Server Deployment Package (SDP) for Perforce Helix

SDP User Guide (for Unix)

Perforce Professional Services

Version v2019.3, 2020-08-14

Table of Contents

Preface

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

- Simplify Management

- High Availability (HA)

- Disaster Recovery (DR)

- Fast and Safe Upgrades

- Production Focus

- Best Practice Configurables

- Optimal Performance, Data Safety, and Simplified Backup

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

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

Please Give Us Feedback

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

Chapter 1. Overview

The SDP has four main components:

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

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

1.1. Using this Guide

Section 2 describes concepts and re-requisites

Section 3 consists of what you need to know to setup Helix Core sever on a Unix platform.

Section 4 covers upgrading SDP

Section 5 covers details of the configuration

Section 6 gives information around the Backup, Restoration and Replication of Helix Core.

Section 7 is about Server Maintenance, upgrades and

Section 8 gives some performance tips.

Section 9 covers the scripts used within the SDP in some more detail.

1.2. Getting the SDP

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

Chapter 2. Setting up the SDP

This section tells you how to configure the SDP to setup a new Helix Core server. Whilst the standard installation of Helix Core is fully covered in the System's Administrator Guide this section covers the details most relevant to the SDP.

The SDP can be installed on multiple server machines, and each server machine can host one or more Helix Core server instances.

2.1. Terminology and pre-requisites

1. The term *server* refers to a Helix Core server *instance*, unless otherwise specified.

2. The term *metadata* refers to the Helix Core database files

3. *Instance:* a separate Helix Core instantiation using its own p4d daemon/process

Pre-Requisites:

1. The Helix Core binaries (p4d, p4, p4broker, p4p) have been downloaded (see section XXXX)

2. *sudo* access is required

3. System administrator available for configuration of drives / volumes (especially if on network or SAN or similar)

4. Supported Unix version, currently these versions are fully supported - for other versions please speak with Perforce Support

    ◦ Ubuntu 16.04 LTS (xenial)

    ◦ Ubuntu 18.04 LTS (bionic)

    ◦ CentOS or Red Hat 6.x

    ◦ CentOS or Red Hat 7.x

    ◦ SUSE Linux Enterprise Server 12

2.2. Volume Layout and Hardware

As can be expected from a version control system, good disk (storage) management is key to maximising data integrity and performance. Perforce recommend using multiple physical volumes for **each** server instance. Using three or four volumes per instance reduces the chance of hardware failure affecting more than one instance. When naming volumes and directories the SDP assumes the "hx" prefix is used to indicate Helix volumes (your own naming conventions/standards can be used instead). For optimal performance on UNIX machines, the XFS file system is recommended but not mandated.

- **Perforce metadata (database files), 1 or 2 volumes:** Use the fastest volume possible, ideally SSD or RAID 1+0 on a dedicated controller with the maximum cache available on it. These volumes default to `/hxmetadata1` and `/hxmetadata2`.

    It is fine to have these both pointing to the same physical volume, e.g. `/hxmetadata`.

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

  If a separate logs volume is not available, put the logs on the `/hxmetadata1` or `/hxmetadata` volume.

- **Depot data, archive files, scripts, and checkpoints**: Use a large volume, with RAID 5 on its own controller with a standard amount of cache or a SAN or NAS volume (NFS access is fine). This volume is the only volume that **must** be backed up. The SDP backup scripts place the metadata snapshots on this volume.

  This volume is normally called `/hxdepots`.

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

  > 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.

On Unix/Linux platforms, the SDP will create a "convenience" directory containing links to the volumes for each instance, by default named `/p4`. The volume layout is shown in Appendix SDP Package Contents. This convenience directory enables easy access to the different parts of the file system for each instance.

For example:

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

Chapter 3. Installing the SDP on Unix / Linux

To install Perforce Server and the SDP, perform the steps laid out below:

- Set up a user account, file system, and configuration scripts.

- Run the configuration script.

- Start the server and configure the required file structure for the SDP.

