **please note warnings as to how long that might take!**

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 (i.e. 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!



Please note the warnings about how long this process may take at [Section 8.4.6](#), “`live_checkpoint.sh`”

Usage

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

8.4.12. 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
```

8.4.13. run_if_master.sh

The `/p4/common/bin/run_if_master.sh` script is explained in [Section 8.4.16](#), “`run_if_master/edge/replica.sh`”

8.4.14. run_if_edge.sh

The `/p4/common/bin/run_if_edge.sh` script is explained in [Section 8.4.16](#), “`run_if_master/edge/replica.sh`”

8.4.15. run_if_replica.sh

The `/p4/common/bin/run_if_replica.sh` script is explained in [Section 8.4.16](#), “`run_if_master/edge/replica.sh`”

8.4.16. 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)

8.4.17. sdp_health_check.sh

This script is described in the appendix [Appendix H, SDP Health Checks](#).

USAGE for sdp_health_check.sh v1.17.1:

sdp_health_check.sh

or

sdp_health_check.sh -h|-man

DESCRIPTION:

This script does a health check of the SDP. It generates a report log, which can be emailed to support-p4@perforce.com. It identifies SDP instances and reports on general SDP health.

It must be run as the OS user who owns the `/p4/common/bin` directory. This should be the user account which runs the p4d process, and which owns the `/p4/common/bin` directory (often 'perforce' or 'p4admin').

Characteristics of this script:

- * 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.

Assumptions:

- * The SDP has always used `/p4/common/bin/p4_vars` as the shell environment file. This is consistent across all SDP versions.

OPTIONS:

`-D` Set extreme debugging verbosity.

HELP OPTIONS:

`-h` Display short help message

`-man` Display man-style help message

EXAMPLES:

This script is typically called with no arguments.

LOGGING:

This script generates a log file and also displays it to stdout at the end of processing. By default, the log is:

`/tmp/sdp_health_check.<timestamp>.log`

or

`/tmp/sdp_health_check.log`

The exception is usage errors, which result an error being sent to stderr followed usage info on stdout, followed by an immediate exit.

EXIT CODES:

An exit code of 0 indicates no errors or warnings were encountered.

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 v5.37.0:

`verify_sdp.sh [<instance>] [-online] [{-skip,-warn,-extra,-only} <test>[,<test2>,...]]`
`[-skip_summary] [-c|-csec] [-si] [-L <log>|off ] [-d|-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 expect p4d, p4broker, and/or p4p to be online. Any servers for which there are *_init scripts in the Instance Bin directory are checked. An error is reported if p4d is expected to be online and is not; warnings are displayed if p4broker or p4p are not online. The Instance Bin directory is the /p4/N/bin directory, where N is the SDP instance name.

-c Specify '-c' to call ccheck.sh to compare configurables, using the default config file: /p4/common/config/configurables.cfg

See 'ccheck.sh -man' for more information.

This option can only be used in Online mode; if '-c' is specified, '-online' is implied.

-csec

Specify '-csec' to call ccheck.sh checking only for security settings.

This option can only be used in Online mode; if '-csec' is specified, '-online' is implied.

`-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.
- * `commitid`: Skip check ensuring ServerID of commit starts with 'commit' or 'master'.
- * `masterid`: Synonym for commitid.
- * `offline_db`: Skip checks that require a healthy offline_db.
- * `owner`: Skip checks that require ownership of specific files/folders.
- * `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
```

The '-skip' option can be used with '-warn' and '-extra', but is mutually exclusive with '-only'.

`-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
```

The '-warn' option can be used with '-skip' and '-extra', but is mutually exclusive with '-only'.

```
-extra <test>[,<test2>,...]
```

Some tests are not executed by default, but are instead invoked only on request with the '-extra' option. The following tests are executed if specified with the '-extra' option:

- * commit_defined
- * owner
- * p4config
- * protections
- * remote_disabled
- * server_type_known
- * systemd_config
- * all (short for all of the above)

commit_defined: Do a test to check defined server specs, and ensure that exactly one server spec has a 'Services' field value of 'commit-server', and that no server specs are defined with a 'Services' field value of 'standard' (the obsolete predecessor to 'commit-server'). This test requires p4d to be online and thus implies '-online'.

p4config: Check that P4CONFIG points to a non-existent file. In SDP, the P4CONFIG value is intended to a non-existent file because P4CONFIG files override shell environment variable and can interfere with SDP operations.

owner: The check for proper ownership, when done with '-extra' are expanded to include \$LOGS and /p4/common.

protections: Do sanity check on the Protections table:
 * Ensure the last line of the Protections table starts with: 'super user <P4USER>'.

remote_disabled: This test ensures the legacy built-in user 'remote' is disabled. Any one of the following must be true to pass this

check:

- * P4D version is 2025.1+
- * security=4
- * A 'p4 protects -u remote' lists nothing but exclusionary mappings.

server_type_known: Do a test to confirm that exactly one of run_if_master.sh, run_if_replicas.sh, and run_if_edge.sh returns true. If 0 or more than 1 are true, report that as an error.

systemd_config: Check the systemd service configuration. This checks that:

- * Appropriate system services (p4d/p4broker/p4p) are defined corresponding init scripts existing in the Instance Bin directory, /p4/N/bin.
- * The p4d_N service, if defined, matches the template.
- * The p4broker_N service, if defined, matches the template.
- * The p4p_N service, if defined, matches the template.

The '-extra' option can be used with '-skip' and '-warn', but is mutually exclusive with '-only'.

-only <test>[,<test2>,...]

Use the '-only' option to execute only specified tests. If this option is used, even tests that cannot be skipped with '-skip' and thus are usually always executed are not executed. This option is primarily intended to support testing of verify_sdp.sh.

Only a limited set of tests can be specified with '-only', including:

- * crontab
- * commit_defined
- * commitid
- * excess
- * owner
- * p4config
- * protections
- * remote_disabled
- * server_type_known
- * systemd_config
- * version

The '-only' option is mutually exclusive with '-skip', '-warn', and '-extra'.

-skip_summary

By default, if any errors or warnings are displayed in the output, a summary of those errors appears at the end. Specify this option to avoid displaying the summary at the end.

-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 Enabled debug messages.

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

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 that is already doing its own logging, use -L off:

```
verify_sdp.sh 1 -L off
```

Example 4: Include online checks

This examples relies on the \$SDP_INSTANCE variable rather than passing the <instance> parameter.

```
verify_sdp.sh -online
```

Example 5: Thorough Usage

Run all available tests (-online is implied):

```
verify_sdp.sh -extra all
```

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. SDP Trigger Scripts

The following table describes SDP trigger scripts.

Table 1. Summary of SDP Trigger Scripts

Script	Comments
<code>enforce_change_type.sh</code>	If the <code>defaultChangeType</code> configurable is set, this script should be used to ensure users cannot bypass the default.
<code>pull.sh</code>	This script illustrates using archive pull triggers to change the replication mechanism between a commit and edge server to use a 3rd party (and thus unsupported by Perforce) alternative archive file transfer mechanism.
<code>pull_test.sh</code>	This is a test script related to <code>pull.sh</code> .
<code>sdp_info.sh</code>	This script appends the SDP version and optional message-of-the-day to <code>p4 info</code> output.
<code>SetWsOptionsAndView.py</code>	Trigger to modify default client spec <code>Options:</code> and views. Use either <code>SetWsOptions.py</code> or <code>SetWsOptionsAndView.py</code> , but not both.
<code>SetWsOptions.py</code>	Trigger to modify default client spec <code>Options:</code> . Use either <code>SetWsOptions.py</code> or <code>SetWsOptionsAndView.py</code> , but not both.
<code>submit.sh</code>	This is a companion to <code>pull.sh</code> .
<code>submit_test.sh</code>	This is test script related to <code>submit.sh</code> and <code>pull.sh</code> .



If you modify any SDP triggers, **DO NOT** modify them in place, or your changes will be removed during the next SDP upgrade. Instead, create a `/p4/common/site/bin/triggers` directory, copy the trigger script(s) you want to modify from `/p4/common/bin/triggers` to `/p4/common/site/bin/triggers`, and make the modifications to the files in `/p4/common/site/bin/triggers`. Then adjust the Triggers table entry to call your local version in `/p4/common/site/bin/triggers`

rather than the default version. The `/p4/common/site` is not affected by future SDP upgrades. Files under `/p4/common` anywhere other than the `site` directory are subject to being changed during future SDP upgrades.

8.6.1. enforce_change_type.sh

If the `defaultChangeType` configurable is set, this script should be used to ensure users cannot bypass the default.

This will take either a changelist number (all numeric) or a temp file path (the form file), and make adjustments as needed to ensure all changelists are restricted. This works in conjunction with having the 'defaultChangeType' configurable set to 'restricted'. This script ensures there is no sneaking around the policy.

Install in triggers table with a pair of entries like this:

```
company.protectform form-in change
"/p4/common/site/bin/triggers/enforce_change_type.sh %formfile%"
company.protectform change-commit //...
"/p4/common/site/bin/triggers/enforce_change_type.sh %changelist%"
```

Future Proofing consideration: Currently, the only possible values for the Type field of a changelist are 'public' and 'restricted'. This is not expected to change. If it ever did for some reason, this code would need review.

8.6.2. 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 8.7.30, “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 they 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 8.6.3, “pull_test.sh”](#)

See the `/p4/common/bin/triggers/pull.sh` script for details and to customize for your environment.

8.6.3. pull_test.sh

The `/p4/common/bin/triggers/pull_test.sh` script is a test script.



THIS IS A TEST SCRIPT - it substitutes for [Section 8.6.2, “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>

8.6.4. sdp_info.sh

This script appends the content of the SDP Version file, `/p4/sdp/Version`, to the p4 info output.

This also appends the content of an optional message-of-the-day (motd) file to p4 info output.

Install in the Triggers table with this entry:

```
SDPInfo command post-user-info "/p4/common/bin/triggers/sdp_info.sh %clientprog%"
```

Once the trigger is installed, commands using the **p4** client binary (but not other clients such as P4V), when doing a **p4 info** command, will display an additional line of text containing the SDP version, looking something like this:

```
Rev. SDP/MultiArch/2025.1/31674 (2025/06/04).
```

If a "Message of the Day" (motd) file exists as **/p4/common/site/config/motd.txt**, its contents will also be appended after the SDP Version.

8.6.5. SetWsOptionsAndView.py

This script is designed to run as a form out trigger on the server. It will set default for new clients as follows:

- Preset the options set in the **OPTIONS** variable,
- Change the SubmitOption from "submitunchanged" to "leaveunchanged"
- Initialize the view to something that forces the user to set a reasonable default.
- Pass through -t template content unmodified.

Install in the Triggers table with this entry:

```
SetWsOptsAndView form-out client "/p4/common/bin/triggers/SetWsOptionsAndView.py  
%formfile% %argsQuoted%"
```



Do not use both **SetWsOptions.py** and **SetWsOptionsAndView.py**.

8.6.6. SetWsOptions.py

Install in the Triggers table with this entry:

```
SetWsOpts form-out client "/p4/common/bin/triggers/SetWsOptions.py %formfile%"
```

This script is designed to run as a form out trigger on the server affecting client specs. It will preset the options set in the **Options:** field of the client spec. It also changes the submit option from "submitunchanged" to "leaveunchanged".



Do not use both **SetWsOptions.py** and **SetWsOptionsAndView.py**.

8.7. 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.

8.7.1. 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.

8.7.2. 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
```

8.7.3. ccheck.sh

The script `/p4/common/bin/ccheck.sh` script compares configurables against a set of defined best practices.

Usage

USAGE for ccheck.sh v2.3.2:

```
ccheck.sh [<SDPInstance>] [-p <Profile>] [-fix|-FIX] [-sec [-no_ssl]] [-p4config
/path/to/.p4config] [-c <CfgFile>] [-y] [-v] [-d|-D]
```

or

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

DESCRIPTION:

This script compares configurables set on the current server with best practices. When used to check for security best practices, use the '-sec' option.

The best practices are defined in a data file. The default data file

provided is:

```
/p4/common/config/configurables.cfg
```

Optionally, if the '-fix' or '-FIX' options are used, this script can make changes to p4d, via 'p4 configure' commands, to bring it in line with best practices. A preview of proposed changes is then displayed. The '-y' option can then be used to proceed with the operation.

This script is currently intended to run only on a commit server. It may be updated in the future to provide further information for replica and edge servers.

OPTIONS:

-p <Profile>

Specify a profile defined in the config file, such as 'demo' or 'hcc'. A profile defines a set of expected configurable values that can differ from the expected values in other profiles. For example, for a demo environment, the `filesystem.P4ROOT.min`

might have an expected value of 128M, while the expected value in a prod (production)

profile might be 5G, and the same value might be 30G for 'prodent', the profile for

production at large enterprise scale.

The 'always' profile defines settings that always apply whether '-p' is specified or not. The profile specified with '-p' applies in addition to the 'always' configuration, adding to and possibly overriding settings from the 'always' configuration.

The default profile is 'prod', the production profile.

Specify the special value '-p none' to use only the settings defined in the 'always' profile.

-sec Specify '-sec' to do a focused security check.

This option also adds one additional check to ensure that the P4PORT value used is SSL-enabled (unless '-no_ssl' is specified).

-no_ssl

Specify '-no_ssl' with '-sec' to bypass the check for an SSL-enabled P4PORT.

-p4config <P4CONFIG_File>

Specify the path to a P4CONFIG file containing P4PORT, P4USER, P4TICKETS, and P4TRUST settings to connect as a super user to any P4 Server.

This can be used with '-sec' on an SDP-managed server to check security settings of a non-SDP server.

If the server targeted by the P4CONFIG settings is a non-SDP server the '-sec'

option is not used, expect some errors related to not following best practices, some of which may be SDP-specific.

Before using '-p4config' with this script, first create and test your P4CONFIG file using the 'p4' command line client with the '-E' option to exercise the settings.

Start by creating the P4CONFIG file in /p4/common/config, a good location for site-local config files.

```
mkdir -p /p4/common/config
echo P4PORT=ssl:OtherServer:1666 > /p4/common/config/.p4config.OtherServer
echo P4USER=bruno >> /p4/common/config/.p4config.OtherServer
echo P4TICKETS=/p4/common/config/.p4tickets.OtherServer >>
/p4/common/config/.p4config.OtherServer
echo P4TRUST=/p4/common/config/.p4trust.OtherServer >>
/p4/common/config/.p4config.OtherServer
```

Next, establish trust (unless the other server is not SSL-enabled):

```
p4 -E P4CONFIG=/p4/common/config/.p4config.OtherServer trust -y
```

Then login (which may require SSO authentication if your other server is thusly configured):

```
p4 -E P4CONFIG=/p4/common/config/.p4config.OtherServer login
```

Ensure you are a super on the other server, and that Protections on the other server allows you to connect from your current machine. You may need to modify Protections on the other server to get this to work:

```
p4 -E P4CONFIG=/p4/common/config/.p4config.OtherServer protects -m
```

Then give it a try, e.g.:

```
ccheck.sh -sec -p4config /p4/common/config/.p4config.OtherServer
```

-c <CfgFile>

Specify an alternate config file to define best practice configurables. This is intended primarily for testing. It can also be useful to define a site-local definition of best practices to compare against. To use this option, first copy the default file to create a local copy in the /p4/common/config, e.g.

```
cp -p /p4/common/config /p4/common/config/configurables.cfg
```

Then reference it with '-c /p4/common/config/configurables.cfg'.

WARNING: If you maintain a site-local copy of configurables.cfg, you will need to keep it current after SDP upgrades by manually merging in changes from the latest SDP version delivered with each release. Thus, using this

option is discouraged.

-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 a log file pointed to by a symlink:

`$LOGS/ccheck.log`

The symlink is for convenience. It refers to the log from the most recent run where '-L' was not used.

Each time this script is run, a new timestamped log is started, and the symlink updated to reference the new/latest log during startup. Log files have timestamps that go to the second (or millisecond if needed) to differentiate logs.

NOTE: This script is self-logging. That is, output displayed on the screen is simultaneously captured in the log file. Using redirection operators like '> log' or '2>&1' are unnecessary, as is using 'tee' (though using 'tee' or redirects is safe and harmless).

-fix Specify -fix to take corrective action to resolve differences between current settings and the recommended/required values. Optional settings are not affected by '-fix', only those indicated as Recommended or Required.

When -fix is specified, this script determines on a per configurable basis whether it is safe to proceed immediately with the advised change, or if should be deferred until potentially disruptive impacts are understood. This determination is made based on the configuration file, which provides links go guidance documentation for configurables that are best changed with awareness of potential impact. Such changes are displayed with '-fix', but require use of '-FIX' to process.

As an example and a special case, the 'security' configurable will be changed to 4 with '-fix' if the starting value is 3, because that is deemed a low-risk change. Changing the 'security' value to 4 if the starting value is 0-2 requires -FIX, as that is more likely to be impactful to users, and thus is best done with coordinated communications.

In any case, if the security configurable is to be modified, additional guidance is provided regarding the potential impact to other p4d servers that access this server using the remote depot feature. Such access via the remote depot feature will cease to function when security is set to 4.

This option previews advised changes by default. Use with -y to make changes.

-FIX Specify -FIX to make all changes that have been automated, even those that '-fix' would refuse to process immediately.

In some cases even with `-FIX`, there may be follow up work to do. Follow up tasks will be indicated with 'TO DO:' comments in the output.

The `-FIX` option implies `-fix`.

This option previews advised changes by default. Use with `-y` to make changes.

