SDP Legacy Upgrade Guide (for Unix)

Perforce Professional Services

Version v2023.2, 2023-12-18

Table of Contents

Preface

This document provides an overview of the process to upgrade the Perforce Helix Server Deployment Package (SDP) from any older version (dating back to 2007) to the SDP 2020.1 release, also referred to as "r20.1".

If your SDP version is 2020.1 or newer, refer to the SDP Guide (Unix) for instructions on how to upgrade from SDP 2020.1 to any later version. Starting from SDP 2020.1, the upgrade procedure for the SDP is aided by an automated and incremental upgrade mechanism similar to p4d itself, capable of upgrade SDP from the current release to any future version so long as the current release is SDP 2020.1 or newer.

This document describes the process of upgrading to SDP 2020.1.

Please Give Us Feedback

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

Chapter 1. Overview

The Perforce Server Deployment Package (SDP) software package, just like the Helix Core software it manages, evolves over time and requires occasional upgrades to remain supported. Further, patches may be released over time.

This document discusses how to upgrade the SDP, and when in relationship to Helix Core itself.

1.1. Upgrade Order: SDP first, then Helix P4D

The SDP should be upgraded prior to the upgrade of Helix Core (P4D). If you are planning to upgrade 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 the 'p4' command line client on the server machine. Even if not strictly required, upgrading the SDP first is **strongly** recommended, as SDP keeps pace with changes in the p4d upgrade process, and can ensure a smooth upgrade.

1.2. Upgrading Helix P4D and Other Software

See the SDP Guide (Unix) for instructions on how to upgrade Helix binaries in the SDP structure after the SDP has been upgraded to 2020.1 or later.

1.3. SDP and P4D Version Compatibility

The SDP is often forward- and backward-compatible with P4D versions. However, for best results they should be kept in sync by upgrading SDP before P4D. This is partly because the SDP contains logic to upgrade P4D, which can change as P4D evolves.

The SDP is aware of the P4D version(s) it manages, 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 then officially supported versions, though without the "officially supported" guarantee.

1.4. SDP Upgrade Methods

There are several methods for upgrading to a new version of the SDP:

- **In-Place, Manual Upgrades**: Manual upgrades of the SDP must be performed to upgrade from versions older than 2020.1 in-place on existing machines. This document provides details on how to do this.
- **In-Place, Automated Upgrades**: Automation assisted in-place upgrades can be done if your current SDP version is 2020.1 or later. and you are upgrading to 2021.1 and later. Refer to documentation in SDP Guide (Unix) for upgrading *from* SDP 2020.1 onward.

- **Migration-Style Upgrades**: A migration-style upgrade is one in which the existing server machines (virtual or physical) are left in place, and brand new "green field" machines are installed fresh using the Helix Installer (which installs the latest SDP on a "green field" baseline machine with only the operating system installed). Then the Helix Core data is migrated from the existing hardware to the new hardware. This approach is especially appealing when upgrading other aspects of the infrastructure at the same time as the SDP, such as the hardware and/or operating system. Migration style upgrades require new hardware, and provide a straightforward rollback option because the original hardware is left in place.
- **Custom with HMS**: The Helix Management System is used by some customers. See Appendix 3.A, *Custom HMS Managed Installations*

Chapter 2. Upgrade Planning - In-Place Upgrades

Legacy SDP upgrades require some familiarization that should be done prior to scheduling an upgrade.

The following information is useful for planning SDP in-place upgrades.

2.1. Plan What Is Being Upgraded

Presumably if you are reading this, you are intending to upgrade the SDP. During planning, you'll want to decide if you want to upgrade only the SDP. Or you may want to upgrade Helix Core software in the same maintenance window.

Tactically, the SDP upgrade is done first. However, upgrading SDP and Helix Core can be done in the same maintenance window, so both upgrade tasks can be done in one upgrade session. Alternately, Helix Core can be upgraded at a later date after the SDP upgrade.

Decide whether Helix Core will be upgraded during the same work session that SDP is upgraded.

2.2. Upgrade Duration

Key questions to answer during upgrade planning are:

- Will p4d need to be taken offline?
- If so, how long?

Most upgrades to SDP to 2020.1 will require downtime for the Helix Core (p4d) server, even if p4d is not being upgraded. The only exception to requiring downtime is if the current SDP is r2019.*x* **and** your SDP is *not* configured to use `systemd`, i.e. does not have `/etc/systemd/system/p4*.service` files. (Most 2019-2021 era SDP deployments use systemd, so this is a narrow exception.)

If the current SDP **and** P4D versions are both r2019.1 or later, the downtime can be brief. The scripts can be upgraded while the server is live, stopping p4d only long enough to change the `systemd *.service` files with new ones.

If your SDP is older than 2019.1, other changes will be needed depending on your SDP version, which will extend the downtime for p4d needed to upgrade the SDP. Read on to get a sense for what steps are required, which vary based on your SDP version.

If you are upgrading P4D along with the SDP, and your P4D version is older than 2019.1, live checkpoints are recommended after the upgrade is complete. This can significantly extend downtime required. The "to-or-thru P4D 2019.1" upgrades involve significant Helix Core database structural changes. If your Helix topology includes edge servers, you'll want to account for taking checkpoints of the edge servers in "to-or-thru 2019.1" upgrade planning; edge checkpoints can occur in parallel with the checkpoint on the master server to reduce overall upgrade process

duration.

Drivers of Downtime Duration

The big drivers of required downtime duration are:

- Is SDP 2019.1+? If so, no SDP structural changes are needed, so less downtime is needed.
- Is P4D being upgraded to-or-thru 2019.1? If yes, significantly longer downtime is needed, both for the p4d upgrade process and taking a live checkpoint after.
- Time required to execute the SDP upgrade steps that you'll define in detail with information from later in this document. The older the SDP version, the more steps required.
- Sophisticated global topologies with many machines take longer for these one-time legacy upgrades, due to needing to run commands on multiple machines.

Note that, once on the SDP 2020.1+ and using P4D 2019.1+, we do not expect any future upgrades to require extended downtime. Upgrade simplification and downtime reduction are priorities for both Helix Core and the SDP.

2.3. SDP Machines and Instances

Early in your planning, you'll want to take stock of all server machines on which the SDP is to be upgraded.

For each machine, you'll need to be aware of what SDP instances need to be upgraded on that machine.

For each instance on any given machine, determine what servers or services are in place. For example, a given machine might be a master p4d server for one instance, a replica for another, and a simple proxy for a third instance.

2.4. Mount Point Names

You will need to be aware of the three standard SDP mount points. While referred to as "mount points" in this document, in any given installation, any or all of the three SDP mount points may be simple directories on the root storage volume, or symlinks to some other storage volume. In some installations, fewer than three volumes were used, and in some cases 4 were used (2 for metadata). In some cases the operating system root volume was used as one of the volumes. Investigate and be aware of how your installation was configured. Comparing the output of `pwd` and `pwd -P` in the same directory can be informative.