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

   ```
   sudo groupadd perforce
   ```

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

   ```
   sudo useradd -d /p4 -s /bin/bash -m perforce -g perforce
   ```

3. Create or mount the server file system volumes (per layout in previous section)

   - /hxdepots

   - /hxmetadata1

   - /hxmetadata2

   - /hxlogs

4. These directories should be owned by: `perforce:perforce`

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

5. Either download the SDP directly or move the previously downloaded version to `/hxdepots`

   ```
   cd /hxdepots
   ```

   ```
   wget
   https://swarm.workshop.perforce.com/downloads/guest/perforce_software/sdp/downloads
   /sdp.Unix.2019.3.26571.tgz
   ```

   Or:

   ```
   mv sdp.Unix.2019.3.26751.tgz /hxdepots
   ```

6. Untar and uncompress the downloaded sdp files:

```
tar -zxvf sdp.Unix.2019.3.26751.tgz
```

7. Set environment variable SDP, this makes certain later steps easier.

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

8. Make the entire $SDP (/hxdepot/sdp) directory writable:

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

9. Download the appropriate p4, p4d and p4broker binaries for your release and platform (substituting desired release for r20.1 below)

```
cd $SDP/Server/Unix/p4/common/bin
wget http://ftp.perforce.com/perforce/r20.1/bin.linux26x86_64/p4
wget http://ftp.perforce.com/perforce/r20.1/bin.linux26x86_64/p4d
wget http://ftp.perforce.com/perforce/r20.1/bin.linux26x86_64/p4broker
```

10. make them executable

```
chmod +x p4*
```

3.1. Initial setup

The next steps highlight the setup and configuration of a new Helix Core instance using the SDP.

1. cd to $SDP/Server/Unix/setup and copy mkdirs.cfg to an instance specific version such as mkdirs.1.cfg and edit it, information on the variables can be found in Chapter 5, *Configuration script* of this document.

   Example:

   ```
   cd $SDP/Server/Unix/setup
   cp mkdirs.cfg mkdirs.1.cfg
   vi mkdirs.1.cfg
   ```

   Set the following:

```
P4ADMINPASS=********
MAILFROM=perforceadmin@myDomain.com
MAILHOST=myMailServer.myDomain.com
P4DNSNAME=thisMachine.myDomain.com
P4SERVICEPASS=********
MASTER_ID=myName.${SDP_INSTANCE}
```

2. As the root user (or sudo), run this:

```
mkdirs.sh _<instance number/name>_
```

e.g.

```
mkdirs.sh 1
```

> **!** If you use a name for the instance you MUST modify the P4PORT variable in `mkdirs.cfg`

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

+ e.g. `mkdirs.sh 1` requires `mkdirs.1.cfg`, or `mkdirs.sh lon` requires `mkdirs.lon.cfg`, or

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

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

> **!** the license file must be renamed to `license`

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

3.2. For Systems using systemd

RHEL 7, CentOS 7, SuSE 12, Ubuntu (>= v16.04) (and other) distributions utilize **systemd / systemctl** as the mechanism for controlling services, replacing the earlier init process. At present mkdirs.sh does **not** generate the systemd configuration file(s) automatically, but a sample is included in the SDP distribution in (`$SDP/Server/Unix/setup/systemd`), along with a README.md file that describes the configuration process, including for multiple instances.

We recommend that you give the OS user (perforce) sudo access, so that it can run the commands below prefixing them with sudo.

For simple installation run these commands as the root user (or prefix with sudo):

```
cp $SDP/Server/Unix/setup/system/p4d_1.system /etc/systemd/system/
systemctl enable p4d_1
```

|The above enables service for auto-start on boot. The following show management commands:

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

> If you are using systemd and you have configured systemctl services, then it is vital you ALWAYS use systemctl to start/stop etc. Otherwise you risk 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 doign it cleanly and waiting for them, because it thinks the service is not running).

3.3. For (older) systems, still using init.d

The mkdirs.sh script creates a set of startup scripts in the instance-specific bin folder:

```
/p4/1/bin/p4d_1_init
/p4/1/bin/p4broker_1_init     # only created if a p4broker executable found
/p4/1/bin/p4p_1_init          # only created if a p4p executable found
```

Run these commands as the root user (or sudo): Repeat this step for all init scripts you wish to add.

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

Chapter 4. Upgrading an existing SDP installation

If you have an earlier version of the Server Deployment Package (SDP) installed, you'll want to be aware of the new -test flag to the SDP setup script, mkdirs.sh e.g.

```
sudo mkdirs.sh 1 -test
```

This will install into /tmp and allow you to recursively diff the installed files with your existing installation and manually update as necessary.

See the instructions in the file README.md / README.html in the root of the SDP directory.