`-y` Live operation mode. By default, any commands that affect data, such as setting configurables, are displayed, but not executed. With the `-y` option, commands may be executed.

`-d` Display debug messages.

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

`-si` Silent Mode. No output is displayed to the terminal (except for usage errors on startup). Output is captured in the log. The `'-si'` cannot be used with `'-L off'`.

HELP OPTIONS:

`-h` Display short help message.

`-man` Display man-style help message.

`-V` Display script name and version.

GENERAL ADVICE and DISCLAIMER:

This script is based on a data file that represents generalized best practices. This data file should be considered a source of information to be checked against other sources, such as the System Administration Guide and/or documentation on specific configurables found here:

<https://help.perforce.com/helix-core/server-apps/cmdref/current/Content/CmdRef/configurables.alphabetical.html>

Before acting on information provided by the script, and especially before using the `'-fix'` and `'-FIX'` options, be sure to review the output carefully.

Contact Perforce Technical Support for guidance as needed.

EXAMPLES:

Example 1: Basic Check, No Logging

Check configurables with the default profile ('prod' for a commercial production server) and no logging:

```
ccheck.sh -L off
```

Example 2: Alternate Profile

Check configurables with the 'pub' profile (for a public/open source server):

```
ccheck.sh -p pub
```

Example 3: Verbose comparison

Check configurables with the 'demo' profile, doing a verbose comparison:

```
ccheck.sh -p demo -v
```

Example 4: Security focused check

Use the '-sec' option to report only security-related settings:

```
ccheck.sh -sec
```

Example 5: Security focused check with fixes

To do a security focused check making only non-disruptive fixes, use '-fix', and start with a preview by omitting the '-y' option:

```
ccheck.sh -sec -fix
```

If the output is correct, append the '-y' to the command to make changes:

```
ccheck.sh -sec -fix -y
```

Example 6:

To do a security focused check, including potentially disruptive fixes, use '-FIX', and start with a preview by omitting the '-y' option:

```
ccheck.sh -sec -FIX
```

Be sure to review the output carefully and read about potential impacts. Some configurables, such as 'auth.id'. require planning to change. For those configurables that require planning, additional guidance is provided if the script advises those settings be changed. Read and heed the guidance.

In the case of 'auth.id' specifically, this script will call 'p4login -v', 'p4 login -service', and 'p4login -v -automation' if that configurable is set, account for the need to login again after setting 'auth.id'. The script also displays a warning indicating that those commands should also be run manually on any other server machines in the fleet.

If the output is correct, append the '-y' to the command to make changes:

```
ccheck.sh -sec -FIX -y
```

FUTURE ENHANCEMENTS:

- * Add multi-version support for backward compatibility. This version assumes P4D 2024.2+ (though it may be useful for older versions).

FILES:

The default configurables config file is: `/p4/common/config/configurables.cfg`

This file contains further documentation on the format of entries in the file.

8.7.4. 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 8.7.25, “recover_edge.sh”](#)

8.7.5. 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.

8.7.6. 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.

8.7.7. 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 8.4.11, “recreate_offline_db.sh”](#).

Usage

USAGE for load_checkpoint.sh v3.2.6:

```
load_checkpoint.sh {<checkpoint> [<jnl.1> <jnl.2> ...] | -latest | -latest_jnls | -jo
<jnl.1> [<jnl.2> ...] | -jo_latest } [-R|-F <SafetyFactor>] [-i <instance>] [-s
<ServerID>] [-t <Type>] [-no_start | [-no_xu] [-verify {default|"Verify Options"} [-
delay <delay>]]] [-c] [-l] [-r] [-b] [-y] [-L <log>] [-si] [-d|-D]
```

or

```
load_checkpoint.sh [-h|-man]
```

DESCRIPTION:

This script can load a specified checkpoint and/or numbered journals into P4ROOT (/p4/N/root) and/or /p4/N/offline_db (where 'N' is the SDP instance name). It supports a variety of use cases for replaying checkpoints and journals, including:

- * Seeding or Reseeding a replica or edge server.
- * Loading a checkpoint on the commit, e.g. in a recovery scenario.

Checkpoints and/or journals can be specified in one of two ways: they can be specified as parameters to this script, or they can be determined by this script if they appear in the SDP standard location according to the journalPrefix standard. The key methods are:

- * Specify the path to the checkpoint to replay. The checkpoint can be in the form of a compressed .gz file, an uncompressed checkpoint file, or a directory (for parallel checkpoints).
- * Use '-latest' to have this script find the latest checkpoint available. For a commit server, /p4/N/checkpoints/p4_N is searched. For other servers, their journalPrefix is used. The timestamp on the latest available *.md5 file is used to determine what checkpoint is the latest available, regardless of checkpoint form (compressed or uncompressed file, or a directory for parallel checkpoints).
- * Use '-latest_jnls' to find the latest checkpoint as with '-latest', and then also find and replay any available subsequent numbered journals.
- * Use '-jo' ("journal only") to specify path(s) to one or more numbered journals to be supplied as parameters to this script. Journal files provided may be compressed or uncompressed.
- * Use '-jo_latest' to find any numbered journals available to be replayed based on the journal counter of the data set.

At the start of processing, preflight checks are done. Preflight checks include:

- * The specified checkpoint and corresponding *.md5 file must exist.
- * The specified checkpoint can be a compressed or uncompressed file or a directory (for parallel checkpoints).
- * All journal files to replay (if any are 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 (such as an edge server).
- * The SDP structure and key files must exist.
- * Disk space checks are done to attempt to determine if sufficient space is available to replay the checkpoint.

If the preflight passes, the p4d_N service is shutdown. The p4broker_N service is shutdown if it is 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 specified in a list of journals to replay, then it is not moved aside).

Next, any existing state* files in P4ROOT are removed.

Next, any existing database files in P4ROOT are preserved and moved aside, unless '-R' is specified to remove them.

Next, the specified checkpoint is loaded. Upon successful completion, 'p4d -xu' is executed (by default) to help ensure the service can be started with the p4d binary used to replay the checkpoint. Then the Helix Core service is started with the current p4d binary.

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.

Finally, the offline_db is rebuilt using the same specified checkpoint and journals.

ARGUMENTS AND OPTIONS:

<checkpoint>

Specify the path to the checkpoint file or directory to load. Exactly one checkpoint must be specified. If a checkpoint file is specified, a serial

checkpoint replay will be done. If a checkpoint directory is specified, a parallel replay will be done using the individual files in the directory.

For checkpoint files:

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.

For checkpoint directories:

This option is required unless the '-latest' option is used.

<jnl.1> [<jnl.2> ...]

Specify the path to the one or more journal files to replay after the checkpoint, in the correct sequence order.

-latest

Specify this as an alternative to providing a specific checkpoint file or directory. The script will then search for the latest *.md5 file in the standard checkpoints directory and use that to replay.

The standard checkpoints directory search is one of the following:

Commit servers: /p4/N/checkpoints

Standby servers: /p4/N/checkpoints

Edge servers: /p4/N/checkpoints.<ShortServerID>

For standby servers that target an edge server, where the ServerID starts with p4d_ha_edge, p4d_ham_edge, p4d_fs_edge, or p4d_fsm_edge, the directory for the target edge server is searched. (If NFS sharing, this directory will naturally exist. Otherwise, the directory should be created and populated as needed on the standby of the edge for seeing with checkpoints from the edge.

The most recent *.md5 file found in the standard checkpoints directory determines which checkpoint to load. The actual checkpoint can be a file (gzipped or not) or directory (for parallel checkpoints).

This option is mutually exclusive with '-latest_jnls'.

-latest_jnls

This option is similar to '-latest'. However, with '-latest_jnls', in addition to replaying the latest checkpoint, any subsequent numbered journals available in the standard checkpoints directory are also replayed.

This option will only replay numbered journals, not the live P4JOURNAL file. However, if the \$P4JOURNAL is provided, then it will be replayed after all available numbered journals are replayed.

This option is mutually exclusive with '-latest'.

If used with '-jo', where the checkpoint and possibly some numbered journals will already have been replayed into P4ROOT, then the meaning of this option changes. It will replay needed numbered journal up to the latest available, so long as those journals appear in the standard checkpoints directory with the usual naming convention. With this option, the journal needed are calculated based on the journal counter stored in database in the P4ROOT dir.

-R Specify '-R' to remove db.* files in P4ROOT rather than moving them aside.

By default, databases are preserved for possible future for investigation. A folder named 'MovedDBs.<datestamp>' is created under the P4ROOT directory, and databases are moved there.

Keeping an extra set of databases requires sufficient disk space to hold the extra set of db.* files.

If -R specified, old databases in P4ROOT are removed, along with state* and other files, and the server.locks directory.

-F <SafetyFactor>

When replacing an existing set of db.* files, a safety factor is used. This is simply the factor by which the size of pre-existing databases is multiplied when comparing against available disk space.

Specify '-F 0' to disable the safety factor check.

The disk space safety check is only meaningful if P4ROOT was previously populated with a full set of data.

Specifying a number greater than 1, say 1.2 (the default) gives more breathing room.

Specifying a value lower than 1, say 0.95, may be OK if you are certain the expanded-from-a-checkpoint db.* files are significantly smaller than size the prior set of db.* files.

This option is mutually exclusive with '-R'. If '-R' is used, databases are removed, and there is no need to calculate disk space.

-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 commit 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 commit can be used.

For a filtered forwarding replica, a proper seed checkpoint must be loaded. This can be created on the commit using key options to p4d, including '-P <ServerID> -jd <SeedCkp>' on the commit (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 commit 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. The options to pass to p4verify.sh can be passed in a quoted list, or '-verify default' can be used to indicate these default 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 caught up with metadata before the archive pulls are scheduled. The delay is a workaround for job079842.

This option is cannot be used with '-no_start'.

-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', 'ham', 'fs,' or 'fsm'. Replicas that are potential targets for a 'p4 failover' need a license file for a failover to work.

-r Specify '-r' to replay only to P4ROOT. By default, this script replays both to P4ROOT and the offline_db.**-no_start**

Specify '-no_start' to avoid starting the p4d service after loading the

checkpoint.

This option is cannot be used with '-verify'.

-no_xu

Specify '-no_xu' to skip the 'p4d -xu' step that upgrade the database schema.

By default, a 'p4d -xu' is done to help ensure the service can be started with the current p4d binary after the checkpoint is replayed.

If the p4d binary used to replay the checkpoint is a newer major version than the one used to create the checkpoint, the service will not start after the replay until the 'p4d -xu' step is done. If this '-no_xu' option is used and the p4d binary is a newer major version, have a plan to get the 'p4d -xu' done before the service is started.

In EXAMPLES below, see the example titled "Multi Pass Replay of Checkpoints and Journals" for an example of using this option as part of a migration procedure.

-jo <jnl.1> [<jnl.2> ...]

Specify '-jo' to replay only one or more numbered journals without first replaying a full checkpoint. With this option, the cleanup that normally occurs before the replay is disabled. The db.* and state* files in P4ROOT, as well as P4LOG and P4JOURNAL files, etc. are left in place.

With '-jo', the paths to journal files must be specified.

This option is mutually exclusive to the similar option '-jo_latest'.

This option implies '-r'.

-jo_latest

Specify '-jo_latest' to replay only one or more numbered journals without first replaying a full checkpoint. With this option, the cleanup that normally occurs before the replay is disabled. The db.* and state* files in P4ROOT, as well as P4LOG and P4JOURNAL files, etc. are left in place.

With '-jo_latest', numbered journals to replay are calculated and determined, not specified as parameters.

This option is mutually exclusive to the similar option '-jo'.

This option implies '-r'.

-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.
- 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 debugging verbosity.
- D Extreme debugging verbosity using bash 'set -x' mode.

HELP OPTIONS:

- h Display short help message
- man Display man-style help message

USAGE TIP:

All the non-interactive examples below illustrate the practice of using redirects to create an extra log file named 'load.log' in the \$LOGS directory for the instance. This load.log file is identical to, and in addition to, the standard timestamped log generated by this script. The intent of this practice is to make it easier to find the log for the last checkpoint loaded on any given server machine. This convention is only useful if used consistently.

Several examples below illustrate the instance option, '-i' option to specify the SDP instance. This is optional and can safely be omitted in an environment where the standard SDP shell environment is sourced on login, and where there is only a single instance on the server machine.

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 > /p4/1/logs/load.log 2>&1 &
```

Then, monitor with:

```
tail -f $(ls -t $LOGS/load_checkpoint.*.log|head -1)
```

EXAMPLE 2: Checkpoint Load then Verify, for the SDP Instance alpha.

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/alpha/checkpoints/p4_alpha.ckp.95442.gz -i alpha
-verify -recent -nu -ns -y < /dev/null > /p4/alpha/logs/load.log 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 >
/p4/1/logs/load.log 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 > /p4/1/logs/load.log 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 > /p4/1/logs/load.log 2>&1 &
```

EXAMPLE 7: Load a Parallel Checkpoint on an Edge and Verify Recent

This non-interactive example loads a parallel checkpoint directory. The usage difference is that the checkpoint path provided is a parallel checkpoint directory rather than a single checkpoint file. This example loads the checkpoint for a new edge server, and verifies only the most recent 3 changes in each depot. The delay before calling `p4verify.sh`, 10 minutes (600) by default, is shortened to 5 seconds in this example.

```
nohup /load_checkpoint.sh /p4/1/checkpoints/p4_1.ckp.4025 -i 1 -s p4d_edge_syd
-verify "-o MISSING -recent=3 -ns -L /p4/1/logs/p4verify.fast_and_recent.log" -delay 5
-y < /dev/null > /p4/1/logs/load.log 2>&1 &
```

EXAMPLE 8: Multi Pass Replay of Checkpoints and Journals

In this example, we want to use a multi-pass procedure involving replay of a checkpoint at one point in time, and then later replay subsequent numbered journals later. This method can be useful to reduce downtime required for migration procedures involving a checkpoint replay if the checkpoint replay takes a while, e.g. a few hours or more. The gist of the approach is to replay the checkpoint a day or so ahead of the scheduled maintenance. Then replay subsequent numbered journals each day after. Then in the maintenance window, replay just the last numbered journal from the old environment in the new environment.

This approach involves a few options:

- * When the checkpoint is replayed, '-no_start' and '-no_xu'. Either specify the path to the checkpoint, or use '-latest'.
- * When the numbered journals are replayed in days leading up to the maintenance window, use the '-jo_latest' option to replay only a numbered journal.
- * During the maintenance window, load any final numbered journals, then start the service.

Pass 1, 3 days before maintenance:

```
nohup load_checkpoint.sh -latest -no_start -no_xu -r -y < /dev/null >
/p4/1/logs/load.log 2>&1 &
```

Pass 2, 2 days before maintenance:

```
nohup load_checkpoint.sh -jo_latest -no_start -no_xu -y < /dev/null >
/p4/1/logs/load.log 2>&1 &
```

Pass 3, 1 day before maintenance:

```
nohup load_checkpoint.sh -jo_latest -no_start -no_xu -y < /dev/null >
/p4/1/logs/load.log 2>&1 &
```

Pass 4, during the maintenance window:

```
nohup load_checkpoint.sh -jo_latest -y < /dev/null > /p4/1/logs/load.log 2>&1 &
```

8.7.8. 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
```

8.7.9. gen_sudoers.sh

The `/p4/sdp/Server/Unix/setup/gen_sudoers.sh` script generates a 'sudoers' file that grants the limited sudoers access to the OS user that runs p4* services, typically 'perforce'. The grant honors the principle of least privilege.

The `gen_sudoers.sh` script called by `install_sdp.sh` during initial setup.

It can be run manually to add limited sudoers if unrestricted sudoers was initially specified (or vice-versa).

It must be run as **root**.

For more information:

Usage

USAGE for gen_sudoers.sh v1.6.0:

```
gen_sudoers.sh {-full|-limited} [-y [-f]] [-L <log>] [-D]
```

or

```
gen_sudoers.sh [-h|-man]
```

DESCRIPTION:

This script generates a sudoers file for the OS user that owns /p4/common, which must be the same user that the p4* services runs as under systemd, typically 'perforce'. The intent is to honor the principle of least privilege. The generated sudoers file grants the access to user commands reasonably needed for service management (e.g. service start/stop) and diagnostics (e.g. lslocks, lsof), but no more.

By default, the sudoers file is generated for review. If the '-y'

option is specified, the newly generated files is installed as the live sudoers file by copying to `/etc/sudoers.d/<OSUSER>` and adjusting permissions to `0400`.

If `'-full'` (full sudo) is specified, a one-line sudoers file is generated that looks something like this:

```
performer ALL=(ALL) NOPASSWD: ALL
```

Warning: Using `'-full'` does NOT honor the principle of least privilege, and thus grants full sudo. This is not recommended because it elevates access of remote super users.

If `'-limited'` is specified, a limited sudoers file is generated granting only necessary access to the performer user, and thus honoring the principle of least privilege.

If the sudoers file already exists, it will not be updated unless `'-f'` (force) is provided.

The limited sudoers option is recommended for production deployments.

This script must be executed as the root user.

OPTIONS:

`-full`

Specify `'-full'` to indicate that a sudoers file is to be generated granting full root access to the server machine.

The `'-full'` or `'-limited'` option must be specified.

This option is discouraged as it is not as secure as the `'-limited'` option.

`-limited`

Specify `'-limited'` to indicate that a sudoers file is to be generated granting limited access to the server machine.

The `'-full'` or `'-limited'` option must be specified.

This option is recommended for optimal security.

`-y` This is confirmation to install the generated sudoers as the live sudoers file.

`-f` Specify `'-f'` to overwrite an existing limited sudoers file, `/etc/sudoers.d/<OSUSER>`

`-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:

```
/root/gen_sudoers.<Datestamp>.log
```

NOTE: This script is self-logging. That is, output displayed on the screen is simultaneously captured in the log file.

-D Enable bash 'set -x' extreme debugging verbosity.

HELP OPTIONS:

-h Display short help message
-man Display man-style help message

EXAMPLES:

EXAMPLE 1: Generate a limited sudoers file for review.