The mount points do not necessarily need to be changed during the SDP upgrade process, as the SDP structural design has always and remains flexible with respect to mount point names. However, understanding whether the "mount points" are actual mount points, regular directories or symlinks is something to be aware of for detailed planning.

In the examples below, the modern SDP mount point names are used:

- `/hxdepots` - Volume for versioned files and rotated/numbered metadata journals.
- `/hxmetadata`- Volume for active and offline metadata. In some cases the single `/hxmetadata` is replaced with the pair `/hxmetadata1` and `/hxmetadata2`.
- `/hxlogs` - Volume for active journal and various logs.

Depending on the version of the SDP, the above values may be used, or earlier defaults such as:

- `/depotdata` for `/hxdepots`
- `/metadata` for `/hxmetadata`
- `/logs` for `/hxlogs`.

In some cases, custom values were used like `/p4depots`, `/p4db`, `/p4jnl`, etc. In these cases, it is important to know what standard names are referred to by the local names.

In the sample steps in this document, adapt the steps to use your local values for mount point names to the new values.

If your site uses two volumes for metadata, `/hxmetadata1` and `/hxmetadata2`, continue using those same names.

2.5. SDP Structure

If your SDP is 2019.1 or newer, skip this section.

Become familiar with the Pre-SDP 2019.1 and SDP 2019.1+ structures. (Note: This is not related to the P4D version).

Determine if SDP structural changes are needed.

2.5.1. SDP Pre-2019.1 Structure

Before SDP 2019.1:

- `/p4` is a directory on the operating system root volume, `/`.
- `/p4/N` is a symlink to a directory is typically the mount point for a storage volume (`/hxdepots` by default).
- `/p4/N` contains symlinks for `/hxdepots`, `/hxmetadata`, and `hxlogs`, as well as tickets and trust files.

2.5.2. SDP 2019.1+ Structure

In SDP 2019.1+:

- `/p4` is a directory on the operating system root volume, `/`, (same as Pre-2019.1 Structure).
- `/p4/N` is local directory on the operating system root volume,

- `/p4/N` contains symlinks for `/hxdepots`, `/hxmetadata`, and `hxlogs`, as well as tickets and trust files (same as the Pre-2019.1 structure)
- `/p4/N/bin` is local directory on the operating system root volume. The `bin` directory is the only actual directory in `/p4/N`; other items are files or symlinks to directories.

The `verify_sdp.sh` script (included in the SDP starting with SDP 2019.1) gives errors if the 2019.1+ SDP structure is not in place.

Converting the SDP structure in-place to the new style requires downtime on the edge/replica of interest.

2.6. Topology Option: Shared Archives

A topology option with SDP deployments is to share the `/hxdepots` mount point across machines, e.g. with NFS. SDP upgrade procedures involve updating the `/p4/common` directory that is physically on the `/hxdepots` volume. When updating the `/p4/common` on one machine, be aware that any changes will be immediately visible on all other machines that share from the same NFS network location.

Be aware of any shared `/hxdepots` volumes when planning SDP upgrades.

2.7. P4HOME: Dir or Symlink

If your SDP is 2019.1 or newer, skip this section.

In current and all legacy SDP installations, including all topology variations, the top-level `/p4` directory is always a regular directory on the local machine. However, the instance directory, e.g. the `1` in `/p4/1/root`, might be a directory or a symlink depending on your SDP version.

In the modern SDP, the `/p4/N` directory (where `N` is the SDP instance, e.g. `1`), is also a regular directory on the local machine. This `/p4/N` directory is referred to as the `P4HOME` directory for the instance.

In older versions of the SDP, the `N` was a symlink rather than a local directory. Check with a command like `ls -l /p4/N` to determine if the P4HOME dir is a symlink or a directory. If it is a symlink, the upgrade procedure will need to change the symlink to a directory.

If the `N` in `/p4/N` is a symlink rather than a directory, you'll need to change that to a directory in the upgrade procedure.

2.8. Instance Bin: Dir or Symlink

If your SDP is 2019.1 or newer, skip this section.

In the modern SDP, the "Instance bin" directory, `/p4/N/bin`, is a directory. As with P4HOME, on older versions of the SDP, `/p4/N/bin` may be a directory or symlink. Check with a command like `ls -l /p4/N/bin` to determine if that is a directory or symlink, depending on how old the SDP is.

If the `bin` in `/p4/N/bin` is a symlink rather than a directory, you'll need to change that to a directory in the upgrade procedure.

2.9. Operating System User (OSUSER)

You will need to be aware of your operating system user (OSUSER) that `p4d` runs as in your environment.

The sample upgrade steps below assume that Perforce runs as the `perforce` operating system user, which is typical. You do not need to change it, but you will need to adapt the samples below if your OSUSER is something other than `perforce`.

The OUSER user should dedicated to operating the Perforce Helix Core and related Helix services.

2.10. Home Directory for OSUSER

In modern installations, the default home directory is `/home/perforce`, though in some older installations the home directory is `/p4`. In either case, this does not need to be changed during the upgrade process.

As a general guideline, we recommend using a local home directory for the account under which p4d runs, as opposed to an auto-mounted home directory. Using an auto-mounted home directory can be a source of operational instability. While changing to a local directory is not a hard requirement, it is recommended.

2.11. Metadata Symlink Type

Depending on how old the SDP in place is, the structure will have either *Fixed* or *Variable* Metadata Symlinks. Determine which you have.

To determine this, login as the OSUSER (e.g. `perforce`), and run a command like this sample (for instance `1`):

```
ls -l /p4/1/root /p4/1/offline_db
```

The `root` and `offline_db` will be always symlinks in all versions of the SDP. However, they might be fixed or variable.

Variable Metadata Symlink References

If one of the symlinks points to a directory ending in `db1`, and the other in `db2` (it doesn't matter which is pointing to which), you have **variable metadata symlinks**.

Fixed Metadata Symlink References

If the target of the `root` and `offline_db` symlinks points to directories ending in the same names, i.e. `root` and `offline_db`, then you have **fixed metadata symlinks.**

If you have **fixed metadata symlinks**, your upgrade procedure will need to convert them to **variable metadata symlinks**, per examples below.

2.12. Init Mechanism: Systemd or SysV

You will need to determine whether you are running with the Systemd or SysV init mechanism.

Generally, newer operating systems like RHEL/CentOS 7 & 8, Ubuntu 18.04 and 20.04, and SuSE 12 & 15 will run with the `systemd` init mechanism. Older ones likely use the SysV init scripts (with some exceptions). If the command `systemctl` exists in your path, then you are running a system that supports systemd.

2.13. Depot Spec Map Fields

During planning, for each SDP instance, determine if depot spec `Map:` fields are used, and if the triggers table references the script `SetDefaultDepotSpecMapField.py`.