Chapter 5. Configuration script

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

Parameter	Description
DB1	Name of the hxmetadata1 volume (can be same as DB2)
DB2	Name of the hxmetadata2 volume (can be same as DB1)
DD	Name of the hxdepots volume
LG	Name of the hxlogs volume
CN	Volume for /p4/common
SDP	Path to SDP distribution file tree
SHAREDDATA	TRUE or FALSE - whether sharing the /hxdepots volume with a replica - normally this is FALSE
ADMINUSER	P4USER value of a Perforce super user that operates SDP scripts, typically perforce or p4admin.
OSUSER	Operating system user that will run the Perforce instance, typically perforce.
OSGROUP	Operating system group that OSUSER belongs to, typically perforce.
CASE_SENSITIVE	Indicates if server has special case sensitivity settings
SSL_PREFIX	Set if SSL is required so either "ssl:" or blank for no SSL
P4ADMINPASS P4SERVICEPASS	Password to use for Perforce superuser account - can be edited later in /p4/common/config/.p4password.p4_1.admin Service User's password for replication - can be edited later - same dir as above.
P4DNSNAME	Fully qualified DNS name of the Perforce master server machine

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

5.1. Use of SSL

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

```
SSL_PREFIX=ssl:
```

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

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

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

For example using instance 1:

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

In order to validate that SSL is working correctly:

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

Check that P4TRUST is appropriately set in the output of:

```
p4 set
```

Update the P4TRUST values (answer yes when prompted - the second command uses the value of the hostname command):

```
p4 -p ssl:1666 trust
```

```
p4 -p ssl:`hostname`:1666 trust
```

Check the stored P4TRUST values:

```
p4 trust -l
```

Check not prompted for trust:

```
p4 login
p4 info
```

5.1.1. Starting/Stopping Perforce Server Products

The SDP includes templates for initialization (start/stop) scripts, "init scripts," for a variety of Perforce server products, including:

- p4d
- p4broker
- p4p
- p4dtg
- p4ftpd
- p4web

The init scripts are named `/p4/<instance>/bin/<service>_<instance>_init`, e.g. `/p4/1/bin/p4d_1_init` or `/p4/1/bin/p4broker_1_init`.

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. The perforce user can start p4d by calling:

```
p4d_1_init start
```

And stop it by calling:

```
p4d_1_init stop
```

Once logged into Perforce as a super user, the p4 admin stop command can also be used to stop p4d.

All init scripts can be started as the perforce user or the root user (except p4web, which must start initially as root). The application runs as the perforce user in any case. If the init scripts are configured as system services (non-systemd distributions), they can also be called by the root user using the service command, as in this example to start p4d:

```
service p4d_1_init start
```

Templates for the init scripts used by `mkdirs.sh` are stored in:

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

There are also basic crontab templates for a Perforce master and replica server in:

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

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

The Perforce should have a super user defined as named by the P4USER setting in mkdir.

To configure and start instance 1, follow these steps:

1. Start the Perforce server by calling

```
p4d_1_init start
```

2. 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 (which calls the p4 login command using the `.p4passwd.p4_1.admin` file)