```
cd /p4/sdp/Server/Unix/setup
./gen_sudoers.sh -limited
```

EXAMPLE 2: Generate a limited sudoers file and install it.

```
cd /p4/sdp/Server/Unix/setup
./gen_sudoers.sh -limited -y
```

EXAMPLE 3: Generate a limited sudoers file and install it, replacing an existing one.

```
cd /p4/sdp/Server/Unix/setup
./gen_sudoers.sh -limited -f -y
```

EXAMPLE 4: Generate a full sudoers file and install it, replacing an

```
cd /p4/sdp/Server/Unix/setup
./gen_sudoers.sh -full -f -y
```

8.7.10. journal_watch.sh

The `/p4/common/bin/journal_watch.sh` script will check disk space 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 **KEEPJNLS** 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
```

8.7.11. 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
```

8.7.12. mkdirs.sh

The **mkdirs.sh** script is intended for the setup and configuration of a **new** Helix Core instance. It should be run only for adding a new instance, not against an existing instance.

Usage

USAGE for mkdirs.sh v7.4.11:

```
mkdirs.sh <instance> [-r <P4BinRel>] [-s <ServerID>] [-t <ServerType>] [-tp
<TargetPort>] [-lp <ListenPort>] [-I <svc>[,<svc2>]] [-MDD /bigdisk] [-MCD /ckps] [-
MLG /jnl] [-MDB1 /db1] [-MDB2 /db2] [-f] [-p] [-no_init|-no_systemd|-no_enable] [-fs|-
ls] [-cleartext] [-no_cron] [-no_firewall] [-no_broker] [-test [-clean]] [-n] [-L
<log>] [-d|-D]
```

OR

```
mkdirs.sh <instance> [-c <CfgFile>] [-f] [-p] [-no_init|-no_systemd|-no_enable] [-fs|-ls] [-cleartext] [-no_cron] [-no_firewall] [-no_broker] [-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. In this case, the user calls the `install_sdp.sh` script, which in turn calls this script. See '`install_sdp.sh -man`' for more information.
- * Adding new SDP instances (separate Helix Core data sets) to an existing SDP installation on a given machine. For this scenario, this `mkdirs.sh` script is called directly.

An 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 *must* be run as root for the first instance on a machine; this first call is handled by `install_sdp.sh`.

Call `mkdirs.sh` directly when adding additional instances to be co-located on the same SDP server machine. Generally, `mkdirs.sh` should always be run as root, as it does a more complete job. If not run as root, the following steps are not done:

- * Init mechanism installation, e.g. Systemd service or SysV init script installation.
- * SELinux configuration
- * Firewall configuration
- * Ownership (`chown`) commands

== 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
* /hxcheckpoints
* /hxdepots
* /hxlogs
* /hxmetadata
* /hxmetadata2
* /opt/perforce/helix-sdp (optional; used for package installations)
```

The directories starting with '/hx' are configurable, and can be changed by settings in the `mkdirs.cfg` file (or `mkdirs.N.cfg`), or with command line options as illustrated here:

```
-MDD /bigdisk
-MCD /ckps
-MLG /jnl
-MDB1 /db1
-MDB2 /db2
```

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

== Crontab ==

Crontabs are generated for all server types.

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).

== Init Mechanism ==

If this script is run as root, the init mechanism (Systemd or SysV) is configured for installed services.

The Systemd mechanism is used if the `/etc/systemd/system` folder exists and `systemctl` is in the PATH of the root user. Otherwise, the SysV init mechanism is used.

== Firewall Configuration ==

This script checks to see if a known firewall type is available. The `firewalld` is checked using the command 'firewall-cmd --state' command,

and the ufw firewall is checked using the 'ufw status'. If either firewall is detected, the ports required for Helix Core applications installed are opened in the firewall. For more information, see the templates in these folders:

```
/p4/common/etc/firewalld
/p4/common/etc/ufw
```

If the firewall service is not online, no firewall configuration is performed.

== SELinux Configuration ==

If Systemd is used and the semanage and restorecon utilities are available in the PATH of the root user, then SELinux configuration for the installed services is done.

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_commit - A master/commit server.
- * p4d_master - A synonym for p4d_commit.
- * 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.
- * p4p - An SDP host running only a standalone p4p, with no p4d.
- * p4proxy - A synonym for p4p.

-tp <TargetPort>

Specify the target port. Use only if ServerType is p4p and p4broker.

-lp <ListenPort>

Specify the listen port. Use only if ServerType is p4p 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. 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 that call the init scripts are supplied in `/p4/common/etc/systemd/system`.

`-MDD /bigdisk`

`-MCD /ckps`

`-MLG /jnl`

`-MDB1 /db1`

`-MDB2 /db2`

Specify the `'-M*'` options to specify mount points, overriding DD/CD/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.

`-c <CfgFile>`

Specify the path to the configuration file to use, overriding the default logic of finding the file based on naming convention.

`-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 time-consuming and unnecessary.

`-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.

`-no_init`

Specify `'-no_init'` to avoid any service configuration, which is done by default if running as root (using systemd if available, otherwise SysV). If `'-no_init'` is used, then neither systemd nor SysV init mechanism is configured for installed services.

This option is implied if not running as root.

This option is implied if `'-test'` is used.

-no_systemd

Specify `'-no_systemd'` to avoid using systemd, even if it appears to be available. By default, systemd is used if it appears to be available.

This is helpful in operating in containerized test environments where systemd does not work even if it appears to be available.

This option is implied if the `systemctl` command is not available in the PATH of the root user.

This option is implied if `'-no_init'` is used.

-no_enable

Specify `'-no_enable'` to avoid enabling systemd services to start automatically after a reboot. If this option is used, systemd services will still be created, allowing services to be manually started and stopped.

Specifically, this options means the `'systemctl enable'` command is not run for generated services.

-no_cron

Specify `'-no_cron'` to avoid loading the crontab.

A crontab file is generated in the `/p4` directory, but but with `'-no_cron'`, this file is not loaded as the active crontab.

-no_firewall

Specify `'-no_firewall'` to avoid attempting firewall configuration.

By default, if the `firewalld` service is found to be running, it is configured so that the ports for `p4d` and `p4broker` are open.

-no_broker

Specify `'-no_broker'` if installing a `p4d` service without a broker on the same server machine as `p4d`. By default, a broker is included with `p4d`.

-fs Specify `'-full'` when calling `gen_sudoers.sh` to install a new, full sudoers file. This option is only available if running as root.

This option is mutually exclusive with `'-ls'`.

See `'gen_sudoers.sh -man'` for more info.

-ls Specify `'-limited'` when calling `gen_sudoers.sh` to install a new, limited sudoers file. This option is only available if running as root.

This option is mutually exclusive with '-fs'.

See 'gen_sudoers.sh -man' for more info.

Specifying neither '-fs' nor '-ls' avoids calling gen_sudoers.sh. This is not advised because it not work at all, or may result in a system that does not work properly or may be insecure. The recommended option is to use '-ls' for enhanced security.

-cleartext

By default, SDP passwords are generated as encoded files. If this option is used, cleartext passwords are generated instead.

if a cleartext password file is specified, the file will be:
/p4/common/config/.p4passwd.p4_<SDP_Instance>.admin

Encoded password files have that name with a '.enc' suffix.

-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>.<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'.

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/hxdepts
- * /tmp/hxmounts/hxlogs
- * /tmp/hxmounts/hxmetadata

This option implies '-no_init'.

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

Running 'makedirs.sh N' will load configuration settings from makedirs.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 makedirs.cfg
```

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

```
$ ./makedirs.sh 1
```

A log will be generated, makedirs.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 makedirs.sh on a given machine should be done as root, but subsequent instance additions can 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 mkdirs.cfg mkdirs.abc.cfg
$ chmod +w 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 targeting commit.example.com:1666 and listening locally on port 1666.

```
$ sudo su -
$ cd /hxdepots/sdp/Server/Unix/setup
$ ./mkdirs.sh alpha -t p4p -tp commit.example.com:1666 -lp 1666
```

Example 4: Setup of instance named '1' to run a standalone p4broker targeting commit.example.com:1666 and listening locally on port 1666.

```
$ sudo su -
$ cd /hxdepots/sdp/Server/Unix/setup
$ ./mkdirs.sh 1 -t p4broker -tp commit.example.com:1666 -lp 1666
```

Example 5: Setup 2 instances A and B with limited sudoers on a fresh new machine:

```
$ sudo su -
$ cd /hxdepots/sdp/Server/Unix/setup
$ cp mkdirs.cfg mkdirs.A.cfg
```

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

```
$ cp mkdirs.A.cfg mkdirs.B.cfg
```

Adjust settings in mkdirs.B.cfg as desired, e.g P4PORT, P4BROKERPORT, etc. Ensure port numbers do not conflict. Then generate Instance A:

```
$ ./mkdirs.sh A -ls
```

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

Next generate instance B, updating the limited sudoers to reference both instances.

```
$ ./mkdirs.sh B -ls
```

SEE ALSO:

See 'install_sdp.sh -man' for more info on installing on a new machine.

See 'gen_sudoers.sh -man' for more info on generating/replacing sudoers.

See template:

- * systemd service file templates: /p4/common/etc/systemd/system
- * firewalld templates: /p4/common/etc/firewalld
- * ufw firewall templates: /p4/common/etc/ufw
- * Init script templates: /p4/common/etc/init.d

8.7.13. **opt_perforce_sdp_backup.sh**

The **opt_perforce_sdp_backup.sh** script is intended to be called by the systemd **opt_perforce_sdp_backup.service** run by the systemd **opt_perforce_sdp_backup.timer**.

Usage

USAGE for opt_perforce_sdp_backup.sh v2.11.8:

opt_perforce_sdp_backup.sh [-d|-D]

or

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

DESCRIPTION:

This script is intended to be called by a systemd timer on systems that support systemd.

There is an **opt_perforce_sdp_backup.service** for this; it can be reviewed by doing:

```
$ systemctl cat opt_perforce_sdp_backup.service
```

This service is triggered by a systemd timer, which can be viewed by:

```
$ systemctl cat opt_perforce_sdp_backup.timer
```

If operating on a system without systemd, then it is OK to call this directly, e.g. via crontab. It must execute as root.

This script backs up key P4 Server Deployment Package (SDP) files and directories. It does not back up actual P4 Server application data. The job of this script is to ensure any SDP files that are stored on the local OS root volume are backed up a data volume that is backed up. If no data volume is found, an extra copy of files is made on the OS root volume.

To support backup to NFS volumes with root squash enabled, this script stages backup content in a temporary directory, creates a gzip-compressed tarball owned by the SDP owner user, and copies that tarball to the backup location. The tarball preserves all file ownership and permissions for recovery.

BACKUP LOCATION

This script determines the optimal backup location based on the server type being backed up. Some sample locations for backup are:

P4 Server: /hxdepots/backup/opt_perforce_helix-sdp.<ShortHostname>

P4 Proxy: /hxdepots/backup/opt_perforce_helix-sdp.<ShortHostname>

P4 Broker: /hxlogs/backup/opt_perforce_helix-sdp.<ShortHostname>

The backup consists of a gzip-compressed tarball and a recovery script:

- * sdp_backup.<ShortHostname>.tgz - Contains all backed up files

- * recover_opt_perforce_sdp.sh - Script to restore from the backup

GENERATED RECOVERY SCRIPT

Each time a successful backup completes, this script creates a recovery script. The recovery script is generated each time to ensure that site-specific settings such as SDP instances names and mount point locations are accounted for. This ensures that the recovery faithfully places files in the original structure.

The generated recovery script is in the backup directory and is named: recover_opt_perforce_sdp.sh

During each backup in which the generated recovery script changes, diffs from the prior recovery script are displayed, and the old recovery script is backed up in the backup location by moving it aside to a '.bak.<timestamp>' suffix, and the active file is updated. It is expected that this backup script will change infrequently, e.g. when new SDP instances are added (or removed), and possibly in future SDP version changes. In any case, the new script will update and replace the old.

This does whatever is needed to restore SDP as it was at the time of backup, including:

- * Creates the group of the SDP Owner if needed.
- * Creates the SDP Owner user if needed.
- * Restores the SDP Owner home directory if needed.
- * Restores the SDP Owner crontab.
- * Restores all SDP files and symlinks that exist outside the data volumes (e.g. /hxdepots, /hxlogs, /hxmetadata[1,2]).

It does NOT restore operating system package installations.

RECOVERY PROCEDURE

The recovery procedure extracts the backup tarball and runs the recovery script. Operating as root:

1. Create a working directory and copy the tarball there:

```
mkdir -p /root/opt_perforce_sdp_backup
cd /root/opt_perforce_sdp_backup
cp /hxdpots/backup/opt_perforce_helix-
sdp.<ShortHostname>/sdp_backup.<ShortHostname>.tgz .
```

2. Extract the tarball (this restores all files with correct ownership):
tar -xzf sdp_backup.<ShortHostname>.tgz
3. Run the recovery script from the extracted content:
cd opt_perforce_helix-sdp.<ShortHostname>
./recover_opt_perforce_sdp.sh

The gzip-compressed tarball preserves all file ownership and permissions, so extraction as root will restore files with their original owners (including root).

OTHER FILES TO RESTORE - HOME DIRECTORY

This does NOT backup files in the '' user home directory, as this is expected to be backed up by other means and/or per local policy, and is outside the scope of this script.

The home directory may include files that affect server operation, and thus should be recovered before using script. Some files to be recovered are:

- * ~/.bash_profile
- * ~/.bashrc
- * ~/.p4aliases.

Templates for these can be found in /p4/sdp/Server/Unix/setup/bash.

The home directory may also include an ~/.ssh directory and ~/.config and similar directories.

OPTIONS:

-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 a log file pointed to by a symlink:

opt_perforce_sdp_backup

The symlink is for convenience. It refers to the log from the most recent run of the script.

Each time this script is run, a new timestamped log is started, and the symlink updated to reference the new/latest log during startup. Log files have timestamps that go to the second (or millisecond if needed) to differentiate logs.

NOTE: This script is self-logging. Output displayed on the screen is

simultaneously captured in the log file. Using redirection operators like '> log' or '2>&1' are unnecessary, as is using 'tee' (though using 'tee' or redirects is safe and harmless).

-d

Display debug messages.

-D

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

-si

Silent Mode. No output is displayed to the terminal (except for usage errors on startup). Output is captured in the log. The '-si' cannot be used with '-L off'.

HELP OPTIONS:

-h Display short help message.

-man Display man-style help message.

-V Display script name and version.

FILES:

/etc/systemd/system/opt_perforce_sdp_backup.service

/etc/systemd/system/opt_perforce_sdp_backup.timer

LIMITATIONS:

If SELinux is used in enforcing mode, some 'semanage' and 'restorecon' commands may be needed on specific files after recovery. The generated recovery script does not currently handle SELinux contexts.

TO DO:

A future version of this script may preserve the crontab of the OSUSER.

8.7.14. 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.

8.7.15. p4broker_base

The `/p4/common/bin/p4broker_base` script is very similar to [Section 8.7.14, “p4d_base”](#) but for the `p4broker` service instance.

See [p4broker in SysAdmin Guide](#)

8.7.16. p4ftpd_base

The `/p4/common/bin/p4ftpd_base` script is very similar to [Section 8.7.14, “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](#).

8.7.17. p4p_base

The `/p4/common/bin/p4p_base` is very similar to [Section 8.7.14, “p4d_base”](#) but for the `p4p` (P4 Proxy) service instance.

See [p4proxy in SysAdmin Guide](#)

8.7.18. 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/-s]
or
p4pcm.pl -h
```

This utility removes files in the proxy cache if the amount of free disk space available to the cache falls below the low threshold (default 12). It removes cache files based on time last accessed starting with the least recently accessed continuing until either all files are deleted or the free disk space available to the cache specified by the high threshold (default 25) is reached. Specify numeric threshold values in kilobyte units (kb), or as a number less than 100 to specify percentage of the total disk space available to the cache. A high_threshold near the available disk space typically results in a full clear of the cache defeating the purpose of a proxy.

The `'-d "proxy_cache_dir"'` argument is required unless `$P4PCACHE` is defined. The `-d` argument takes precedence. `proxy_cache_dir` should be a fully rooted path starting with `'/'`. Relative or local paths are fatal to tool operation.

The log is `$LOGS/p4pcm.log` if `$LOGS` is defined, else `p4pcm.log` in the current directory.

The removal nomination file list is `$LOGS/p4pcm.nomlist` if `$LOGS` is defined, else `p4pcm.nomlist` in the current directory. The nomination list file contains an access time ordered list of all cache files. If the nomination list file exists at the start of this tool, the tool exits assuming a separate run is currently in progress. The nomination list file is deleted when the tool completes operation unless the `'-s'` argument is specified.

Use `'-n'` or `'-s'` to show what files would be removed. `'-s'` also causes the nomination list file to remain undeleted. The nomination list file must be

manually removed prior to a subsequent successful use of this tool.

8.7.19. p4review2.py

The `/p4/common/bin/p4review2.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 is not required if you have installed Swarm which also performs notification functions and is easier for users to configure.

1. Run `p4review2.py --sample-config > p4review.conf`
2. Edit the file `p4review.conf`
3. Add a crontab similar to this:
 - `*** python3 /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).

8.7.20. 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
```

8.7.21. 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
```

8.7.22. 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
```

8.7.23. 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)
```

8.7.24. 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)> <P4_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
```

8.7.25. 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 8.7.4, “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
```

8.7.26. 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
```

8.7.27. replica_status.sh

The `/p4/common/bin/replica_status.sh` script is regularly run by crontab on a replica or edge (using [Section 8.4.15, “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 `p4 pull -lrv` and `p4 pull -ls` commands 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
```

8.7.28. 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
```

8.7.29. rotate_journal.sh

The `/p4/common/bin/rotate_journal.sh` does a journal rotation and replay. Because journal rotations can only be initiated on the commit server, this script only operates on a commit server.