The modern SDP uses the `server.depot.root` configurable to define where depots are stored. In older versions of the SDP, the `Map:` field of the depot spec was used to track which depots were stored physically on which volumes. Starting in 2014, `p4d` introduced the `server.depot.root` configurable to define a standard location for all depots. The SDP took advantage of this, and stopped using the depot spec `Map:` fields. The `Map:` field was managed manually, or with the add of the (now obsolete) `SetDefaultDepotSpecMapField.py`.

For each instance, use the `p4 depots` command, and examine the `Map:` field of each depot spec to see if it has the default value or not. The default value is the name of the depot followed by a trailing `/…`. If any `Map:` fields have values other than the default, we want to be aware of that during planning, as we'll need to deal with them during the upgrade.

Following is a way to perform the analysis quickly:

```
# Get a list of depots.
p4 -ztag -F %type%:%name%:%map% depots > /tmp/d1.txt
```

```
# Trim depot types that don't apply for this check.
grep -Ev '^(remote|graph):' /tmp/d1.txt  > /tmp/d2.txt
cut -d ':' -f 2,3 /tmp/d2.txt > /tmp/d3.txt
```

```
# This next long single-line command does the check.
while read -r dm; do d=${dm%%:*}; m=${dm##*:}; echo D=[$d] M=[$m]; [[ "${d}/..." ==
"$m" ]] || echo NONDEFAULT_MAP for depot $d; done < /tmp/d3.txt
```

If the text `NONDEFAULT_MAP` appears in the output, there are depots with non-default `Map:` fields that need to be addressed.

Why not use depot spec `Map:` fields?

Using the `Map:` field of the depot spec had operational complexities, as it could not be easily seen unless the `p4d` process was online, which often it would not be during maintenance when depot moves occurred. Using symlinks to track what depots are on what physical volumes is simpler, always accessible, and allows for per-machine variations that are sometimes needed.

Use the `p4 triggers -o` command and see if your triggers table contains a reference to the `SetDefaultDepotSpecMapField.py`. If so, plan to remove that trigger during the upgrade.

2.13.1. Multiple Depot Storage Volumes

The SDP uses the `server.depot.root` configurable to define where depots are stored. **All** depots must **appear** to p4d as being in `/p4/N/depots`, which is the `server.depot.root` value. In cases where depots are physically be stored on multiple storage volumes, symlinks in the `/p4/N/depots` directory reference the actual storage location.

For example, say for instance `1` you use `/hxdepots`, `/hxdepots2` and `/hxdepots3` volumes. The `/p4/N/depots` symlink will point only to `/hxdepots/p4/N/depots`. This "primary storage volume," i.e. the one pointed to by the `depots` symlink, contains a mix of depot directories (for depots physically stored on that `/hxdepots` volume) and symlinks for each depot stored on the other volumes. Running an `ls` might look like:

```
$ ls -l /hxdepots/p4/1/depots/
total 0
drwxrwxr-x   4 perforce  staff  128 Feb 16  2012 HR
drwxrwxr-x   8 perforce  staff  256 Feb 16  2012 depot
drwxrwxr-x   5 perforce  staff  160 Feb 16  2012 gwt
lrwxr-x--x   1 perforce  staff   34 Feb 28 10:29 gwt-streams ->
/hxdepots3/p4/1/depots/gwt-streams
drwxrwxr-x   4 perforce  staff  128 Feb 16  2012 jam
drwxrwxr-x   7 perforce  staff  224 Feb 16  2012 pb
drwxr-x--x  13 perforce  staff  416 Feb  4 12:55 spec
lrwxr-x--x   1 perforce  staff   29 Feb 28 10:29 system ->
/hxdepots2/p4/1/depots/system
```

During planning, determine if you need to add symlinks in your primary depot storage volume. These symlinks can safely be added immediately, though `p4d` will not use them until the `Map:` field of the server spec is cleared during the upgrade.

Once you have determined which instances require clearing of depot spec `Map:` fields and/or removal of the `SetDefaultDepotSpecMapField.py` trigger, plan to run the `clear_depot_Map_fields.sh` script for those instance during the upgrade processing.

For info on this script:

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.
```

2.14. ServerID Definition and Replica Setup

Modern SDP uses and requires `server.id` files in the \$P4ROOT directory. If your SDP does not have `server.id` files in the \$P4ROOT directory, you'll need to add them before starting p4d after upgrading.

When upgrading the SDP, you may want to redo replica, replacing any existing replicas with new ones generated with the SPD `mkrep.sh` script, and ensuring the ServerID values follow the SDP Server Spec Naming Standard and replication best practices. See the SDP Guide (Unix) to learn about using `mkrep.sh`.

If you do plan to do this, you may want to plan your overall upgrade procedure as follows:

1. Upgrade SDP (as detailed in this document).
2. Upgrade Helix Core to the latest version.
3. Redo Replication (if not already using SDP best practices).

Tackling the replication redo increases the benefit of the overall upgrade. However, if checkpoints take a long time in your environment, it also may significantly increase the required duration of the downtime in the maintenance window. This is due to the need for new checkpoints as needed during the replication recreation process. It is possible to defer replication upgrades to a separate maintenance operation.

Chapter 3. Upgrade Procedure - In-Place Upgrades

With the familiarization and planning from the last section complete, move on to defining an upgrade procedure for your topology.

The procedure is broken into 3 phases:

- **Preparation**: Preparation steps can be done in a non-disruptive manner on a production server ahead of the Execution, possibly days or more ahead of the target date for the actual upgrade.
- **Execution**: Execution steps are generally performed in a scheduled maintenance window.
- **PostOp**: PostOp steps are done some time after the upgrade is complete, perhaps days or weeks later. For example, some cleanup is done in PostOp. Often the upgrade procedure leaves copies of various files and directories around to support a fast abort of the upgrade. Such files are comforting to have around during the upgrade procedure, but after a time become clutter and should be removed.

3.1. Preparation

Preparation steps are detailed below.

These procedure involves deploying new files on the Production server ahead of the actual upgrade. As prescribed, the steps avoid interaction with a live running p4d server and with the SDP script operations of the current SDP. The preparation steps can be safely done without affecting behavior until you are ready to execute the upgrade. However, human error is always a possibility when working on a production server. Respect the machine, and type carefully!

3.1.1. Plan Communications

Crafting emails can be time consuming! Communications to end users should be prepared so that they are ready to send quickly during maintenance.

For the most part, this type of "back end system" upgrade is transparent to end users other than the associated downtime. However, if you have power users granted direct SSH access to the server machines, such users should also be made aware of SDP changes.

3.1.2. Plan System Backups

If your environment has special backup capabilities, such as snapshots of key storage volumes and/or the entire machine, it should be determined whether such things are to be utilized during this upgrade (generally before anything starts).

3.1.3. Run verify_sdp.sh

If you SDP is 2019.1 or newer, it will have the `verify_sdp.sh` script. Run it during preparation to

ensure you have a good start state. Resolve issues detected by the script.

3.1.4. Plan User Lockout

There are a variety of strategies for locking out users during maintenance. Choose what combination to apply in your environment.

- **Protections table**: Create a near-empty "maintenance mode" Protections table that references only the P4USER used by the SDP (typically `perforce`) and any replication service users (`svc_*`). As maintenance starts, save the standard Protections table, and put the maintenance one in place. At the end of maintenance, bring the original one back in place. (Be aware of whether your site has any polices or custom automation that might interfere with this method.)
- **Temp Firewall Changes**: Using network and/or host firewall rules can block end users out during maintenance. Be wary not to block replicas. (Warning: This can be complex and hard to get right, and may involve coordination with other teams if the Perforce admin does not have direct control over such things. Choose this option with care.)
- **Temp P4PORT Change**: In some environments, the P4PORT and P4BROKERPORT configured in the Instance Vars files are changed to non-production values not known to users during maintenance, and switched back when done. If there are replicas, there is a need to do `p4d -cset` commands to change replica's P4TARGET back and forth between production and maintenance mode values.
- **DFM Brokers**: Down For Maintenance "(DFM)" brokers can be used for some types of upgrade. DFM brokers cannot be used if SDP structure change are needed, as all processes must be down in that case. The legacy upgrade procedure outlined in this document does not account for using DFM brokers, and for this particular upgrade we do not recommend using them. However, it is possible; see Appendix 3.B, *DFM Brokers for Legacy Upgrade*.

3.1.5. Acquire Downloads

Download the latest SDP tarball release from this link: https://swarm.workshop.perforce.com/projects/perforce-software-sdp/download/downloads/sdp.Unix.tgz.

Copy the downloaded tarball to the machine and put it in `/hxdepots/sdp.Unix.tgz`. (If a file with the same name exists from a previous upgrade, move it aside first.)

3.1.6. Extract new `sdp` Directory

```
mkdir /hxdepots/new
cd /hxdepots/new
tar -xzf /hxdepots/sdp.Unix.tgz
cat /hxdepots/new/sdp/Version
```

Verify that the contents of the `Version` file are as expected.

The `/hxdepots/new` folder structure is not referenced by the SDP, and thus a new SDP can be safely staged there ahead of the execution of the SDP upgrade.

3.1.7. Generate new SDP Shell Environment Files

The following SDP shell environment files are generated and should be reviewed, comparing new files generated with a `*.new` extension with the corresponding files in the existing installation (without the `.new` suffix). Be careful not to modify the production files; only update the *.new files.

The Shell Environment Files to generate for review are:

3.1.7.1. The p4_vars File

The single `p4_vars` file is the main SDP shell environment file. Generate the new form of the file like this example, operated as the `perforce` OSUSER:

```
cd /p4/common/bin
```

```
# This long one-line command does a sed and redirects to a file.
sed -e "s:REPL_OSUSER:$USER:g" -e "s:REPL_SDPVERSION:$(cat
/hxdepots/new/sdp/Version):g"
/hxdepots/new/sdp/Server/Unix/p4/common/config/p4_vars.template > p4_vars.new
```

```
# Preserve the current KEEP* settings.
grep -E 'export KEEP.*=' p4_vars >> p4_vars.new
```

The old file will contain `KEEP*` settings, possibly with custom values that should be preserved. That `grep` command above handles preservation of the `KEEP*` settings.

3.1.7.2. Instance Vars Files

The `/p4/common/config` directory in the SDP contains `p4_N.vars` shell environment files, one per SDP instance (e.g. `p4_1.vars`, `p4_abc.vars`, etc.).

What if there is no `/p4/common/config` ?

If the existing SDP does not have a `/p4/common/config` directory at all, as will be the case for very old versions of SDP, you can safely create the `config` directory during preparation.

In that case, leave off the `*.new` extension for files to be generated in `/p4/common/config, i.e. the `p4_N.vars` files (one per instance) and optional `p4_N.p4review.cfg` files created in that directory. Create the `/p4/common/config` directory by copying from the new SDP area, like so:

```
su - perforce
cp -pr /hxdepots/new/sdp/Server/Unix/p4/common/config /p4/common/.
```

In the following example, replace `1 abc` with your actual list of SDP instance names, delimited by

spaces. Note that if you have many machines in your topology, it is possible that each machine may have a different set of instances. You'll need to be aware of which instances are active on which machines. On any given machine with a given set of instances, do something like:

```
su - perforce
cd /p4/common/config
for i in 1 abc; do cp
/hxdepots/new/sdp/Server/Unix/p4/common/config/instance_vars.template
p4_${i}.vars.new; done
```

The format of `p4_N.vars` files has evolved over time, so it is important to generate new files from the new template. For each p4_N.vars.new file, search for the string `REPL_` in the file to find strings that need to be replaced (everywhere except in comment blocks). Settings to be replaced are:

- MAILTO
- MAILFROM
- P4USER
- P4MASTER_ID
- SSL_PREFIX
- P4PORTNUM
- P4BROKERPORTNUM
- P4MASTERHOST (appears in some older versions as P4MASTER)

As needed, refer to the original `p4_N.vars` files to retrieve values. Following is the complete list of settings to check and see if you have values defined in your original Instance Vars file that may differ from the new template:

- MAILTO
- MAILFROM
- P4USER
- P4MASTER_ID
- SSL_PREFIX
- P4PORTNUM
- P4BROKERPORTNUM
- P4MASTERHOST
- PROXY_TARGET
- PROXY_PORT
- P4DTG_CFG
- SNAPSHOT_SCRIPT
- SDP_ALWAYS_LOGIN

- SDP_AUTOMATION_USERS
- The 'umask' setting, which is set with a command like `umask 0026`.

3.1.8. Plan Password File Location Changes

In the modern SDP, passwords are stored in files like:

```
/p4/common/config/.p4passwd.p4_1.admin
/p4/common/config/.p4passwd.p4_abc.admin
```

where `1` and `abc` are sample instance names. If you have such files, no changes for passwords are needed during the upgrade procedure. In very old versions of the SDP, the password for the super user (defined by the P4USER setting in the shell environment files) was defined in the single file:

```
/p4/common/bin/adminpass
```

If you have the old password file `/p4/common/bin/adminpass`, plan to copy from the legacy location to the new home in `/p4/common/config` it during the upgrade procedure. If you have more than one instance, you will need to copy the `adminpass` file to multiple `/p4/common/config/.p4passwd.p4_N.admin` files.

3.1.9. Account for Typical Customization

If the SDP has been customized in your environment, custom upgrade procedures may be required. An understanding of what was customized and why will be useful in determining if custom upgrade procedures are required.

In typical deployments, the SDP is not customized, or only customized in some way that is no longer needed due to improvements in the "stock" SDP.

Starting with the SDP 2020.1 release, there is a new mechanism to cleanly separate typical configuration changes from customizations. The SDP `p4_vars` file and Instance Vars files may contain local custom modifications made by administrators or Perforce Consultants. When generating new Instance Vars files, be sure to review old Instance Vars files for any custom code or logic.

The `p4_vars` files and generated Instance Vars files (e.g. `p4_1.vars`) each have a line at the bottom of the generated portion of the file that looks like this:

```
### MAKE LOCAL CHANGES HERE:
```

If any custom functionality is to be preserved, be sure to add it ***only*** after that line containing `# MAKE LOCAL CHANGES HERE:`

Future automated SDP upgrades will preserve any lines below the `# MAKE LOCAL CHANGES HERE:` line. In addition to preserving content below that line, the right-side of all variable assignments

anywhere in the file will be preserved extracted from existing Instance Vars files as well as `p4_vars`. For example, say if you have:

```
export MAILLIST=MyAdminList@MyCompany.com
```

That variable assignment will be preserved in future SDP upgrades.

Use this information when reviewing your `p4_vars.new` and `p4_1.vars.new` (or other Instance Vars files) in preparation for the upgrade.

3.1.10. Account for SDP Additions

In many cases, customers have added custom trigger scripts into the SDP structure. Triggers are typically added in `/p4/common/bin/triggers` and possibly with instance-specific triggers in `/p4/N/bin/triggers`. Generally, custom triggers should not need to be changed, and can remain in place during the SDP upgrade.

However, custom triggers should be reviewed to determine if they depend on any custom environment variables that may have been added to the SDP shell environment files (the `p4_vars` file and/or Instance Vars) to support the custom triggers. Consider custom triggers in Section 3.1.9, “Account for Typical Customization”.

3.1.11. Deeper Customizations

If you need help determining if and how the SDP was customized in your environment, Perforce Consulting may be of assistance. Note that customizations are not supported by Perforce Support.

3.1.12. Instance P4Review Scripts

The legacy p4review "review daemon" scripts are still supported in the SDP, but have become obsolete with Helix Swarm offering the needed functionality. Determine if you still need them. If not, skip to the next section.

If Helix Swarm is used, use Swarm's `honor_p4_reviews` feature to displace the legacy `p4review` scripts and config files. Swarm has its own "project-based" email notification scheme, which can be augmented with `honor_p4_reviews` to also provide notifications based on the `Reviews:` field of the user spec.

If you need them, generate *.new config files as per this example. In the following example, replace `1 abc` with your actual list of SDP instance names for which review daemons are active. Review daemons are only enabled on the master server machine for an instance.

```
su - perforce
cd /p4/common/config
for i in 1 abc; do cp
/hxdepots/new/sdp/Server/Unix/p4/common/config/p4review.cfg.template
p4_${i}.p4review.cfg.new; done
```

Review the generated *.new config file and search for any settings whose values (the the right of the = sign) start with REPL_. Replace any such settings with appropriate values using your original file as a guide.

It is possible the new files, after configuration, may not have changed from the older ones if your SDP version is recent enough.

3.1.13. Broker Config Files

Broker config files, p4_N.broker.cfg, may also exist in /p4/common/config. These are not affected by the SDP upgrade procedure, and can be ignored.

In some older SDP deployments, broker config files were deployed to other locations such as /p4/N/bin. Any broker config files outside of /p4/common/config must be moved there and named per the naming convention for broker configuration files. For more information, see: https://swarm.workshop.perforce.com/projects/perforce-software-sdp/view/main/doc/SDP_Guide.Unix.html#_configuring_systemd_p4broker_multiple_configs

3.1.14. Generate New SDP Instance Bin Files

If the current SDP is 2018.1 or newer, skip this section.

Examine the p4d_N_init script in the 'instance bin' folder, /p4/N/bin.

Does the actual code look like this sample (with comments and the "shebang" #!/bin/bash line removed)?

```
export SDP_INSTANCE=N
/p4/common/bin/p4d_base $SDP_INSTANCE $@
```

If the p4d_N_init script already looks like this, then the 'instance bin' folder does not need to be touched during the upgrade process. Skip the rest of this section.

If, however, the *_init script has more code, then all the p4*_init scripts will need to be replaced during the upgrade execution. Templates are available in /p4/common/etc/init.d. The templates contains a few values that will need to be replaced.

Identify existing p4*_init scripts to be replaced, and create new files with a *.new suffix. For example, for instance 1, generate new p4d and p4broker init scripts like this:

```
cd /p4/1/bin
```

```
# This long command line generates p4d_1_init.new.
sed -e s:REPL_SDP_INSTANCE:1:g
/hxdepots/new/sdp/Server/Unix/p4/common/etc/init.d/p4d_instance_init.template >
p4d_1_init.new
```

```
chmod +x p4d_1_init.new
```

```
# This long command line generates p4broker_1_init.new.
sed -e s:REPL_SDP_INSTANCE:1:g
/hxdepots/new/sdp/Server/Unix/p4/common/etc/init.d/p4broker_instance_init.template >
p4broker_1_init.new
```

```
chmod +x p4d_1_init.new
```

3.1.15. Check for systemd service files

The format of Systemd service files (sometimes referred to as 'Unit' files) changed with the SDP 2020.1 release. As part of planning, it is helpful to identify if systemd is already in use, and which Perforce Helix services are managed with systemd.

If the `/etc/systemd/system` directory exists, then the Systemd init mechanism is available. On systems that use the Systemd init mechanism, we recommend using it. You can check that with an `ls` command:

```
ls -ld /etc/systemd/system
```

Once systemd is configured for any given service, SDP 2020.1 **requires** using the systemd mechanism (i.e. the `systemctl` command) to start/stop Perforce Helix services (for data safety and consistency of management). Depending on your SDP version and how it was installed, there may or may not already be `p4*.service` files.

You can get a list of such services with:

```
ls -lrt /etc/systemd/system/p4*.service
```

In any case, in the Execution phase below, new systemd `p4*.service` files will be put in place, which may be new, or may replace existing `p4*.service` files.

3.1.16. Plan Changes to Configurables

This step is optional, and requires basic familiarity with bash scripting.

Modern SDP versions contain a script named `configure_new_server.sh` that makes various configuration changes to a server. That script captures best practices for a "green field" fresh new server installation.

This implements best practices in various ways:

- Creates `spec` and `unload` depots.

- Runs several `p4 configure set` commands to define configurable values.

The `configure_new_server.sh` script should **not** be executed for existing instances as is, because it is intended only for "green field" installs. However, it can be used as a reference. Start by making a copy of it, an in your own version review what it does. Make case-by-case decisions on what configurables you want, what values you want, and whether you want it to create `spec` and `unload` depots. If you already have such depots, remove the logic block that creates the new depots. Be sure to understand each line of the script, and ensure the logic and values set fit your environment.

Expect this to be a painstaking process. Only set configurables that you are comfortable with.

This documentation may prove helpful: https://www.perforce.com/manuals/cmdref/Content/CmdRef/configurables.configurables.html. Also, you can contact Perforce Support for information on specific configurables.

The script should be crafted during upgrade preparation, which may require some research. It should be executed when specified in the Execution phase.