The script performs the following actions:

- Performs a journal rotation.

- Deletes the `offline_db_usable.txt` semaphore file to indicate the `offline_db` is being operated on and is not usable.
- Replays whatever numbered journal files are needed to make the `offline_db` current, up to but not including the live P4JOURNAL.
- Writes a new `offline_db_usable.txt` semaphore file to indicate the `offline_db` is healthy and ready to use.
- Rotates logs files and handles checkpoint, journal, and log retention according to settings `$KEEPLOGS`, `$KEEPJNLS`, and `$KEEPCKPS`.

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
```

8.7.30. 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 8.6.2, “pull.sh”](#)

Uses `fstat -0b` 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 8.7.31, “submit_test.sh”](#)

See the `/p4/common/bin/triggers/submit.sh` script for details and to customize for your environment.

8.7.31. submit_test.sh

The `/p4/common/bin/submit_test.sh` script is a test script.



THIS IS A TEST SCRIPT - it substitutes for [Section 8.7.30, “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>

8.7.32. sync_replica.sh

The `/p4/common/bin/sync_replica.sh` script is included in the standard crontab for a replica.

It runs an `rsync` to mirror the checkpoints directory from its upstream (P4TARGET) server to the local replica machine. This script detects whether the upstream server uses the First Form or Second Form of the `journalPrefix` and rsyncs the appropriate checkpoints directories (see [Appendix B, The journalPrefix Standard](#)).

It then uses the latest checkpoint in that directory and any subsequent numbered journals 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). It is also useful to have a current and maintained `offline_db` in the event that a failover to a standby occurs, to provide additional redundancy even post-failover to the new commit server.

The `keep_offline_db_current.sh` script is an alternative to `sync_replica.sh`, particularly useful if `rsync` is not available between the replica and its upstream server.

Usage

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

8.7.33. 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.

8.7.34. 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 `run_if_master.sh` script.

It ensures that all *regular* users accounts are added to the `limits` group, where means those users reported with `p4 users` (and excluding those accounts only listed with `p4 users -a`, such as replication service users). This makes it easy for an administrator to apply global limits defined in group specs, such as `MaxScanRows`, `MaxSearchResults`, `MaxMemory` and others to all regular users. You can select which settings you to apply; you can set some and leave others `unset`. In all cases, the idea is to select values that represent "crazy big" individual commands from unduly impacting the P4 Server.

Setting optimal values for any given environment is best done with testing and tuning, as there are a variety of factors affecting optimal values including hardware, data, and local workflows. In one environment, opening 50,000 files may be a clear sign of something done wrong, so setting `MaxOpenFiles` to 50000 might be reasonable. In another environment, opening 200,000 files may be routine.

Until limits are applied in the group spec of the `limits` group (e.g. with a `p4 group limits` command), membership in the group has no functional impact. When the group spec is updated to apply limits, the limits take immediate effect.

Before global limits are applied to the `limits` group, any users that should be exempted from limits should be added to the `no_limits` group. At a minimum, this should include the SDP admin P4USER (commonly named `perforce` or `p4admin`). Here is a sample procedure to create this group:

```
p4 --field Owners=$P4USER --field Users=$P4USER --field Description="Limits exemption
group." group -o no_limits | grep -v ^# | sed s/unset/unlimited/g >
$P4TMP/no_limits.group.p4s
p4 group -i < $P4TMP/no_limits.group.p4s
```

To help determine optimal values, the best practice is to do the following:

- Define which settings you intend to change in the group spec.

- Create a `test_limits` group with test values, and including test users. (DO NOT include the SDP admin P4USER.)
- Take time to refine values by testing various commands. The intent is only to block "crazy big" individual commands while not interfering with "big but allowed" actions people (and bots) need to do to do their work.

Before setting these values, please review further information on limits settings in group specs. See:

- [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 aware of this possibility is important for installing in various existing environments.

The behaviors of the `perforce-python3` package install vary slightly 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 or 3.9 exist on the machine when the `perforce-p4python3` package is installed, P4Python will be added to the existing Python 3.8/3.9 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 or 3.9, such as Python 3.6, installing the `perforce-p4python3` package will add a new Python 3.9 installation with the version of Python 3 it uses (e.g. `python3.9`), but it will **not** adjust the existing `python3` symlink. **The P4Python module will not be available by calling `python3`; `python3.9` must be specified.** You can at that point decide to manually adjust the `python3` symlink to point to `python3.9`, though

this has 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 `python3.9` specifically rather than just `python3`.

- In any case, avoid using `python2`, which refers to Python 2.7 on modern Linux.
- Using just `python` commonly refers to Python 2.7 on some Linux distros, and should not be referenced on such systems.
- On more recent distros in 2024, `python` without a version identifier now refers by default to Python 3.x rather than Python 2; Python 2 may not be installed at all. On such systems, confirm that the P4Python module is available by calling just `python` before using it.



Be aware that the Helix Core triggers table is centralized across the topology. The central triggers table must apply to all p4d servers, calling the correct version of Python (to include P4Python), across the fleet of p4d server machines. Within the triggers table, you can include a specific interpreter for each line in the triggers table, which effectively overrides the shebang line in the script. Alternately, you can avoid specifying the interpreter and call the script directly, in which case the shebang line in the given script applies. Whichever strategy you choose, ensure that it works for the p4d server fleet.

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`, and must also have P4Python installed. See: [Section 9.1, “Installing Python3 and P4Python”](#).

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 appropriate 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/
chmod +wx /p4/common/site/bin/triggers/CheckCaseTrigger.py
```

3. Inject a *shebang* line as the first line of the trigger script file `/p4/common/site/bin/CheckCaseTrigger.py`. This must reference the location of Python as you have installed it locally, and this Python must have the P4Python module available. The *shebang* might look like one of these examples:

- `#!/usr/bin/python3`

- `#!/usr/local/bin/python3`
- `#!/bin/env python3` ← This will search for Python in the PATH.

4. Test on an existing (small) changelist:

```
p4 changes -s submitted -m 9
```

pick a suitable changelist number, e.g. 1234

```
/p4/common/site/bin/triggers/CheckCaseTrigger.py 1234
```

5. Test that it works

- Add appropriate line to triggers table on the P4 Server:

```
CheckCaseTrigger change-submit //test/...
"/p4/common/site/bin/triggers/CheckCaseTrigger.py %changelist%"
```

- Create test workspace
- Submit simple `Test.txt`
- Attempt to submit `test.txt` and check for error

6. After testing, change the path field in the P4 triggers table to `//...`:

```
CheckCaseTrigger change-submit //...
"/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": "https://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. If it is up at the start of processing, it will be shutdown by the `recover_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 `offline_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 essence of the reseed operation.
- After the edge reseed is complete, the p4d process is started. It will then start replicating new data from the master since the time of the edge seed checkpoint creation. The p4d service may hang and be unresponsive for several minutes after it is started. If you choose to monitor closely, when a `p4 pull -lrv` 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/checkpoints.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 maintenance" 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 included in the SDP package/tarball, including documentation and sample scripts. A subset of these files are deployed to server machines during the installation process.

Figure 1 - SDP Package Directory Structure

```
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)
        Unsupported (folder containing unsupported extras)
```

A.1. SDP Classic and OS Package Structures

As the SDP evolves toward OS package installation, changes have been made to the directory structure. These changes to the SDP structure **have no effect** on SDP scripts.



Any custom scripts at your site reference the SDP logical path structure using symlinks starting with `/p4`, such as `/p4/common/bin`, will be unaffected by the eventual conversion to the SDP Package Structure. However, if any custom scripts reference physical mount points directly, e.g. referencing `/hxdepots/p4/common/bin`, such scripts will need to be adapted before converting to the Package Structure.

The new structure is currently available only for new installs on new machines using the `install_sdp.sh` script.

A brief timeline:

- November 2024: SDP OS Package Structure is available for new installations only using

`install_sdp.sh`. SDP upgrades (using `sdp_upgrade.sh`) process only the Classic structure.

- December 2024: SDP Upgrades (using `sdp_upgrade.sh`) is compatible with both SDP Classic and OS Package structures. Upgrading will not convert the structure.
- Future: At some point in the future, `sdp_upgrade.sh` may be updated to convert from the Classic to OS Package structure during upgrades.
- Future: Conversion to the OS Package Structure will be required prior installing the coming `helix-sdp` OS installation packages (not yet available).

The SDP OS Package Structure uses the `/opt/perforce/helix-sdp` directory tree to store all SDP. If you do not see this directory on your machine, then the Classic structure is in place.

Table 2. SDP Package Structure Highlights

Directory	Owner/Perms	Comments
<code>/opt/perforce/helix-sdp</code>	root:perforce/775	SDP Package Base
<code>/opt/perforce/helix-sdp/downloads</code>	perforce:perforce/755	Supersedes <code>/hxdepots/downloads</code> .
<code>/opt/perforce/helix-sdp/sdp</code>	root:root/755	Immutable SDP directory. Contains only the extracted SDP tarball. Updated only during SDP upgrades.
<code>/opt/perforce/helix-sdp/p4</code>	perforce:perforce/700	Writable SDP directory. Contains subset of files extracted from SDP tarball, but also includes other files such as: • Everything under <code>/p4/sdp</code> (superseding <code>/hxdepots/sdp</code>) • Everything under <code>/p4/common</code> (superseding <code>/hxdepots/p4/common</code>, including: ◦ Helix binaries, e.g. as <code>p4d_2024.1.2661979</code>. ◦ Various SDP symlinks such as <code>p4_1_bin</code>, <code>p4_2024.1_bin</code>. ◦ Various locally generated shell environment and configuration files. ◦ The <code>site</code> folder.

While a `helix-sdp` OS Package is not yet available, the new structure introduced in SDP 2024.1 was

defined to support future package installation. Traditional tarball installation will still be possible for UNIX and Linux distros for which OS packages are not available.

In SDP Legacy and Package structures, the SDP root directory on an installed system `/p4`, is a directory on the local OS root volume.

A.2. SDP Runtime Structure

The *application* administrator’s view of the system is illustrated here. This shows how to navigate the directory structure to find databases, log files, and versioned files in the depots. The following example is illustrated with `1` as the SDP instance name.

Figure 2 - SDP Runtime Structure



The following table explains some of what is found in the folders in this structure, and where files live in the underling mounted storage volumes.

Directory	Remarks
<code>/p4</code>	Must be under root (<code>/</code>) on the OS root volume
<code>/p4/1/bin</code>	Host local folder on the OS root volume. This is the SDP Instance Bin directory. The list of <code>p4*_init</code> scripts in the Instance Bin directory indicates which server types (p4d, p4broker, p4p, p4dtg) are expected to run on this machine.
<code>/p4/1/depots</code>	Versioned files are stored here.
<code>/p4/1/tmp</code>	This is used by p4d and various scripts as temp storage. The <code>\$P4TMP</code> variable points here.
<code>/p4/common/config</code>	Contains <code>p4_<instance>.vars</code> file, e.g. <code>p4_1.vars</code>
<code>/p4/common/bin</code>	Contains server binary files, symlinks, various SDP scripts, etc.
<code>/p4/common/etc</code>	Contains <code>init.d</code> and <code>cron.d</code> .

Directory	Remarks
<code>/p4/common/site</code>	Contains site-specific and custom files, such as custom triggers in <code>/p4/common/site/bin/triggers</code> . See Section A.2.1, “The Site Directory” .
<code>/p4/1/logs</code>	Contains P4JOURNAL and various application and script logs. The \$LOGS variable points here.
<code>/p4/1/root</code>	Contains live server databases. The \$P4ROOT variable points here.
<code>/p4/1/offline_db</code>	Contains offline copy of main server databases.
<code>/p4/1/root/save</code>	Used only during running of <code>refresh_P4ROOT_from_offline_db.sh</code> for extra redundancy.
<code>/p4/sdp</code>	Contains SDP files as extracted from a tarball.
<code>/p4/sdp/Version</code>	The SDP Version file. Cat this file to see the current SDP Version.

A.2.1. The Site Directory

The Site Directory, `/p4/common/site`, has a special purpose and usage. See [The Site Directory](#) for more information.

A.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.4. Storage Volumes Layout

This section describes storage volume layout for Helix Core Server (P4D), Helix Proxy, and a Helix Broker.

A.4.1. Storage Volumes for a Helix Core Server

The following table describes storage volume layout for a Helix Core Server (P4D):

Table 3. Storage Volumes

Name	Mount Point	Contents	Backup?	Comments
HxDepots	<i>/hxdepots</i>	Depots, the storage directories for archived content (by default a case-sensitive volume containing versioned files), Small SDP folders, backups of small SDP folders		
HxCheckpoints	<i>/hxcheckpoints</i>	Checkpoints (point-in-time snapshots of metadata) and numbered/completed metadata journal files.		
HxMetadata	<i>/hxmetadata or + /hxmetadata{1,2}</i>	P4ROOT, offline_db, server license files.	No	Never allow OS-level backup utilities such as tar or rsync to touch this volume directly. Backups via VM snapshots are OK.
HxLogs	<i>/hxlogs</i>	P4JOURNAL, P4LOG, structured logs, application and SDP script logs.	<i>Optional</i>	-
Root	<i>/</i>	OS, backups of small SDP folders, and <i>/opt/.</i> and other apps/scripts.	Yes	This is the OS root volume rather than a mount point. This contains scripts

A.4.2. More About HxDepots

Use a large volume with high capacity. If using RAID, use RAID 6 on its own controller with a standard amount of cache. Alternately use a SAN or NAS volume (NFS access is fine). SSD is fine if available as well.

This volume **must** be backed up, as it contains critical data needed for recovery.

The default mount point for this volume is */hxdepots*.

A.4.2.1. Using Multiple Depot Storage Volumes

If needed, additional storage volumes can be added. These might be named `/hxdepots-2`, `/hxdepots-3`, etc. The primary volume, generally the first one to exist, is the one pointed to by the `server.depot.root` configurable. The value is `/p4/N/depots`; this path is a symlink to `/hxdepots/p4/1/depots`. The primary should be called simply `/hxdepots`, not `/hxdepots-1`, even if multiple volumes are used, to visually distinguish it as the primary volume. So `/p4/N/depots` directory is your one-stop-shopping to find all versioned files in all depots, even though some are physically located on other volumes and symlink'd.

A key idea for using multiple storage depots is that *all* depots are always listed in the `server.depot.root` directory, i.e. `/p4/N/depots` (e.g. `/p4/1/depots`). Depots that physically exist on `/hxdepots` will be directories in that `server.depot.root` directory, while depots that exist on the other volumes will be symlink'd from there.



Generally, it is preferable to use a *single* volume for depots when practical. Using multiple volumes adds an element of configuration complexity to backup and recovery operations. It may also add maintenance tasks, such as moving depots from one volume to another.



The number of volumes used and the assignment of which depots are stored on which volumes is not absolutely required to be kept the same across a fleet of P4 server machines. However, keeping them consistent helps contain the complexity of using multiple volumes.



If you are using Infrastructure as Code (IaC) methodologies, such as Terraform, AWS CloudFormation Templates (CFT), or Azure Resource Manager (ARM) Template, you can account for the volume layout in the machine templates. Beware this will need to be maintained.

Some scenarios where it may be desirable to accept the increased complexity of multiple volumes include:

Table 4. Multiple Depot Storage Volumes

Scenario	Description	Sample Mounts
Increased Capacity	While it is usually possible (and preferred) to have an arbitrarily large storage volume (e.g. by using expandable storage, perhaps on NFS), some environments choose or are constrained to using multiple storage volumes as the path to increasing storage capacity.	<code>/hxdepots-2</code> <code>/hxdepots-3</code>

Scenario	Description	Sample Mounts
	Per-Depot Cost Accounting	Some customers prefer to assign depots to specific teams/projects, and then store those depots on separate volumes. This simplifies internal cost per-project (or per-team) storage cost accounting, as all assets for any given depot are associated to a given team or project. (Some shared depots may have costs split across teams).
<code>/hxdepots-ProjA</code> <code>/hxdepots-TeamAlpha</code>		Archival and Cost Management
Using different depots allows a form of static cost-tiering, as different mounts can have different underlying storage hardware.	<code>/hxdepots</code> (fast) <code>/hxdepots-cold</code> <code>/hxdepots-hdd</code>	

Example Using Multiple Depots

Say you have the following:

- ActiveProject depot on `/hxdepots`.
- Tools depot on `/hxdepots`.
- LegacyProject on `/hxdepots-hdd`

That would be set up like so:

- `/hxdepots` mount point
- `/hxdepots-hdd` mount point (say, hdd)

There would be these directories:

- `/hxdepots/p4/1/depots/ActiveProject`
- `/hxdepots/p4/1/depots/Tools`
- `/hxdepots-hdd/p4/1/depots/LegacyProject`

In `/hxdepots/p4/1/depots` (and thus `/p4/1/checkpoints`), there would be a symlink `LegacyProject` that points to `/hxdepots-hdd/p4/1/depots/LegacyProject`.

A.4.3. More About HxCheckpoints

This volume is not used in the default installation. If not used, checkpoints are stored on HxDepots.

If this optional volume is used, it must be backed up as it will contain critical data.

A.4.4. More About HxMetadata

Use a *fast* and *low latency* storage option, 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 $(2 \times (\text{size of db.* files}) + \text{room for growth})$ approaches or exceeds the storage capacity of the storage device used for metadata. 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.

A.4.5. More about HxLogs

Use 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 mounted as `/hxlogs` and can optionally be backed up. It contains application logs and the critical P4JOURNAL file that is continuously written during normal operation.

If a separate logs volume is not available, logs can be placed on the `/hxmetadata` or `/hxmetadata1` volume, as metadata and logs have similar performance needs that differ from `/hxdepots`. This is not ideal or recommended, but a reasonable compromise when working with physical hardware limitations.



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 SDP installations, the `/opt/perforce/helix-sdp` folder, with `/opt` typically on the OS root volume, contains SDP scripts, Helix Core binaries, and various configuration files and symlinks.

On all SDP machines, a `/p4` directory will exist containing a directory for each instance named `/p4/<instance>`. The volume layout (which maps logical names via links to the physical directory structure) 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

On a production Helix Core Server, typically all the `/hx*` directories are mounted volumes.

For an illustration of how to create format and mount storage volumes on an AWS server, see: [Perforce Helix Core Sample Storage Setup - AWS](#).

A.4.6. Storage Volumes for a Helix Proxy

The following table describes storage volume layout for a Helix Proxy (P4P):

Table 5. Storage Configuration for a Helix Proxy

Directory	Used For	Comments
<code>/</code>	OS root volume.	The <code>/hxlogs</code> may be stored here.
<code>/hxdepots</code>	Versioned file cache, application software.	Cache managed with <code>p4pcm.pl</code> to avoid filling up.
<code>/hxlogs</code>	Application logs.	Rotated regularly. Low utilization (10%) is typical and healthy to avoid service interruption on the busiest days.

On a production Helix Proxy, typically only the `/hxdepots` directory is a mounted volume; `/hxlogs` can be a directory on the OS root volume.

A.4.7. Storage Volumes for a Helix Broker

A Helix Broker (`p4broker`) has no data, only software, configuration files and logs. Thus a production Helix Broker typically uses only the OS root volume, or a small `/hxdepots` volume.

Table 6. Storage Configuration for a Helix Broker

Directory	Used For	Comments
<code>/</code>	OS root volume.	The <code>/hxlogs</code> and <code>/hxdepots</code> may be stored on the OS root volume for a broker.
<code>/hxdepots</code>	Broker config file, application software.	

Directory	Used For	Comments
/hxlogs	Application logs.	Rotated regularly. Low utilization (10%) is typical and healthy to avoid service interruption on the busiest days.

A.5. Memory and CPU

Make sure the server has enough memory (RAM) 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 actively used database files in memory.

The p4d process itself is frugal with system resources such as RAM. However, p4d 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.6. Case Insensitive P4D on UNIX/Linux

By default **p4d** is case sensitive on UNIX/Linux 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.jnl.320` and checkpoints would have names like `p4_1.ckp.320.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 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 replica's `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 replica's `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. 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.5. 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.6. Goals of the **journalPrefix** Standard

Some design goals of 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 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>
```