```
cd /hxdepots/new/sdp/Server/setup
cp configure_new_server.sh configure_THIS_server.sh
chmod +wx configure_THIS_server.sh
```

Then modify your local `configure_THIS_server.sh` script as desired. Typically the workflow for modifying the script involves running `p4 configure show` commands for any configurable the script would set with a `p4 configure set` command, and using the `p4 configure show` output to current values for settings with value the script would set.

If you need help determining evaluating your script, Perforce Consulting may be of assistance. While this part of the upgrade is optional, some of the configurables can result in significant performance and functional benefits.

3.2. Execution

This section outlines sample steps for executing an actual upgrade after planning and preparations identified in Section 3.1, “Preparation” have been completed. The following typically performed in a scheduled maintenance window.

Execution steps detailed below.

3.2.1. Lockout Users

Execute whatever steps was planned in Section 3.1.4, “Plan User Lockout”.

3.2.2. Disable Crontabs

Capture original crontabs on all servers, and then disable the active crontab. On each machine as `perforce`:

```
crontab -l
```

If the `crontab -l` command reports `no crontab for perforce`, you can skip remaining steps on that machine. Typically SDP machines will have something in crontab, so this is likely an issue you may want to fix before or during the upgrade procedure. If you decide no crontab is needed on a given machine, be aware of that for the execution in Section 3.6, “Re-Enable Crontabs”. If that `crontab -l` command does generate output, continue as `perforce`:

```
[[ -d /p4/common/etc/cron.d ]] || mkdir -p /p4/common/etc/cron.d
cd /p4/common/etc/cron.d
```

```
crontab -l > crontab.$USER.$(hostname -s).old.DELETE_ME_LATER
```

```
# This next long command adjusts the crontab.
sed 's:daily_backup.sh:daily_checkpoint.sh:g' crontab.$USER.$(hostname
-s).old.DELETE_ME_LATER | grep -Ev 'recreate_db_' > crontab.$USER.$(hostname -s)
```

```
crontab -r
```

You should then review the crontab.$USER.$(hostname -s) file. The long command above with the `sed` and `grep -v` is safe to run in that it will generate a functionally correct crontab file. However, it might result in a file that has comments mismatched with deleted text. That can be cleaned up with manual review.

3.2.3. Stop Services

Stop the `p4d` service for all instances on this machine. Also stop all p4broker services running on this machine (if any).

For this SDP maintenance, the broker cannot be left running (e.g. to broadcast a "Down For Maintenance (DFM)" message) because the structure change cannot be started until *all* processes launched from the SDP directory structure have stopped.

3.2.3.1. Stop Services with Systemd

Sample systemd commands to stop the services, executed as `perforce`:

```
sudo systemctl status p4d_1 p4broker_1
sudo systemctl stop p4d_1 p4broker_1
sudo systemctl status p4d_1 p4broker_1
```

The extra `status` commands before and after the start/stop commands are for situational awareness. These are not strictly necessary.

3.2.3.2. Stop Services with SysV

Sample SysV commands to stop the services, executed as `perforce`:

```
p4d_1_init status
p4d_1_init stop
p4d_1_init status
```

```
p4broker_1_init status
p4broker_1_init stop
p4broker_1_init status
```

The extra `status` commands before and after the start/stop commands are for situational awareness. These are not strictly necessary.

3.2.4. Backup the current SDP Common Dir

On each machine, copy the old SDP directory:

```
cd /hxdepots/p4
cp -pr common OLD.common.$(date +'%Y-%m-%d')
```

3.2.5. Upgrade Physical Structure

In this step, the physical structure of the upgrade is done for pre-2019.1 SDP. Skip this step if you are already on the 2019.1+ structure.

The structure of the SDP changed in the 2019.1 release, to increase performance and reduce complexity in post-failover operations. The following notes describe how to do an in-place conversion to the new structure.

Following is the procedure to upgrade the structure in-place on a machine.

In the following sample procedure, the default SDP instance name of `1` is used, and default mount point names are used. Adapt this to your environment by applying this procedure to each instance on any given machine. If you have multiple instances, apply this procedure for each instance, one at a time.

3.2.5.1. Replace Instance Symlink with Directory

Skip this step if you are already on the 2019.1+ structure.

Move the instance symlink aside, and replace it with a regular directory. Then copy the `.p4*` files (e.g. `.p4tickets` and `.p4trust`) into the new directory. Sample commands:

```
cd /p4
```

```
mv 1 1.old_symlink.OLD_DELETE_ME_LATER
mkdir 1
cd 1
cp -p /p4/1.old_symlink.OLD_DELETE_ME_LATER/.p4t* .
```

3.2.5.2. Convert Fixed to Variable Metadata Symlinks

Skip this step if you are already using Variable Metadata Symlinks.

If you have Fixed Metadata Symlinks, first convert them to to Variable Metadata Symlinks. If you already have Variable Metadata Symlinks, proceed to Section 3.2.5.1, “Replace Instance Symlink with Directory”

In this step, move the underlying directories that will be pointed to by the `root` and `offline_db` symlink names, and move them to their `db1` and `db2` names.

```
mv /hxmetadata/p4/1/root /hxmetadata/p4/1/db1
mv /hxmetadata/p4/1/offline_db /hxmetadata/p4/1/db2
```

3.2.5.3. Replace Instance Symlink with Directory

Recreate the same symlinks you see reported by the `ls` command:

```
ls -l /p4/1.old_symlink.OLD_DELETE_ME_LATER/*
cd /p4/1
```

```
ln -s /hxmetadata/p4/1/db1 root
ln -s /hxmetadata/p4/1/db2 offline_db
```

Do not just copy the sample commands above. Pay close attention to the `ls` output, and make sure the `root` points to whatever it was pointing to before, either a directory ending in `db1` or `db2` (unless you just converted from Fixed Metadata Symlinks in STEP 4). Also confirm that `offline_db` and `root` aren’t both pointing to the same directory; one should be pointing to `db1` and the other to `db2`.

Then, create additional symlinks akin to whatever else is in `/p4/1.old_symlink.OLD_DELETE_ME_LATER`

That should look something like this:

```
cd /p4/1
ln -s /hxdepots/p4/1/depots
ln -s /hxdepots/p4/1/checkpoints
ln -s /hxdepots/p4/1/checkpoints.YourEdgeServerID
ln -s /hxlogs/p4/1/logs
ln -s /hxlogs/p4/1/tmp
```

```
ls -l
```

Next, create the `bin` directory, as a local directory and copy files to it:

```
mkdir bin
cd bin
cp /p4/1.old_symlink.OLD_DELETE_ME_LATER/bin/p4d_1_init .
cp /p4/1.old_symlink.OLD_DELETE_ME_LATER/bin/p4broker_1_init .
ln -s /p4/common/bin/p4broker_1_bin p4broker_1
ln -s /p4/common/bin/p4_bin p4_1
```

Last, take a look at `/p4/1.old_symlink.OLD_DELETE_ME_LATER/bin/p4d_1` - that `p4d_1` will be either a tiny script or a symlink (depending on whether your p4d is case sensitive or not). If you server is case sensitive, it will be a symlink. If your server is case-insensitive, it will be a tiny script.

If your server is case sensitive, create the symlink like this:

```
ln -s /p4/common/bin/p4d_1_bin p4d_1
```

OR, if your server is case-insensitive, that p4d_1 will be a tiny script, so just copy it:

```
cp /p4/1.old_symlink.OLD_DELETE_ME_LATER/bin/p4d_1 .
```

Then, start your server again, and run the `verify_sdp.sh` script and confirm that it's happy now.

3.2.6. Put New SDP in place

Put the new SDP in place:

```
cd /hxdepots
mv sdp sdp.$(date +'%Y-%m-%d').OLD_DELETE_ME_LATER
mv new/sdp .
```

```
cd /p4
ln -f -s /hxdepots/sdp
```

The `/p4/sdp` symlink will now point to the new SDP, and `cat /p4/sdp/Version` will display the new SDP version. The `/p4/sdp` can be thought of as the full contents of the SDP tarball, while the `/p4/common`, `/p4/ssl`, and `/p4/<instance>` structure are the live SDP files and scripts.

3.2.7. Deploy new SDP Common Files

The `rsync` command here will update files in the live `/p4/common` structure with the latest files from

the SDP, overwriting changed SDP files. However, it will not update or remove any files that have been added under `/p4/common`, such as site-local triggers that may have been added in `/p4/common/bin/triggers`.

```
rsync -a /p4/sdp/Server/Unix/p4/common/ /p4/common
```

3.2.8. Put New SDP Init scripts In Place

If you generated new `p4*_init` scripts in preparation, put them in place now, doing something like this as `perforce`:

```
cd /p4/1/bin
mv p4d_1_init p4d_1_init.OLD_DELETE_ME_LATER
mv p4d_1_init.new p4d_1_init
```

```
mv p4broker_1_init p4broker_1_init.OLD_DELETE_ME_LATER
mv p4broker_1_init.new p4broker_1_init
```

3.2.9. Put New p4_vars In Place

Put the new p4_vars.new file in place doing something like this as `perforce`:

```
cd /p4/common/bin
mv p4_vars p4_vars.OLD_DELETE_ME_LATER
mv p4_vars.new p4_vars
```

3.2.10. Put New Instance Vars In Place

Put all the new Instance Vars files in place doing something like this as `perforce` and replacing `1 abc` with your list of instances:

```
cd /p4/common/config
```

```
# This single-line command moves old instance Vars files aside, and puts new ones in
place:
for i in 1 abc; do mv p4_${i}.vars p4_${i}.vars.OLD_DELETE_ME_LATER; mv
p4_${i}.vars.new p4_${i}.vars; done
```

3.2.11. Put New P4Review Files In Place

If you generated new P4Review files put them in place, doing something like this as `perforce` and replacing `1 abc` with your list of instances:

```
cd /p4/common/config
```

```
# This single-line command moves old P4Review files aside and puts the new ones in
place:
for i in 1 abc; do mv p4_${i}.p4review.cfg p4_${i}.p4review.cfg.OLD_DELETE_ME_LATER;
mv p4_${i}.p4review.cfg.new p4_${i}.p4review.cfg; done
```

3.2.12. Copy and Relocate Password Files

If the need to copy password files was identified in Section 3.1.8, “Plan Password File Location Changes”, do those copies now. Do something like this as `perforce` and replacing `1 abc` with your list of instances:

```
for i in 1 abc; do cp -pf /p4/common/bin/adminpass
/p4/common/config/.p4passwd.p4_${i}.admin; done
```

```
mv /p4/common/bin/adminpass /p4/common/bin/adminpass.DELETE_ME_LATER
```

3.2.13. Upgrade systemd service files

The format of systemd unit files changed with the SDP 2020.1 release. If systemd is not used, skip this section.

The SDP 2020.1 release includes templates for System unit files in `/p4/common/etc/systemd/system`. These should be deployed on each machine that uses SDP, and for each Helix service (e.g. `p4d`, `p4broker`, `p4p`) within each SDP instance.

For example, the following sample commands, which must be executed as `root`, will create or replace systemd `*.service` files for `p4d` and `p4broker` for SDP instance `1`:

```
cd /etc/systemd/system
```

```
sed -e s:__INSTANCE__:1:g -e s:__OSUSER__:perforce:g
/p4/common/etc/systemd/system/p4d_N.service.t > p4d_1.service
```

```
chmod 755 p4d_1.service
```

```
sed -e s:__INSTANCE__:1:g -e s:__OSUSER__:perforce:g
/p4/common/etc/systemd/system/p4broker_N.service.t > p4broker_1.service
```

```
chmod 755 p4broker_1.service
```

```
systemctl daemon-reload
```

Repeat the above procedure for each additional instance you have, replacing the `1` with the instance name. Be careful to make two substitutions on each `sed` command line. For example, if your instance is `abc`, replace `:1:` with something like `:abc:`, and replace `_1` with `_abc`.

The new format of the systemd *.service files works only with the new SDP scripts in `/p4/common/bin`. The services will not start if the new scripts are deployed without the changes to the *.service files, or vice versa.

Do **not** make the `chmod` commands more restrictive; the `*.service` files must be world-readable (.e.g `755`).

3.2.14. Upgrade SysV /etc/init.d

This step applies only to machines using the SysV init mechanism; see Section 2.12, “Init Mechanism: Systemd or SysV”.

In older versions of the SDP, there was sometimes a copy of the `/p4/N/bin/*_init` scripts in `/etc/init.d`. In modern SDP, these are replaced with symlinks. If they are already symlinks, no changes are needed.

The following sample commands, which must be executed as `root`, will create or replace init files for `p4d` and `p4broker` for SDP instance `1`:

```
cd /etc/init.d
mv p4d_1_init p4d_1_init.old.DELETE_ME_LATER
mv p4broker_1_init p4broker_1_init.old.DELETE_ME_LATER
ln -s /p4/1/bin/p4d_1_init
ln -s /p4/1/bin/p4broker_1_init
chkconfig --del p4d_1_init
chkconfig --del p4broker_1_init
chkconfig --add p4d_1_init
chkconfig --add p4broker_1_init
chkconfig p4d_1_init on
chkconfig p4broker_1_init on
```

The `mv` and `chkconfig --del` commands will report errors if there are no existing files to move or configurations to delete. Such errors can be safely ignored, or those commands can be skipped if the referenced files don’t exist.

3.2.15. Start Services

Start the `p4d` service for all instances on this machine. Also start p4broker services running on this machine (if any).

3.2.15.1. Start Services with Systemd

Sample systemd commands to start the services, executed as `perforce`:

```
source p4_vars 1
sudo systemctl status p4d_1 p4broker_1
sudo systemctl start p4d_1 p4broker_1
sleep 3
sudo systemctl status p4d_1 p4broker_1
p4 info
```

The extra `status` commands before and after the start/stop commands are for situational awareness. These are not strictly necessary.

The `systemctl start` command returns immediately after a request to the system mechanism has been made to start service. However, the return of the command should NOT be taken as an indication that the service is actually up; that must verified, e.g. with `p4 info`.

If you execute the second `systemctl status` command or the `p4 info` command too quickly after the `start` command, the service will not have started. Give it several seconds and try again.

3.2.15.2. Start Services with SysV

Sample SysV commands to start the services, executed as `perforce`:

```
p4d_1_init status
p4d_1_init start
p4d_1_init status
```

```
p4broker_1_init status
p4broker_1_init start
p4broker_1_init status
```

The extra `status` commands before and after the start/stop commands are for situational awareness. These are not strictly necessary.

3.3. Set SDP Counters

Set the SDP_VERSION and SDP_DATE counters.

The SDP_VERSION counter is the SDP version.

```
p4 counter SDP_VERSION "$(cat /p4/sdp/Version)"
p4 counter SDP_DATE "$(date +'%Y-%m-%d')"
```

3.4. Update Depot Specs

Based on the planning done in Section 2.13, "Depot Spec Map Fields", instances should be identified that require clearing of depot spec `Map:` fields and/or removal of the `SetDefaultDepotSpecMapField.py` trigger. For each instance, run the script as in the following example, replacing *N* with your actual instance name(s):

```
/p4/sdp/Server/Unix/p4/common/sdp_upgrade/clear_depot_Map_fields.sh -i N
```

If you have more than one instance, run the command for each instance, changing the *N* value for each instance.

3.5. Execute Planned Configuration Changes

If changes to configuration changes were identified in Section 3.1.16, "Plan Changes to Configurables" and a `configure_THIS_server.sh` script was crafted, execute it now as the perforce user:

```
/p4/sdp/Server/setup/configure_THIS_server.sh
```

3.6. Re-Enable Crontabs

Re-enable crontabs as `perforce` on each machine with commands like these:

```
crontab -l
crontab /p4/common/etc/cron.d/crontab.$USER.$(hostname -s)
crontab -l
```

The `crontab -l` before and after is done for situational awareness and to confirm the crontab was loaded correctly.

3.7. Open The Flood Gates

At this point, do any sanity testing you desire to have confidence everything is OK.

Then, allow users back in by undoing whatever you did to lock users out in Section 3.2.1, “Lockout Users”.

3.8. Post Operation Steps

Cleanup steps can occur after the upgrade. In some cases cleanup is done immediately following the upgrade; in other cases it may be deferred by days or weeks.

3.8.1. Cleanup

Temporary files and directories with `.DELETE_ME_LATER` suffixes created during the upgrade procedure can now be deleted.

Appendix A: Custom HMS Managed Installations

If the Helix Management System (HMS) is used to manage this installation, you should have custom site-specific documentation for upgrading the SDP that supersedes this documentation. If the file `/p4/common/bin/hms` exists at your site, you have an HMS-managed site. Contact Perforce Consulting for more information.

Note that HMS solutions are inherently custom and not officially supported, but can be fully automated for global Helix Core topologies.

Appendix B: DFM Brokers for Legacy Upgrade

If you desire to use DFM brokers for the legacy upgrade to SDP 2020.1 as outlined in this document, you cannot use the standard init scripts and procedures for managing the p4broker process, as those scripts and procedures will be disrupted during the upgrade process. Also, using a DFM broker will tie up the port number, creating a bit of complexity and interfering with scripts or procedures that start/stop the services. Care must be taken to avoid attempting to run the DFM broker during certain parts of the upgrade process.

With those caveats noted, if care is taken, it is possible to use a specially configured DFM broker for the legacy upgrade to SDP 2020.1. It can be done by making a copy of the new `p4broker` binary to some non-SDP-managed location (such as `/home/perforce`) and generating a disposable configuration file.

Here is a sample procedure to use a DFM broker. You will need to modify the sample configuration file to use the correct ports for your environment.

```
mkdir /home/perforce/dfm
cd /home/perforce/dfm
curl -s -O http://ftp.perforce.com/perforce/r23.1/bin.linux26x86_64/p4broker
```

```
chmod +x p4broker
```

Here is a similar sample using a DFM broker on SSL:

```
mkdir /home/perforce/dfm
cd /home/perforce/dfm
curl -s -O http://ftp.perforce.com/perforce/r23.1/bin.linux26x86_64/p4broker
chmod +x p4broker
chmod 700 /home/perforce/dfm
export P4SSLDIR=/home/perforce/dfm
./p4broker -Gc
```

Then, create a broker config file `/home/perforce/dfm/p4broker.cfg` with contents like this sample:

```
  target      = ssl:1666;
  listen      = ssl:1667;
  directory   = /home/perforce/dfm;
  logfile     = p4broker.dfm.log;
  admin-email = null;
  admin-phone = null;
  admin-name = null;

  command: .*
  {
    action  = reject;
    message = "\nPerforce Helix is Down for scheduled maintenance\n";
  }
```

The DFM broker can then be started with a simple command line like:

```
export P4SSLDIR=/home/perforce/dfm/p4broker
/home/perforce/dfm/p4broker -c /home/perforce/dfm/p4broker.cfg
```

It can be stopped with a command like:

```
pkill -f /home/perforce/dfm/p4broker
```

This stop command will stop only the DFM broker. (Note: The `-f` in that command is not "force", but rather "full regex matching"; see `man pgrep` for more information.)

In your planning, account for starting and stopping the DFM broker at times to avoid interfering with operations that start/stop services.

If you have multiple machines in your environment, copy the `/home/perforce/dfm` directory to all machines, and account for starting/stopping the DFM broker on all machines in your planning.

Appendix C: Upgrading from an earlier patch of SDP 2020.1

If you are upgrading SDP from a prior patch of 2020.1, the procedure is the same as above, though most steps can be skipped (per details above). **When upgrading from the 2020.1 GA release or any subsequent patch to the current latest patch, no Helix Core service downtime is required**.

Note that customers upgrading from development DEV-PRE-RELEASE versions of SDP 2020.1 will require a small amount of downtime if the the systemd init mechanim is used, as the systemd *.services files will need to be updated.