or, for the singular commit server in a data set:

```
{commit|master}[.<OrgName>[.<SDPInstance>]]
```

C.1.1. Commit Server Spec

The server spec name for a commit server starts with the literal token **commit** or **master**, followed by an optional organization tag name (separated by a dot), followed by an optional SDP instance name (separated by a dot).

The server spec name for a commit server is intended to be unique to enable certain cross-instance sharing workflows, e.g. using remote depots and Helix native DVCS features (e.g. **p4 fetch**, **p4 push**, etc.). The combination of `<SDPInstance>.<OrgName>` give a reasonable assurance of uniqueness (without resorting to GUIDs which aren't suitable as names that humans need to type frequently).

The `<SDPInstance>` and `<OrgName>` both have these characteristics:

The `<SDPInstance>` and `<OrgName>` tags can be any alphanumeric name. Underscores (`_`) and dashes (`-`) are also allowed. Dots, spaces, and other special characters are not.

The `<SDPInstance>` name is typed often in various admin operational tasks, so:

- Instance names are best kept short. A length of 1-5 characters is recommended, with a maximum of 32 characters.
- Lowercase letters are preferred and required at some sites, but not required by the SDP.

The `<OrgName>` is not typed often and can be longer. A length of 2-10 characters is recommended, with a maximum of 32 characters.

See [Section 2.1.2, "Instance"](#) for more information on an SDP Instance.



A good value for **auth.id** is **p4_[<OrgName>.<SDPInstance>]**. The **auth.id** must also be unique across servers that do any cross-server communication using remote depots and/or Helix native DVCS features, or even if client programs connect to

multiple servers. This avoids ticket stomping issues.



For more information about `auth.id`, see [Changing auth.id](#).

Sample values for the commit server:

- `master` - Simple, but does not guarantee uniqueness.
- `commit` - Simple, but does not guarantee uniqueness.
- `master.1` - Commit server for SDP instance 1.
- `commit.1` - Commit server for SDP instance 1.
- `commit.ExampleCo.fgs` - Commit server for SDP instance `fgs` for the organization ExampleCo.

Note that changing the server spec of a commit server can entail some work, as the `ReplicatingFrom` field of any server specs that target the commit server would need to be updated if it is ever changed. Also, changing the `auth.id` involves user impact and thus is best done with communication to users. For more information, see [Changing Commit ServerID](#).

C.1.2. 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](#)
- `swarm`: Helix Swarm

As a special case, the `HelixServerTag` is omitted for the `ServerID` of the master server spec.

C.1.3. Replica Type Tags

The `ReplicaType` is one of:

- `commit` or `master`: 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.1.3.1. 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.1.4. 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. `awsuse1` 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_awsuse1`: 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](#).

D.2. How do I change the super user password?

There are two critical accounts to be aware of:

- The UNIX/Linux operating system user account with a password managed by the operating system of the machine, referred to as the OSUSER.
- The Perforce application super user with a password in the Perforce database. The SDP standard shell environment sets P4USER to refer to the super user.

The user account name **perforce** is the default for both OSUSER and P4USER, but they each can have different values. The OSUSER applies to the server machine, while the P4USER can vary on a per-instance basis.



Some admins choose to use the same password for the **perforce** OSUSER and P4USER (for convenience and to reduce confusion), and then do routine rotations of both passwords (for enhanced security).



The P4 application super user should always use P4 password management, even if other accounts are configured to use LDAP, SSO, or some other authentication method.

To change the OSUSER, use your standard operating system commands. This may be the **passwd** command, but may be different depending on your operating system and other factors.

The following describes how to change the Perforce application super user password.

Step 1. Get a maintenance Window

Plan to do this work in a maintenance window. The procedure can cause disruption if any triggers or extensions rely on a valid ticket for your application super user. Also, much automation such as the SDP **daily_checkpoint.sh** script rely on having a valid ticket.



If you are fully aware of all the ways the password is used and thus the potential impacts, you can do the work outside of a maintenance window. Changing the password can disrupt triggers, extensions, and various automation, but will not have any impact on Helix Core service itself.

Step 2. Pick a Password

Select your new password. Depending on your local policy, you may manually create a password, generate one, and possibly store it in a vault of some kind, or perhaps use a password manager.

Step 3. Login as the OSUSER

Login as the OSUSER (e.g. **perforce**), and ensure that the standard SDP shell environment is set.



If the OSUSER shell environment files `~/.bash_profile` and `~/.bashrc` are set correctly, this step is done just by logging into the **perforce** OSUSER account.

Step 4. If you are using cleartext passwords, get the current password from the admin password file. The shell variable `$SDP_ADMIN_PASSWORD_FILE` contains the path to the password file for the current instance, something like `/p4/common/config/.p4passwd.p4_N.admin`. Do

```
cat $SDP_ADMIN_PASSWORD_FILE
```

Take note of the current/old password.

If you are not using cleartext passwords, this file will not exist. Instead, a file will exist with the same name but a `.enc` suffix will exist, containing an encoded password file. Decode the password like this:

```
base64 -d - < ${SDP_ADMIN_PASSWORD_FILE}.enc
```

Step 5. Put the new password in the admin password file.

If using cleartext passwords, store the password in the file `$SDP_ADMIN_PASSWORD_FILE`, e.g.:

```
echo 'YourNewPassword' > $SDP_ADMIN_PASSWORD_FILE
chmod 600 ${SDP_ADMIN_PASSWORD_FILE}.enc
```

If you are not using cleartext passwords, encode the password in the `.enc` file:

```
echo 'YourNewPassword' | base64 - > ${SDP_ADMIN_PASSWORD_FILE}.enc
chmod 600 ${SDP_ADMIN_PASSWORD_FILE}.enc
```

Step 6. Set the password in the P4 database.

Do:

```
p4 passwd
```

Provide the old once and the new password twice as prompted. The passwords will *not* be

displayed on the screen.

Step 7. Call the `p4login` script to exercise the new password file:

```
p4login -v
```

This script will use the encoded password file if available, otherwise it will fall back to use the cleartext file.

Confirm you have a valid ticket afterward with:

```
p4 login -s
```

Step 8. Copy the password file (either the cleartext or encoded form) to any and all replica and edge server machines.

Step 9. On each replica and edge, login as `perforce` and also do `p4login -v -service` and then `p4 login -s`.

D.3. How do I change from using cleartext to encoded passwords?

Support for encrypted passwords arrived with SDP 2025.1 Patch 1. However, in-place upgrades of SDP to 2025.1 Patch 1 do not change current behaviors on existing machines. Thus, cleartext passwords remain in place. Any previously documented local procedures will continue to work as before. However, you can choose to change to using encrypted passwords for the SDP super user.

The user account name `perforce` is the default for the P4USER, but it can have different values. The P4USER can vary on a per-instance basis.

See also: [Section D.2, “How do I change the super user password?”](#)

Step 1. Login as the OSUSER

Login as the OSUSER (e.g. `perforce`).

Step 2. Generate the encrypted password file.

```
base64 - < $SDP_ADMIN_PASSWORD_FILE > ${SDP_ADMIN_PASSWORD_FILE}.enc
```

Step 3. Remove the cleartext file.

```
rm -f $SDP_ADMIN_PASSWORD_FILE
```



As of SDP 2025.1 Patch 1, the `install_sdp.sh` script will install SDP on new

machines using encrypted passwords by default, though this behavior can be changed by using the `UseEncryptedPassword` setting in the `sdp_install.cfg` file. For more information, see [Appendix I, More Detail on install_sdp.sh](#).

D.4. Can I remove the perforce user?

No. This account is required for critical operations like checkpoints for backup.



This account need not occupy a licensed seat. Once a Helix Core server becomes licensed, you can fill out the [Helix Core Request for Background User](#) form to request up to 3 "background users" to support background automation tasks. This accounts for the `perforce` super user, a `swarm` user, and typically one named something like `builder` for automated builds.

D.5. Can I clone a VM to create a standby replica?

Yes, cloning a virtual machine (VM) of a Helix Core commit server is a great way to simplify the process of creating a standby replica of the commit server. Similarly, cloning an edge server is useful in creating a standby replica of the edge.

Cloning can be done with various technologies and in cloud and on-prem environments. For example, in AWS, creating an AMI of an EC2 instance (i.e. a virtual machine) is just different terminology for creating a clone of the virtual machine. Azure, GCP, and other clouds have similar concepts and capabilities, as do on-prem virtual infrastructure such as VMware ESX servers. Even non-virtual infrastructure tools exist for cloning bare metal server machines.

Nothing needs to change other than the `server.id` file whether the machine you're cloning is a commit server (to make a standby of the commit) or an edge (to make a standby of the edge). There is a slight SDP structure difference between an commit an an edge—an edge will have a `/hxdepots/p4/N/checkpoints.edge_SITE` directory and `/p4/N/checkpoints.edge_SITE` symlink to it. As long as you clone the machine that you're making a standby of, be it commit or edge, you'll have the correct structure on the standby.

While nothing should need to change, there are a few things to double check before initiating the cloning process:

- Check that the SDP Instance Vars file, `/p4/common/config/p4_N.vars` has correct values for **P4MASTERHOST** and **P4MASTER_ID**.
- The **P4MASTER_ID** must be the `server.id` of the commit server, always, and that will be the same regardless of what machine you're on. The **P4MASTERHOST** should be a DNS name for the commit server that works—i.e. that is valid to reference from the standby server after cloning. Using the same DNS name used by regular users is preferred—it can be an FQDN or a short name depending on how DNS is setup locally. If DNS isn't available in the server environment (as is sometimes the case), Plan B for setting **P4MASTERHOST** is to still use the same DNS that users know, but to add an `/etc/hosts` entry ("hack?") on the standby server machine after cloning so that the DNS name works on the standby to reference the commit server. Plan C, which we strongly advise against but do support, is to use an IP address for the

P4MASTERHOST value. Plan A is preferred because Plans B and C require the admin who executes failover to be aware of the "hacks" — `/etc/hosts` entry or using an IP address — to be accounted for in the failover procedure.

The general idea is that `/p4/common` structure in the SDP should be *common* across all Helix Core server machines in your fleet. Even on the standby replica, the **P4MASTER_ID** and **P4MASTERHOST** values be exactly the same as on the commit. Cloning the machine is the best way to do it. It's also nice to have a reasonably current set of archives, and nice to ensure all those little SDP config bits are correct.

Here is a sample procedure of cloning a machine to create a standby replica.

Step 1. Verify **P4MASTER_ID** and **P4MASTERHOST** settings are correct.

Step 2. Use `mkrep.sh` to create your standby server. See: [Section 5.3.4, "Using mkrep.sh"](#).

Step 3. Run `p4 admin journal`. (Digression: Use `p4 admin journal` command if you're creating a standby or unfiltered edge or replica, but use the `rotate_journal.sh` script instead if you're creating a filtered edge or filtered forwarding replica, where *filtered* here means using the `*DataFilter` fields in the server spec and/or using `-T` option to the configured `startup.N` thread that does the metadata pull for the ServerID of the new server.)

Step 4. Clone the VM.

Step 5. Start the new VM after the cloning operation. For example, if in AWS, launch an EC2 instance from the AMI.

Step 6. Stop the `p4d_N` (and `p4broker_N`) services if running.

Step 7. Use `hostname -I` to get the local/private IP, and request a new license file for that IP — but don't wait for it.

Step 8. Remove the `$P4ROOT/license` file.

Step 9. Remove the `$P4ROOT/server.id` file.

Step 10. Load the latest checkpoint and numbered journal, and then pull recent archives, e.g. with a command like this sample:

```
nohup load_checkpoint.sh /p4/1/checkpoints/p4_1.ckp.50.gz
/p4/1/checkpoints/p4_1.jnl.50 -s p4d_ha_bos -l -r -b -y -verify default < /dev/null >
/p4/1/logs/load.log 2>&1 &
```

That `load_checkpoint.sh` does the rest. It stops `p4d` and `p4broker` services (just in case you forgot), clears `P4ROOT`, moves `P4LOG` and `P4JOURNAL` aside if they exist (which they would after a cloning situation), puts the new correct `server.id` file in place, reloads from the latest checkpoint and numbered journal (that are sure to have the very latest data due to the `p4 admin journal` done above just before the cloning), does a `p4d -xu` (just in case it's needed, but shouldn't be in this situation), starts the service, and then kicks off a `p4 verify -t` command on all depots to pull over any missing files from the commit.



The above procedure is merely a sample. Certain details, such as the handling of license files, may vary from one site to another.

Appendix E: Troubleshooting Guide

This appendix lists problems sometimes encountered by SDP users, with guidance on how to analyze and resolve each issue.

Do not hesitate to contact p4consulting@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
 - A problem with state file
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
```

E.2.1. Resolution

1. 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
```

E.2.2. Make Replication Errors Visible

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.2.3. Remove state file

Files `state` and `statejcopy` can usually be removed - let the server work out its current state. If you want to know current journal counter for replica:

```
p4d -r /p4/1/root -k db.counters -jd - 2>/dev/null | grep @journal@ | cut -d '@' -f 8
```

If there is a problem with being able to pull over an old journal which no longer exists on the master you may need to reseed the replica!

```
sudo systemctl stop p4d_1
cd /p4/1/root
mv state* save/
cd /p4/1/logs
```

```
[[ -d save ]] || mkdir save          # Create if doesn't exist
mv journal* save/
sudo systemctl start p4d_1
```

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 -ltr /p4/1/checkpoints.<suffix>/*.ckp.*
```

Also you can check for edge being locked (the following may hang):

```
p4 monitor show -al
```

- Then replay the newly created edge checkpoint 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 8.4.11, “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 8.7.32, “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 crontab 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.

E.8. Recovering from an interrupted checkpoint (causes replication to fail)

If you interrupt a checkpoint (e.g. by killing the process, or **Ctrl+C**), then replication to replicas and edges may fail due to a missing (rotated) journal - and what appears to be a "double journal" (meaning that one journal has the contents of two!). Normally, when a checkpoint operation completes (with **p4d -jc** or **p4 admin checkpoint**), the live journal is rotated, and a new journal started. This ensures that replicas and edges can continue to replicate from the correct journal.

The same problem can occur if the user attempts a journal rotation (via **p4 admin journal** or **p4d -jj**) and then aborts it (this scenario is less likely since journal rotation is usually pretty fast).

Note that this problem affects **all replication** and also **updating of the offline_db!**

E.8.1. Resolution

If the last rotated journal is **jnl.128**, then the current journal counter (from **p4 counter journal**) will show **129**. After a successful checkpoint, the live journal will be rotated to **jnl.129** and the journal counter will be updated to **130**.

To prevent journals being applied in the wrong order, correctly rotated journals will have a **@vv@** record ("verify value") within a few lines of the start of the journal, and then at the end of the journal there will be an **@rv@** record (replace value) updating the journal counter to the next sequential number.

So you would see something like this in **jnl.128**:

```
$ grep @journal@ p4_1.jnl.128
@vv@ 1 @db.counters@ @journal@ 128
@rv@ 1 @db.counters@ @journal@ 129
```

If we assume that the last completed journal before the interrupted checkpoint was **jnl.128**, then after the interrupted checkpoint, the live journal will still be **jnl.130** - but the replicas and edges will be expecting to see a rotated **jnl.129** (which does not exist, as the checkpoint was interrupted).

After the interruption, and the next journal rotation (e.g. by running **p4 admin journal**), you will end up with **jnl.130** which included the contents of **jnl.129**! Replication will show errors like:

```
Could not find journal file /p4/1/checkpoints/p4_1.jnl.129!
```

This is an example of what you would see in `jnl.130`:

```
$ grep @journal@ p4_1.jnl.130
@vv@ 1 @db.counters@ @journal@ 129
@rv@ 1 @db.counters@ @journal@ 130
@rv@ 1 @db.counters@ @journal@ 131
```

The above shows 2 journals in one - `jnl.129` and `jnl.130`.

Do a `grep -n` for the above and "split" the file to fix the problem, e.g. what you want to achieve is this:

```
$ grep @journal@ p4_1.jnl.129
@vv@ 1 @db.counters@ @journal@ 129
@rv@ 1 @db.counters@ @journal@ 130

$ grep @journal@ p4_1.jnl.130
@vv@ 1 @db.counters@ @journal@ 130
@rv@ 1 @db.counters@ @journal@ 131
```

Please note - it is not strictly required to have an `@vv@` line, but it is good practice (to avoid confusion and potential replaying of journal files in the wrong order). The critical part is to have the `@rv@` lines correct.

Assuming instance 1:

```
cd /p4/1/checkpoints
mv p4_1.jnl.130 p4_1.jnl.130.broken
grep -n '@journal@' p4_1.jnl.130.broken
```

Suppose this shows:

```
2:@vv@ 1 @db.counters@ @journal@ 129
98123500:@rv@ 1 @db.counters@ @journal@ 130
98123599:@rv@ 1 @db.counters@ @journal@ 131
```

It is most likely that the final 2 lines are near the end of the file. So we can split the file at line 98123500 using the output above (which gives us the first 98123500 lines in one file `parts_aa`, and the rest in another `parts_ab` - check the web for use of the `split` command if unsure):

```
split -l 98123500 p4_1.jnl.130.broken parts_
cp parts_aa p4_1.jnl.129
```

```
echo "@vv@ 1 @db.counters@ @journal@ 130" > p4_1.jnl.130
cat parts_ab >> p4_1.jnl.130
```

Ensure we have the correct journal counters:

```
$ grep @journal@ p4_1.jnl.129
@vv@ 1 @db.counters@ @journal@ 129
@rv@ 1 @db.counters@ @journal@ 130

$ grep @journal@ p4_1.jnl.130
@vv@ 1 @db.counters@ @journal@ 130
@rv@ 1 @db.counters@ @journal@ 131
```

Once the above has been done, you will most likely find that replicas will retry to pull the missing journal, and will seamlessly start working! However "standby" replicas with **journalcopy** thread, may have already copied over incorrect journals. The easiest thing to do is to reset them:

```
# Login to standby or forwarding-standby server (and again assuming instance 1)
sudo systemctl stop p4d_1
cd /p4/1/root
mv state statejcopy save/
cd /p4/1/logs
mkdir -p save
mv journal* save/

# Now remove any incorrectly copied journals
cd /p4/1/checkpoints
rm p4_1.jnl.129 p4_1.jnl.13*

# And restart the service
sudo systemctl start p4d_1
sudo systemctl status p4d_1
```

Give it a minute or two and check replication status: [Section E.2, “Replication appears to be stalled”](#)

E.9. Server won't start: Error: DB check with 'p4d -xvU' failed. Database integrity is in question.

This can happen if a server is shut down ungracefully, and the database is left in an inconsistent state. The normal SDP startup script will check the database integrity before starting the server, and if it fails, the server will not start.



We have seen this happen with AWS instances where the instance was restarted, possible without a graceful shutdown for the server. Recommend to customer to be careful!

The error message will look like this:

```
Error: DB check with 'p4d -xvU' failed. Database integrity is in question.
Please Contact Perforce Support (support-helix-core@perforce.com<mailto:support-helix-core@perforce.com>). The force_start option
is available, but not recommended. Review the bottom of this log:
/p4/1/logs/p4d_init.log
```

If you look in the log you may see something like this:

```
:
Validating db.have
Validating db.label
Validating db.locks
      **** UnlockCount=1 **** possible server crash
Validating db.locksg
Validating db.excl
:
Validating db.revfs
Validating db.working
      **** UnlockCount=1 **** possible server crash
Validating db.workingx
Validating db.workingg
:
```

So you will have errors for one or more tables.

E.9.1. Resolution

This will depend somewhat on how big the tables are that have problems. In an SDP world, your `offline_db` will have a consistent set of tables, restored every 24 hours from the latest checkpoint - we can use those files to restore the tables on the `P4ROOT` dir, but first ensuring that the live journal has been replayed.

The process for small corrupted tables is:

1. Rotate the live journal: `p4d -jj`
2. Replay that journal to your `offline_db`

```
# Confirm which journal has just been rotated
ls -ltrh /p4/1/checkpoints/ | tail

cd /p4/1/offline_db
p4d -r . -vtrack=0 -jd - db.counters | grep @journal@

# Assume the above says 1234
p4d -r . -vtrack=0 -jr /p4/1/checkpoints/p4_1.jnl.1234
```

3. Copy over corrupted tables from the `offline_db` to the `P4ROOT` dir, and restart the server:

```
# For each table which is corrupt
cd /p4/1/root
# Save current versions just in case
mv db.{locks,working} save/
# Copy over the
rsync -avP /p4/1/offline_db/db.{locks,working} .
```

Alternative to Copying corrupted tables: Swap Symlinks If the tables are big, you may wish to use an alternate approach, swapping symlinks rather than copying files. *However*, if you have partitioned clients AND they are stored under `P4ROOT`, you'll also need to copy the directory tree of partitioned clients if you swap symlinks, which could be slower than copying large tables.

If you have partitioned clients *not* stored under `P4ROOT`, then you can skip down to the symlink swapping block below.

If you have partitioned clients *and* they are stored under `P4ROOT`, then you'll need to `rsync` that tree, something like this:

```
rsync -a --delete /p4/1/root/client.readonly.dir/ /p4/1/offline/client.readonly.dir
```

With `offline_db` made current by replaying the journal, instead of copying the `db.*` files, symlink swapping looks like this:

```
cd /p4/1
[[ -r root/license ]] && cp -f root/license offline_db/.
[[ -r root/server.id ]] && cp -f root/server.id offline_db/.
for f in root/state*; do cp -f $f offline_db/.; done
L1=$(readlink root)
L2=$(readlink offline_db)
echo Swapping root to $L2 and offline_db to $L1.
rm root offline_db
ln -s $L1 offline_db
ln -s $L2 root
```

After either the db copy or the symlink swap is complete, restart the service:

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

You may also want to review [Section F.2, “Force Start Mechanism for P4D”](#).

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 `p4d_init.log` file contains a running history of service start and stop operations.

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 `p4*` services, effectively making `sudo` access required for the correct OS user (typically `perforce`). During installation the `install_sdp.sh` script takes care of providing enabling the needed sudoers entries by making a call to `gen_sudoers.sh`. For more information on sudoers, see: [Section 8.7.9, “gen_sudoers.sh”](#).

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. Force Start Mechanism for P4D

Just before `p4d` startup, the init script that starts the `p4d` service does some metadata safety checks:

- It does a `p4d -xvU` check in the `P4ROOT` directory. This is a simple and fast check to see if any database table (*.db) files are in the open state. On a normal clean shutdown, all tables are closed, and so they should normally be closed for a startup. If not, this metadata safety check fails. This could be caused, for example, by a sudden power loss.
- If the file `P4ROOT_not_usable.txt` exists in the `P4ROOT` directory, this indicates the databases are not safe to use because some kind of processing, such as loading a checkpoint, was incomplete at the time the service was requested to start. If this file exists, this metadata safety check fails.

If the normal service 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 `p4d_init.log` file contains a running history of service start and stop operations, and will contain the `p4d -xvU` check results if that failed.

If either metadata safety check fails, contact Perforce Technical Support (support-p4@perforce.com) to get help investigating the cause of the issue and the extent of the database damage. Then determine a course of action.

For more information on this specific error, see [Section E.9, “Server won’t start: Error: DB check with 'p4d -xvU' failed. Database integrity is in question.”](#).

It is not advisable to start the service under these circumstances, so the p4d service does not start by default if any of the metadata safety checks fail. However, if you decide to start the service despite the failures, it is possible using the "force start" mechanism.



Force starting should *not* be your first course of action, and should only be done after an evaluation of the situation and possibly a root cause analysis.

To get the service to start despite failed metadata safety checks, touch a file named `p4d_force_start` in the Instance Bin directory, and then start the service normally. For example, if using systemd, it might look like this for instance 1:

```
touch /p4/1/bin/p4d_force_start
sudo systemctl start p4d_1
```

After the service is started, the `p4d_force_start` file should be removed:

```
rm -f /p4/1/bin/p4d_force_start
```

It is not a good idea to use this feature routinely. Force starting may be useful to help understand the damage.

General advice related to force starts:

- If the `P4ROOT_not_usable.txt`, *never* do a force start. Consider a metadata recovery option such as using the `offline_db` or a checkpoint a checkpoint replay.
- If the `p4d -xvU` check shows multiple damaged tables, do not consider a force start. Consider a metadata recovery option such as using the `offline_db` or a checkpoint a checkpoint replay.
- If the `p4d -xvU` check shows only a single database table file was corrupted and that table is `db.have`, this is a scenario where you might consider doing a `force_start`. A metadata recovery option such as using the `offline_db` or a checkpoint a checkpoint replay may be a better option, but in this scenario a `force_start` may allow you to operate with an "acceptable" amount of data damage. In an extreme example, there might be just one corrupted workspace, and that might not be sufficient cause to keep the service offline for potentially many users.

This "force start" mechanism works whether the systemd or SysV init mechanism is used.



If you are force starting to evaluate data damage, consider taking steps to lock out users to prevent new user activity if there is uncertainty about whether a recovery may be necessary. For example, see [Section 4.2.2.1.4, “Configuring systemd](#)

[p4broker - multiple configs](#)".

F.3. Journal Corruption Check

Just before p4d startup, the init script that starts the p4d service does some a check for possible journal corruption.

The check involves replaying the last 10,000 lines of the active P4JOURNAL file into a temp directory to determine if they replay successfully. If this check fails, an extra journal rotation is done before the service is started. The intent is of the journal rotation is to increase the likelihood that any journal corruption appears at the tail end of a journal file rather than somewhere in the middle of a potentially large journal file. This helps reduce complexity of remediation strategies.

If this happens, you will see an indication in `p4d_init.log`. Contact Perforce Technical Support (support-p4@perforce.com) to get help investigating the cause of the issue and the extent of the journal corruption. If you have any replicas or edge servers downstream of the current p4d server (or if this is the commit server), use `p4 servers -J` to determine if those services are able to replicate properly. Journal corruption can interrupt replication.

F.4. 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/p4d_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 `/p4/common/bin/sdp_health_check.sh` script. This script is included with the SDP (starting with SDP 2023.1 Patch 3). If your installation does not have this script, it can be downloaded separately. Every version of the `sdp_health_check.sh` script can be used any and all versions of the UNIX/Linux SDP dating back to 2007, so you don't need to be concerned with version compatibility.

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 `performer` or `p4admin`):

```
cd /p4/common/bin
```

```
[[ -e sdp_health_check.sh ]] && mv -f sdp_health_check.sh  
sdp_health_check.sh.moved.$(date +%Y-%m-%d-%H%M%S')
```

```
curl -L -O  
https://workshop.perforce.com/download/guest/perforce_software/sdp/tools/sdp_health_ch  
eck.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 does not transfer sensitive information.
- It works for any and all UNIX/Linux SDP version since 2007.

Appendix I: More Detail on install_sdp.sh

I.1. Sample configuration file `sdp_install.cfg`

For use with `install_sdp.sh`:

```
#-----
# Config file for install_sdp.sh v5.17.2.
#-----
# This file is in bash shell script syntax.
# Note: Avoid spaces before and after the '=' sign.

# For demo and training installations, usually all defaults in this file
# are fine.

# For Proof of Concept (PoC) installation, Section 1 (Localization) settings
# should all be changed to local values. Some settings in Section 2 (Data
# Specific) might also be changed.

# Changing settings in Section 3 (Deep Customization) is generally
# discouraged unless necessary when bootstrapping a production installation or
# a high-realism PoC.

#-----
# Section 1: Localization
#-----
# Changing all these is typical and expected, even for PoC installations.

# Specify email server for the p4review script. Ignore if Helix Swarm is used.
SMTPServer=smtp.p4demo.com

# Specify an email address to receive updates from admin scripts. This may be
# a distribution list or comma-separated list of addresses (with no spaces).
P4AdminList=P4AdminList@p4demo.com

# Specify an email address from which emails from admin scripts are sent.
# This must be a single email address.
MailFrom=P4Admin@p4demo.com

# Specify the DNS alias to refer to the commit server, e.g. by end
# users. This might be something like 'perforce.example.com' or
# simply 'perforce', but probably not an actual host name like
# 'perforce-01', which would be known only to admins. The default value,
# localhost, is valid only for a single server topology.
DNS_name_of_master_server=localhost

# Specify a geographic site tag for the master server location,
# e.g. 'bos' for Boston, MA, USA.
SiteTag=bos
```

```
# Specify the hostname. This can be left blank. If set on a system that supports
# the 'hostnamectl' utility, that utility will be used to set the hostname. If the
# command line parameter '-H <hostname>' is used, that will override this setting.
Hostname=
```

```
# Specify the timezone. This can be left blank. If set on a system that supports
# the 'timedatectl' utility, that utility will be used to set the timezone. If the
# command line parameter '-T <timezone>' is used, that will override this setting.
Timezone=
```

```
#-----
```

```
# Section 2: Data Specific
```

```
#-----
```

```
# These settings can be changed to desired values, though default values are
# preferred for demo installations.
```

```
# Specify the TCP port for p4d to listen on. Typically this is 1999 if
# p4broker is used, or 1666 if only p4d is used.
P4_PORT=1999
```

```
# Specify the TCP port for p4broker to listen on. Must be different
# from the P4_PORT.
P4BROKER_PORT=1666
```

```
# Specify SDP instance name, e.g. '1' for /p4/1.
Instance=1
```

```
# Helix Core case sensitivity, '1' (sensitive) or '0' (insensitive). If
# data from a checkpoint is to be migrated into this instance, set this
# CaseSensitive value to match the case handling of the incoming data set
# (as shown with 'p4 info').
CaseSensitive=1
```

```
# If SSL (Secure Sockets Layer) encryption is to be used, specify the prefix,
# typically 'ssl:'. Leave blank if not using SSL.
SSLPrefix=ssl:
```

```
# Set the P4USER value for the Perforce super user.
P4USER=perforce
```

```
# Set the password for the super user (see P4USER). If using this Helix Installer to
# bootstrap a production installation, replace this default password with your own.
Password=F@stSCM!
```

```
# Set to 1 to use encoded password files, 0 for cleartext.
UseEncryptedPassword=1
```

```
# Set to 1 to include broker with p4d, 0 for p4d only. If set to 0 and if
# using port numbers 1666 and 1999 for p4broker and p4d, consider moving p4d to
# 1666 and the unused p4broker to 1999.
```

```
DeployBrokerWithP4D=1
```

```
# Specify '1' to avoid sending email from admin scripts, or 0 to send
# email from admin scripts.
SimulateEmail=1
```

```
# Specify a ServerID value. If left blank for a master/commit server, a sensible
# default value will be assigned. See the Server Spec Naming Standard:
#
https://swarm.workshop.perforce.com/view/guest/perforce\_software/sdp/main/doc/SDP\_Guide.Unix.html#\_server\_spec\_naming\_standard
ServerID=
```

```
# Specify the type of server. Valid values are:
# * p4d_commit - A master/commit p4d server.
# * p4d_master - A synonym for p4d_commit.
# * p4d_replica - Any type of p4d replica (except edge) with all metadata from the
# master, not filtered in
#   any way. May or may not be forwarding or forwarding-standby.
# * p4d_filtered_replica - A filtered replica or filtered forwarding replica.
# * p4d_edge - An edge server.
# * p4d_edge_replica - Replica of an edge server. Also set TargetServerID.
# * p4broker - An SDP host running only a p4broker, e.g. as standalone broker
#   possibly deployed in a DMZ.
# * p4p - An SDP host running only a p4p.
# * p4proxy - A synonym for p4p.
#
# The ServerID must also be set if the ServerType is any p4d_* type other than
# 'p4d_commit' or 'p4d_master'.
ServerType=p4d_commit
```

```
# Set only if ServerType is p4d_edge_replica. The value is the ServerID of
# edge server that this server is a replica of, and must match the
# 'ReplicatingFrom:' field of the server spec.
TargetServerID=
```

```
# Specify the target port for a p4p or p4broker.
TargetPort=
```

```
# Specify the listening port for a p4p or p4broker.
ListenPort=
```

```
#-----
# Section 3: Deep Customization
#-----
# Changing these settings is gently discouraged, but may be necessary for
# bootstrapping some production environments with hard-to-change default values
# for settings such as OSUSER, OSGROUP, Hx*, etc.
#
# Changing these settings is gently discouraged because changing these values
# will cause the configuration to be out of alignment with documentation and
```

```

# sample instructions for settings that are typically left as defaults.
# However, there are no functional limitations to changing these settings.

# Specify the Linux Operating System account under which p4d and other Helix
# services will run as. This user will be created if it does not exist. If
# created, the password will match that of the P4USER.
OSUSER=perforce

# Specify the primary group for the Linux Operating System account specified
# as OSUSER.
OSGROUP=perforce

#Specify a comma-delimited list of any additional groups the OSUSER to be
# created should be in. This is passed to the 'useradd' command the '-G'
# flag. These groups must already exist.
OSUSER_ADDITIONAL_GROUPS=

# Specify home directory of the Linux account under which p4d and other Helix
# services will run as, and the group, in the form <user>:<group>. This user
# and group will be created if they do not exist.
OSUSER_HOME=/home/perforce

# Set HomeDirBackupMode to indicate whether the home directory of the OSUSER is
# to be backed up in such a way that it can be recovered with the recover
# script generated by the opt_perforce_sdp_backup service. Valid values are:
# * Off: The OSUSER home directory is not backed up.
# * Basic: Only key files in the OSUSER home directory are backed up, such as:
#   - .profile*
#   - .bash*
#   - .p4*
#   - .ssh*
# * Full: The entire OSUSER home directory is backed.
# When 'Full' is chosen, the backup is done using 'rsync -a' with '--delete',
# so files removed from the user home directory on any given day are removed
# from the backup on the next run. WARNING: This may consume disk space if
# large files are stored in the OSUSER home directory. To change this setting
# after installation, see the file .p4-sdp.home_dir_backup in the home
# directory of the OSUSER.
HomeDirBackupMode=Full

# The version of Perforce Helix binaries to be downloaded: p4, p4d, p4broker, and p4p.
P4BinRel=r25.2

# The following Hx* settings reference directories that store Perforce
# Helix data. If configuring for optimal performance and scalability,
# these folders can be mount points for storage volumes. If so, they must
# be mounted prior to running the install_sdp.sh script (other than to generate
# this configuration file).
#
# See the Server Deployment Package (SDP) for information and guidance on
# provisioning these volumes.

```

```
# Define the directory that stores critical digital assets that must be
# backed up, including contents of submitted and shelved versioned files.
HxDepots=/hxdepots

# Define the directory that stores critical digital assets that must be
# backed up, including metadata checkpoints and numbered journal files. If set to the
# same value as HxDepots, all critical assets to be backed will be on a single volume.
HxCheckpoints=/hxdepots

# Define the directory used to store the active journal (P4JOURNAL) and
# various logs.
HxLogs=/hxlogs

# The /HxMetadata1 and /HxMetadata2 settings define two interchangeable
# directories that store either active/live metadata databases (P4ROOT) or
# offline copies of the same (offline_db). These typically point to the same
# directory. Pointing them to the same directory simplifies infrastructure
# and enables the fastest recovery options. Using multiple metadata volumes
# is typically done when forced to due to capacity limitations for metadata
# on a single volume, or to provide operational survivability of the host in
# event of loss of a single metadata volume.
HxMetadata1=/hxmetadata
HxMetadata2=/hxmetadata
```

I.2. install_sdp.sh

USAGE for install_sdp.sh v5.17.2:

To Install a Helix Core P4D Server (with optional broker):

```
install_sdp.sh {-init|-empty|-sampledepot|-ued} [-demo] [-c <cfg>] [-no_cron] [-no_ppr] [-no_systemd|-no_enable] [-no_firewall] [-no_sudo|{-limited_sudo|-full_sudo}] [-v] [-no_pkgs|-extra_pkgs] [-s <ServerID>] [-si <SDPInstance>] [-ts <TargetServerID>] [-se] [-H <hostname>] [-T <timezone>] [-local] [-sdp_dir <sdp_dir>] [-d|-D]
```

To Install a standalone Helix Proxy:

```
install_sdp.sh -t p4p [-c <cfg>] [-ued] [-no_cron] [-no_ppr] [-no_systemd|-no_enable] [-no_firewall] [-no_sudo|-limited_sudo|-full_sudo] [-v] [-no_pkgs|-extra_pkgs] [-s <ServerID>] [-si <SDPInstance>] [-ts <TargetServerID>] [-tp <TargetPort>] [-lp <ListenPort>] [-se] [-H <hostname>] [-T <timezone>] [-local] [-sdp_dir <sdp_dir>] [-d|-D]
```

To Install a standalone Helix Broker:

```
install_sdp.sh -t p4broker [-c <cfg>] [-no_cron] [-no_ppr] [-no_systemd|-no_enable] [-no_firewall] [-no_sudo|-limited_sudo|-full_sudo] [-v] [-no_pkgs|-extra_pkgs] [-s <ServerID>] [-si <SDPInstance>] [-ts <TargetServerID>] [-tp <TargetPort>] [-lp
```

```
<ListenPort>] [-se] [-H <hostname>] [-T <timezone>] [-local] [-sdp_dir <sdp_dir>] [-d|-D]
```

or

```
install_sdp.sh -C > sdp_install.cfg
```

or

```
install_sdp.sh [-h|-man]
```

DESCRIPTION:

This script simplifies the process of installing Perforce Helix with the Server Deployment Package (SDP) on a fresh, new server machine.

If you are adding a new SDP instance to a machine that already has SDP installed, use the `makedirs.sh` script for that purpose. See: `makedirs.sh -man`

If you are unsure if SDP is installed, see if there is a `/p4` directory on the machine. If that directory exists, then SDP is already installed, and this `install_sdp.sh` script will refuse to operate on the machine.

This script can be used to install any of the following:

- * A Helix Core Server (p4d commit, standby, edge, etc.) with an optional p4broker.
- * A standalone Helix Broker (p4broker).
- * A standalone Helix Proxy (p4p).

If installing a p4d server, there are four options, one of which must be specified:

- * Use `'-init'` to initialize a new data set using the `configure_new_server.sh` script to get

started with various best practices, and create an initial checkpoint.

- * Use `'-empty'` to leave the unconfigured and ready for configuration.

- * Use `'-sampledepot'` to install the Sample Depot training data set. This is helpful when

bootstrapping a training or demo server.

Use `'-ued'` to use existing (presumably mounted) data volumes.

The following SDP structure is initialized:

`/opt/perforce/helix-sdp/sdp` (root owned, immutable except by SDP upgrades).

`/opt/perforce/helix-sdp/p4/sdp` (owned and writable by OSUSER).

`/opt/perforce/helix-sdp/downloads` (owned and writable by OSUSER).

`/opt/perforce/helix-sdp/p4/sdp/helix_binaries` (owned and writable by OSUSER).

This script handles many aspects of installation. It does the following:

- * Creates the operating system user (OSUSER) that Helix Core processes (p4d, p4broker, and p4 p4p) will run as. The default OSUSER is 'perforce'. The 'useradd' command is used to create the user as a local account on the machine,

and a password. OSUSER creation and password setting is skipped if that account already exists. If a non-local network account is to be used, that must be created first before running this script.

- * Creates the home directory for the OSUSER user, if needed.

Following installation, it also does the following to be more convenient for demos, and also give a more production-like feel:

- * Grants the perforce user sudo access (full or limited).
- * Creates default ~perforce/.bash_profile and .bashrc files.
- * Connects to the Perforce Package Repository (APT and YUM only).
- * Installs SDP crontab for the perforce OSUSER.

This script calls makedirs.sh for additional configuration:

- * Systemd service files are enabled (or SysV on older systems).
- * Firewall ports are opened for installed services.

This script is intended only for initial installation on a new server machine. For safety, it will refuse to operate if it detects any existing SDP directory structures. If installed on a machine where Helix Core data exists in non-SDP structures, it will not interact with the existing data. SDP structures include anything in or under the following directories:

- * /p4
- * /opt/perforce/helix-sdp (unless '-local' is used)
- * /opt/perforce/helix-sdp/sdp and /opt/perforce/helix-sdp/p4/sdp (if '-local' is used)
- * Mount points as configured for checkpoints, depots, logs, metadata, (unless '-ued' is specified to use existing data volumes), for example:
 - /hxcheckpoints
 - /hxdepots
 - /hxlogs
 - /hxmetadata[1,2]

PLATFORM SUPPORT:

This script is intended to work on a variety of Linux distributions and Linux server machines. The following are prioritized for support:

- * Ubuntu 24.04, 22.04, and 20.04. For Ubuntu, only even-numbered *.04 releases are supported).
- * Rocky Linux 9, 8
- * Red Hat Enterprise Linux (RHEL) 9, 8
- * SuSE 15.

This script recognizes SysV, Systemd.

This script requires bash 4.x+ and works with bash 5.x.

This script is not supported on Mac OSX. It recognizes the Launchd init mechanism on Mac but does not support it.

OS PACKAGES:

The following OS packages are installed (unless '-no_pkgs' is used):

* Yum: bc cronie curl elfutils elfutils-libelf elfutils-libs file gawk gdb lsof make nano neovim net-tools openssl openssl-devel policycoreutils-python-utils procs-ng rsync screen sos sysstat systemd-coredump tar tmux tuned wget which zlib zlib-devel

* AptGet: bc cron curl elfutils file gawk gdb libbz2-dev libc6-dbg libncurses5-dev libreadline-dev libsqlite3-dev libssl-dev llvm lsof make nano neovim net-tools policycoreutils-python-utils procs rsync screen sosreport sysstat systemd-coredump tar tmux tuned wget zlib1g-dev

* Zypper: bc cronie curl file gawk lsof make nano neovim net-tools openssl openssl-devel procs rsync screen sos sysstat tar tmux tuned wget which zlib zlib-devel

If '-extra_pkgs' is used, the following packages are installed in addition to those listed above:

* Yum: gcc gcc-c++ glibc-debuginfo glibc-debuginfo-common

* AptGet: build-essential

* Zypper: gcc gcc-c++

Development utilities such as 'make', the 'gcc' compiler, and 'curl' will be available '-extra_pkgs' is used.

In addition, if the Perforce Package Repository is added, these additional packages are installed:

* Yum:

* AptGet:

* Zypper: None, as the Perforce Package Repository does not support the Zypper package management system (e.g. as used on SuSE Linux).

OPTIONS:

-c <cfg>

Specify a config file. By default, values for various settings such as the email to send script logs to are configured with demo values, e.g. P4AdminList@p4demo.com. Optionally, you can specify a config file to define your own values.

For details on what settings you can define in this way, run:
install_sdp.sh -C > sdp_install.cfg

Then modify the generated config file sdp_install.cfg as desired.

The generated config file contains documentation on settings and values. If no changes are made to the generated file, running with '-c sdp_install.cfg' is the equivalent of running without using '-c' at all.

-C See '-c <cfg>' above.

-ued

Specify '-ued' to allow using existing data volumes. By default, this script aborts immediately if anything exists under the configured mount point directories. Use '-ued' in Infrastructure as Code (IaC) installs where data directories may exist from a machine template.

This script always aborts if any of the key directories exist at the start of other processing

This option is valid with '-empty', and mutually exclusive with '-init' and '-sampledepot' because no data changes are made if using existing data volumes.

-init

Specify '-init' to initialize a new Helix Core data set with the configure_new_server.sh script, which applies best practices for a production server installation.

One of '-empty', '-init', or '-sampledepot' is required if installing one of the p4d* server types.

This option is mutually exclusive with '-ued'.

-empty

Specify '-empty' to avoid initialization of a Helix Core data set. No db.* files will be created.

This option should be used if you intend to load a checkpoint created elsewhere on this new server machine, as would be done if you are installing a replica or edge server.

One of '-empty', '-init', or '-sampledepot' is required if installing one of the p4d* server types.

This option is compatible and implied by '-ued'.

-sampledepot

Specify '-sampledepot' to load the Perforce Sample Depot training/demo data set.

One of '-empty', '-init', or '-sampledepot' is required if installing one of the p4d* server types.

This option is mutually exclusive with '-ued'.

-demo

By default, key SDP storage volumes are verified to not appear on the OS root volume. If this is not the case, errors are given in pre-flight checks, and processing aborts. Specify '-demo' to bypass these safety checks.

This option should NOT be used for production installations.

-no_cron

Skip initialization of the crontab. A crontab file is generated in the /p4 directory, but is not loaded as the active crontab.

-no_ppr

Skip addition of the Perforce Package Repository for YUM/APT repos. By default, the Package Repository is added.

Specifying '-local' implies '-no_ppr'.

-no_sudo

Specify that no updates to sudoers are to be made.

WARNING: If systemd/systemctl is used to manage Perforce Helix services, the OSUSER that operates these services ('perforce' by default) requires sufficient sudo access to start and stop services using systemctl. Using '-no_sudo' may result in a unusable service being created if used on a system where the systemctl command is available.

If this option is used, consider also using '-no_systemd' to avoid the requiring systemd. Using systemd is recommended where available.

It is appropriate to use this option if the machine it operates on was based on a machine image that already grants the OSUSER sufficient sudo access.

This option is mutually exclusive with '-limited_sudo' and '-full_sudo'.

-limited_sudo

Specify that limited sudo access for the OSUSER created is to be granted. See 'gen_sudoers.sh -man' for details.

This option is mutually exclusive with '-no_sudo' and '-full_sudo'.

This option is recommended for optimal security.

-full_sudo

Specify that full sudo access for the OSUSER created is to be granted. See 'gen_sudoers.sh -man' for details.

This option is mutually exclusive with '-no_sudo' and '-limited_sudo'.

-v

Specify '-v' to run the verify_sdp.sh script after the SDP installation is complete. If '-v' is specified and the verify_sdp.sh script is available in the SDP, it is executed.

If the Sample Depot is loaded with '-sampledepot' or a new server was initialized with '-init', the '-online' flag to the verify_sdp.sh script is added. If '-no_cron' is specified, the corresponding '-skip cron' option is added verify_sdp.sh. If '-empty' is specified, the '-skip' tests also exclude the offline_db and p4t_files checks in verify_sdp.sh.

-no_pkgs

Specify '-no_pkgs' to skip OS package installation using the package manager (yum, apt-get, or zypper).

WARNING: Using this option may cause the initial install and/or subsequent operation to fail, and this using this is not advised.

-extra_pkgs

Specify '-extra_pkgs' to install additional OS packages as may be needed for development of systems integrations, custom triggers, etc. See package lists above for more detail.

-local

By default, various files and binaries are downloaded from the Perforce Workshop and the Perforce FTP server as needed.

If the server machine on which this install_sdp.sh is to be run cannot reach the public internet or if using files from external sites is not desired, the '-local' flag can be used.

With '-local', needed files must be acquired and put in place on the server machine on which this script is to be run. Any missing files result in error messages and an aborted install.

Specifying '-local' implies '-no_ppr' and '-empty'.

For '-local' to work, the following must exist:

1. Helix Binaries

Helix binaries must exist in /opt/perforce/helix-sdp/helix_binaries:

- * /opt/perforce/helix-sdp/helix_binaries/p4
- * /opt/perforce/helix-sdp/helix_binaries/p4d

```
* /opt/perforce/helix-sdp/helix_binaries/p4broker
* /opt/perforce/helix-sdp/helix_binaries/p4p
```

2. Server Deployment Package (SDP)

The SDP tarball must be acquired and put in place here:

```
* /opt/perforce/helix-sdp/downloads/sdp.Unix.tgz
```

It can be acquired on a machine that can reach the internet with this command:

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

3. Sample Depot Tarball

The Sample Depot appropriate to your platform must exist if the '-sampledepot' option is used:

```
* /opt/perforce/helix-sdp/downloads/sampledepot.tar.gz (on UNIX/Linux or case-
sensitive Mac)
```

See EXAMPLES below for a sample of acquiring files for use with '-local' mode.

-no_firewall

Specify '-no_firewall' to skip updates to firewall.

By default, if on a system for which the host-local firewall service (firewalld or ufw) is available and running when this script is called, then the firewall service is updated to open appropriate ports for the Perforce Helix services installed.

-no_systemd

Specify '-no_systemd' to avoid using systemd, even if it appears to be available. By default, systemd is used if it appears to be available.

This is helpful in operating in containerized test environments where systemd is not available.

This option is implied if the systemctl command is not available in the PATH of the root user.

This option is mutually exclusive with '-no_enable'.

-no_enable

Specify '-no_enable' to avoid enabling systemd services that are installed and enabled by default. Specifically, this means that the call to 'systemctl enable' for installed services is

skipped.

This option is mutually exclusive with '-no_systemd'.

-t <ServerType>

Specify the type of server. Valid values are:

- * p4d_master - A p4d master/commit server.
- * p4d_replica - A p4d replica with all metadata from the master (not filtered in any way).
- * p4d_filtered_replica - A p4d filtered replica or filtered forwarding replica.
- * p4d_edge - An p4d edge server.
- * p4d_edge_replica - A p4d replica of a p4d edge server. The TargetServerID must also be set if ServerType is p4d_edge_replica.
- * p4broker - An SDP host running only a Helix Broker.
- * p4p - An SDP host running only a Helix Proxy.

-s <ServerID>

Specify the ServerID. A ServerID is required if the ServerType is any p4d_* type other than p4d_master.

-si <SDPInstance>

Specify the SDP Instance name. The SDP Instance name is incorporated into the folder structure of the installed service, with many files appearing under '/p4/<SDPInstance>', e.g. /p4/1.

The default is '1'.

-ts <TargetServerID>

Specify the Target ServerID. Set this only if ServerType is p4d_edge_replica. The value is the ServerID of edge server that this server is a replica of, and must match the ReplicatingFrom: field of the server spec.

-tp <TargetPort>

Specify the target port. For p4broker and p4p only.

-lp <ListenPort>

Specify the port to listen on. For p4broker and p4p only.

-se

Specify -se to simulate email. This generates a mail simulator script: /p4/common/site/bin/mail

-H <hostname>

Set the hostname. This is only supported on systems that support the 'hostnamectl' command. The hostname is set by doing: hostnamectl set-hostname --static <hostname>

If the corresponding 'Hostname' setting is defined in the configuration file and this '-H <hostname>' flag is used, the command line option will override the config file.

-T <timezone>

Set the timezone. This is only supported on systems that support the 'timedatectl' command. The timezone is set by doing: `timedatectl set-timezone <timezone>`

If the corresponding 'Timezone' setting is defined in the configuration file and this '-T <timezone>' flag is used, the command line option will override the config file.

DEVELOPMENT OPTIONS:

-sdp_dir <sdp_dir>

Specify a directory on the local host containing the SDP to deploy. This should not be used for production installs.

The directory specified by '-sdp_dir' is expected to contain either:

- * an SDP tarball (sdp.Unix.tgz) file, or
- * an already-extracted SDP directory, which must include the SDP Version file.

Use the special value '-sdp_dir default' to use the /sdp directory (as per the Docker-based SDP Test Suite environment).

DEBUGGING OPTIONS:

- d Enable debug message.
- D Enable extreme debugging with bash 'set -x'. Implies '-d'.

HELP OPTIONS:

- h Display short help message.
- man Display this full manual page.

--help

Alias for -man.

EXAMPLES:

=== Demo Installation - Helix Core Server ===

```
sudo su -
  mkdir -p /root/sdp_install
  cd /root/sdp_install
  curl -L -O
```

```
https://swarm.workshop.perforce.com/download/guest/perforce_software/sdp/main/Server/Unix/setup/install_sdp.sh
```

```
  chmod +x install_sdp.sh
  ./install_sdp.sh -sampledepot -demo
```

=== Typical Production Helix Core Server Installation - New Commit Server ===

Following is a sample set of instructions for a typical new server setup.

STEP 1: Configure storage.

For a production install, storage must first be configured before this script can be run.

See SDP documentation for guidance on storage configuration. There are a variety of options and methods for installing storage. However accomplished, when storage is complete the following, the directories must exist and must have storage mounted that is NOT on the OS root volume:

- * /hxdepots
- * /hxmetadata
- * /hxlogs

These paths are typical, but are configurable. More information is available in the install configuration file generated below.

STEP 2: Install this script.

Install this script in a directory under the root user's home directory with these commands:

```
$ sudo su -
$ mkdir /root/sdp_install
$ cd /root/sdp_install
$ curl -L -O
https://swarm.workshop.perforce.com/download/guest/perforce_software/sdp/main/Server/Unix/setup/install_sdp.sh
$ chmod +x install_sdp.sh
```

STEP 3: Generate install configuration file.

```
$ ./install_sdp.sh -C > sdp_install.cfg
```

STEP 4: Modify install configuration file.

Edit the generated sdp_install.cfg using your preferred text editor, changing the values as desired. This file contains various settings with documentation for each setting.

```
$ vi sdp_install.cfg
```

Once settings are decided, save the file.

STEP 5: Install SDP (Dry Run).

Call this script and reference the configuration file, as a dry run/preview:

```
$ ./install_sdp.sh -c sdp_install.cfg -init
```

Review the generated log of the preview, and address any reported issues.

STEP 6: Install SDP Live Run

```
$ ./install_sdp.sh -c sdp_install.cfg -init -y
```

This will install SDP per the per the command line and settings in the install configuration file.

=== Typical Production Helix Core Server Installation - Edge/Replica ===

When installing SDP on a machine intended to be a standby, replica, or edge server, the steps are exactly the same as for setting up a new commit server.

The content of the generated and then edited sdp_install.cfg file will have different

values for ServerType and ServerID settings. Ensure the CaseSensitivity value matches

the case of the commit server. (If the commit server is a Windows server and this current server machine is to be a Linux replica of a Windows commit server, the Linux server must be setup as case-insensitive.)

=== Standalone Proxy Installation ===

STEP 1: Install this script.

Install this script in a directory under the root user's home directory with these commands:

```
$ sudo su -
$ mkdir /root/sdp_install
$ cd /root/sdp_install
$ curl -L -O
```

```
https://swarm.workshop.perforce.com/download/guest/perforce-software/sdp/main/Server/Unix/setup/install_sdp.sh
```

```
$ chmod +x install_sdp.sh
```

STEP 2: Install the proxy.

Install the proxy, specifying the listen port (ssl:1666 in this example) and the target port (ssl:p4d.myco.com:1666).

```
$ ./install_sdp.sh -t p4p -lp ssl:1666 -tp ssl:p4d.myco.com:1666
```

=== Standalone Broker Installation ===

The instructions for installing the broker are identical to the instructions for installing the proxy, except that '-t p4broker' is used instead of

```
'-t p4p'.
```

```
=== Local Helix Core Server Install for Air Gap Networks ===
```

The following sample commands illustrate how to acquire the dependencies for running with '-local' on a machine that can reach the public internet. The resulting file structure, with paths as shown, must somehow be copied to the machine where this install_sdp.sh script is to be run. This can be used to facilitate installation on a machine over an "air gap" network.

```
$ sudo su -
$ mkdir -p /opt/perforce/helix-sdp/helix_binaries
$ cd /opt/perforce/helix-sdp/helix_binaries
$ curl -L -O
https://swarm.workshop.perforce.com/download/guest/perforce_software/sdp/main/helix_binaries/get_helix_binaries.sh
$ chmod +x get_helix_binaries.sh
```

If the latest major version available of Helix Core binaries are desired, do this:

```
$ ./get_helix_binaries.sh -sbd .
```

Or, if older Helix Core binaries are desired, append the version identifier with '-r rYY.N', as in this example to get Helix Core 2023.2 binaries:

```
$ ./get_helix_binaries.sh -sbd . -r r23.2
```

Next, get the SDP tarball and Sample Depot; the Sample Depot tarball:

```
$ mkdir /opt/perforce/helix-sdp/downloads
$ cd /opt/perforce/helix-sdp/downloads
$ curl -O https://ftp.perforce.com/perforce/tools/sampledepot.tar.gz
$ curl -L -O
https://swarm.workshop.perforce.com/download/guest/perforce_software/sdp/downloads/sdp.Unix.tgz
```

Lastly, acquire this script:

```
$ mkdir /root/sdp_install
$ cd /root/sdp_install
$ curl -L -O
https://swarm.workshop.perforce.com/download/guest/perforce_software/sdp/main/Server/Unix/setup/install_sdp.sh
$ chmod +x install_sdp.sh
```

These acquired files must then be transferred to the machine where the install is to occur, and must appear in the same directory structure. To recap, for a '-local' install, the following files in this structure must exist on the machine on which the install is to occur:

```
/root/sdp_install/install_sdp.sh
/opt/perforce/helix-sdp/downloads/sdp.Unix.tgz
```

/opt/perforce/helix-sdp/downloads/sampledepot.tar.gz (if the '-sampledepot' option is to be used).

```
/opt/perforce/helix-sdp/helix_binaries/p4
/opt/perforce/helix-sdp/helix_binaries/p4d
/opt/perforce/helix-sdp/helix_binaries/p4broker
/opt/perforce/helix-sdp/helix_binaries/p4p
```

An install session would then look something like this:

```
cd /root/sdp_install
./install_sdp.sh -C > sdp_install.cfg      # Generate a config file
vi sdp_install.cfg                        # Edit desired settings.
./install_sdp.sh -c sdp_install.cfg -init  # Dry run. Review log to ensure
success.
./install_sdp.sh -c sdp_install.cfg -init -y # Live run. Review log to ensure
success.
```

If the dry run is not successful, fix reported issues and try again.

Appendix J: More Detail on makedirs.sh

makedirs.sh:

USAGE for makedirs.sh v7.4.11:

```
makedirs.sh <instance> [-r <P4BinRel>] [-s <ServerID>] [-t <ServerType>] [-tp
<TargetPort>] [-lp <ListenPort>] [-I <svc>[,<svc2>]] [-MDD /bigdisk] [-MCD /ckps] [-
MLG /jnl] [-MDB1 /db1] [-MDB2 /db2] [-f] [-p] [-no_init|-no_systemd|-no_enable] [-fs|-
ls] [-cleartext] [-no_cron] [-no_firewall] [-no_broker] [-test [-clean]] [-n] [-L
<log>] [-d|-D]
```

OR

```
makedirs.sh <instance> [-c <CfgFile>] [-f] [-p] [-no_init|-no_systemd|-no_enable] [-fs|-
ls] [-cleartext] [-no_cron] [-no_firewall] [-no_broker] [-test [-clean]] [-n] [-L
<log>] [-d|-D]
```

or

```
makedirs.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. In this case, the user calls the `install_sdp.sh` script, which in turn calls this script. See '`install_sdp.sh -man`' for more information.
- * Adding new SDP instances (separate Helix Core data sets) to an existing SDP installation on a given machine. For this scenario, this `makedirs.sh` script is called directly.

An 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 *must* be run as root for the first instance on a machine; this first call is handled by `install_sdp.sh`.

Call `makedirs.sh` directly when adding additional instances to be co-located

on the same SDP server machine. Generally, makedirs.sh should always be run as root, as it does a more complete job. If not run as root, the following steps are not done:

- * Init mechanism installation, e.g. Systemd service or SysV init script installation.
- * SELinux configuration
- * Firewall configuration
- * Ownership (chown) commands

== 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
- * /hxcheckpoints
- * /hxdepots
- * /hxlogs
- * /hxmetadata
- * /hxmetadata2
- * /opt/perforce/helix-sdp (optional; used for package installations)

The directories starting with '/hx' are configurable, and can be changed by settings in the makedirs.cfg file (or makedirs.N.cfg), or with command line options as illustrated here:

```
-MDD /bigdisk
-MCD /ckps
-MLG /jnl
-MDB1 /db1
-MDB2 /db2
```

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

== Crontab ==

Crontabs are generated for all server types.

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).

== Init Mechanism ==

If this script is run as root, the init mechanism (Systemd or SysV) is configured for installed services.

The Systemd mechanism is used if the the /etc/systemd/system folder exists and systemctl is in the PATH of the root user. Otherwise, the SysV init mechanism is used.

== Firewall Configuration ==

This script checks to see if a known firewall type is available. The firewalld is checked using the command 'firewall-cmd --state' command, and the ufw firewall is checked using the 'ufw status'. If either firewall is detected, the ports required for Helix Core applications installed are opened in the firewall. For more information, see the templates in these folders:

```
/p4/common/etc/firewalld
/p4/common/etc/ufw
```

If the firewall service is not online, no firewall configuration is performed.

== SELinux Configuration ==

If Systemd is used and the semanage and restorecon utilities are available in the PATH of the root user, then SELinux configuration for the installed services is done.

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_commit - A master/commit server.
- * p4d_master - A synonym for p4d_commit.
- * 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.
- * p4p - An SDP host running only a standalone p4p, with no p4d.
- * p4proxy - A synonym for p4p.

-tp <TargetPort>

Specify the target port. Use only if ServerType is p4p and p4broker.

-lp <ListenPort>

Specify the listen port. Use only if ServerType is p4p 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. 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 that call the init scripts are supplied in /p4/common/etc/systemd/system.

-MDD /bigdisk

-MCD /ckps

-MLG /jnl

-MDB1 /db1

-MDB2 /db2

Specify the '-M*' options to specify mount points, overriding DD/CD/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.

-c <CfgFile>

Specify the path to the configuration file to use, overriding the default logic of finding the file based on naming convention.

-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 time-consuming and unnecessary.

-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.

`-no_init`

Specify `'-no_init'` to avoid any service configuration, which is done by default if running as root (using systemd if available, otherwise SysV). If `'-no_init'` is used, then neither systemd nor SysV init mechanism is configured for installed services.

This option is implied if not running as root.

This option is implied if `'-test'` is used.

`-no_systemd`

Specify `'-no_systemd'` to avoid using systemd, even if it appears to be available. By default, systemd is used if it appears to be available.

This is helpful in operating in containerized test environments where systemd does not work even if it appears to be available.

This option is implied if the `systemctl` command is not available in the PATH of the root user.

This option is implied if `'-no_init'` is used.

`-no_enable`

Specify `'-no_enable'` to avoid enabling systemd services to start automatically after a reboot. If this option is used, systemd services will still be created, allowing services to be manually started and stopped.

Specifically, this options means the `'systemctl enable'` command is not run for generated services.

`-no_cron`

Specify `'-no_cron'` to avoid loading the crontab.

A crontab file is generated in the `/p4` directory, but but with `'-no_cron'`, this file is not loaded as the active crontab.

`-no_firewall`

Specify `'-no_firewall'` to avoid attempting firewall configuration.

By default, if the `firewalld` service is found to be running, it is configured so that the ports for `p4d` and `p4broker` are open.

`-no_broker`

Specify `'-no_broker'` if installing a `p4d` service without a broker on the same server machine as `p4d`. By default, a broker is included

with p4d.

-fs Specify '-full' when calling gen_sudoers.sh to install a new, full sudoers file. This option is only available if running as root.

This option is mutually exclusive with '-ls'.

See 'gen_sudoers.sh -man' for more info.

-ls Specify '-limited' when calling gen_sudoers.sh to install a new, limited sudoers file. This option is only available if running as root.

This option is mutually exclusive with '-fs'.

See 'gen_sudoers.sh -man' for more info.

Specifying neither '-fs' nor '-ls' avoids calling gen_sudoers.sh. This is not advised because it not work at all, or may result in a system that does not work properly or may be insecure. The recommended option is to use '-ls' for enhanced security.

-cleartext

By default, SDP passwords are generated as encoded files. If this option is used, cleartext passwords are generated instead.

if a cleartext password file is specified, the file will be:
/p4/common/config/.p4passwd.p4_<SDP_Instance>.admin

Encoded password files have that name with a '.enc' suffix.

-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>.<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'.

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/hxdepts
- * /tmp/hxmounts/hxlogs
- * /tmp/hxmounts/hxmetadata

This option implies '-no_init'.

-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 makedirs.sh on a given machine should be done as root, but subsequent instance additions can 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 makedirs.cfg makedirs.abc.cfg
$ chmod +w makedirs.abc.cfg
$ vi makedirs.abc.cfg
```

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

```
$ ./makedirs.sh abc
```

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

Example 3: Setup of additional instance named 'alpha' to run a standalone p4p targeting commit.example.com:1666 and listening locally on port 1666.

```
$ sudo su -
$ cd /hxdepots/sdp/Server/Unix/setup
$ ./makedirs.sh alpha -t p4p -tp commit.example.com:1666 -lp 1666
```

Example 4: Setup of instance named '1' to run a standalone p4broker targeting commit.example.com:1666 and listening locally on port 1666.

```
$ sudo su -
$ cd /hxdepots/sdp/Server/Unix/setup
$ ./makedirs.sh 1 -t p4broker -tp commit.example.com:1666 -lp 1666
```

Example 5: Setup 2 instances A and B with limited sudoers on a fresh new machine:

```
$ sudo su -
$ cd /hxdepots/sdp/Server/Unix/setup
$ cp makedirs.cfg makedirs.A.cfg
```

Adjust settings in makedirs.A.cfg as desired, e.g P4PORT, P4BROKERPORT, etc.

```
$ cp makedirs.A.cfg makedirs.B.cfg
```

Adjust settings in makedirs.B.cfg as desired, e.g P4PORT, P4BROKERPORT, etc.
Ensure port numbers do not conflict. Then generate Instance A:

```
$ ./mkdirs.sh A -ls
```

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

Next generate instance B, updating the limited sudoers to reference both instances.

```
$ ./mkdirs.sh B -ls
```

SEE ALSO:

See 'install_sdp.sh -man' for more info on installing on a new machine.

See 'gen_sudoers.sh -man' for more info on generating/replacing sudoers.

See template:

- * systemd service file templates: /p4/common/etc/systemd/system
- * firewalld templates: /p4/common/etc/firewalld
- * ufw firewall templates: /p4/common/etc/ufw
- * Init script templates: /p4/common/etc/init.d