3. For new servers, run this script, which sets several recommended configurables:

```
$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.

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

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

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

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

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

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

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

If in doubt contact support.

5.1.2. 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@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 support@perforce.com

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

- Any init scripts used in `/etc/init.d`.

- 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.

5.2. Archiving configuration files

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

- The scheduler configuration.

- Cluster configuration scripts, failover scripts, and disk failover configuration files.

5.3. Configuring protections, file types, monitoring and security

After the server is installed and configured, 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 server using groups. Perforce uses an inclusionary model. No access is given by default, you must specifically grant access to users/groups in the protections table. It is best for performance to grant users specific access to the areas of the depot that they need rather than granting everyone open access, and then trying to remove access via exclusionary mappings in the protect table even if that means you end up generating a larger protect table.

4. To set the server's 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 the server from attempting to compress them again before storing them.

    ◦ For regular binary files, add `binary+l` to make so that only one person at a time can check them out.

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

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

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

5.4. Other server configurables

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

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.

5.5. General SDP Usage

This section presents an overview of the SDP scripts and tools. Details about the specific scripts are provided in later sections.

5.5.1. Linux

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

Older versions of the SDP required you to always run important administrative commands using the `p4master_run` script, and specify fully qualified paths. This script loads environment information from `/p4/common/bin/p4_vars`, the central environment file of the SDP, ensuring a controlled environment. The `p4_vars` file includes instance specific environment data from `/p4/common/config/p4_`**`instance.`**`vars` e.g. `/p4/common/config/p4_1.vars`. The `p4master_run script` is still used when running p4 commands against the server unless you set up your environment first by sourcing p4_vars with the instance as a parameter (for bash shell: `source /p4/common/bin/p4_vars 1`). Administrative scripts, such as `daily_backup.sh`, no longer need to be called with `p4master_run` however, they just need you to pass the instance number to them as a parameter.

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

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

Below are some usage examples for instance 1.

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

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

5.5.2. Monitoring SDP activities

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

complete.

For further monitoring, you can consider options such as:

- Making the SDP log files available via a password protected HTTP server.

- Directing the SDP notification emails to an automated system that interprets the logs.

5.6. P4V Performance Settings

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

Chapter 6. Backup, Replication, and Recovery

Perforce servers maintain *metadata* and *versioned files*. The metadata contains all the information about the files in the depots. Metadata resides in database (db.*) files in the server's root directory (P4ROOT). The versioned files contain the file changes that have been submitted to the server. 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 the Knowledge Base articles about replication.

6.1. Typical Backup Procedure

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

Seven nights a week, perform the following tasks:

1. Truncate the active journal.

2. Replay the journal to the offline database. (Refer to Figure 2: SDP Runtime Structure and Volume Layout for more information on the location of the live and offline databases.)

3. Create a checkpoint from the offline database.

4. Recreate the offline database from the last checkpoint.

Once a week, perform the following tasks:

1. Verify all depot files.

Once every few months, perform the following tasks:

1. Stop the live server.

2. Truncate the active journal.

3. Replay the journal to the offline database. (Refer to Figure 2: SDP Runtime Structure and Volume Layout for more information on the location of the live and offline databases.)

4. Archive the live database.

5. Move the offline database to the live database directory.

6. Start the live server.

7. Create a new checkpoint from the archive of the live database.

8. Recreate the offline database from the last checkpoint.

9. Verify all depots.

This normal maintenance procedure puts the checkpoints (metadata snapshots) on the hxdepots volume, which contains the versioned files. Backing up the hxdepots volume with a normal backup

utility like *robocopy* or *rsync* provides you with all the data necessary to recreate the server.

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

The preceding maintenance procedure minimizes server downtime, because checkpoints are created from offline or saved databases while the server is running.

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

6.2. 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. Use of the broker may require use of a P4AUTH server for authentication.

6.2.1. 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.

6.2.2. Using **mkrep.sh**

This script automates the following:

- creation of all the configurables for a replica appropriate to its type (e.g. forwarding-replica, forwarding-standby, edge-server etc).

- standard naming conventions are used for server ids, service user names etc. This simplifies managing multiple server/replica topologies and understanding the intended use of a replica (e.g. that it is intended for HA - high availability)

- creation of service user account, password, and with appropriate permissions

- creation of server spec

- detailed instructions to follow in order to create a checkpoint and restore on the replica server

Prerequisites:

- You must have a server spec for your master server, typically defined with Services: "commit-server" ("standard" is fine if no edge servers are to be created, but it is not a problem to use commit-server even without any edge servers) - use the serverid (output of "p4 serverid") as the name.

- You should be running p4d 2018.2 or later

- You should have a configuration file which defines site tags - this is part of naming.

Server Types

These are:

- ha - High Availability

- ham - High Availability (Metadata only)

- ro - Read only replica

- rom - Read only replica (Metadata only)

- fr - Forwarding replica

- fs - Forwarding standby

- frm - (Metadata only)

- fsm - (Metadata only)

- ffr - Filtered forwarding replica

- edge - Edge server

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.

Example

An example run is:

```
/p4/common/bin/mkrep.sh -i 1 -t fs -s bos -r replica1 -skip_ssh
```

The above will:

- Create a replica for instance 1

- Of type `fs` (forwarding standby) - with appropriate configurables

- For site `bos` (e.g. Boston)

- On host name `replica1`

- Without checking that passwordless ssh is possible to the host replica1

The tag has several purposes:

- Short Hand. Each tag represents a combination of 'Type:' and fully qualified 'Services:' values used in server specs.

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

- 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).

Mkrep.sh output

The output (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, e.g.

- Rotate the current live journal (to save the configuration parameters required)

- Copy across latest checkpoint and the subsequent rotated journals to the replica host machine

- Restore the copied checkpoints/journals into `/p4/<instance>/root` (and `offline_db`)

- Create a password file for service user

- Create appropriate server.id files

- Login the service user to the upstream server (usually commit server)

- Start the replica process

- Monitor that all is well with `p4 pull -lj`

More details on these steps can be found in the manual process below as well as the actualy mkrep.sh output.

Usage

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

$THISSCRIPT -i <SDP_Instance> -t <Type> -s <Site_Tag> -r <Replica_Host> [-f
<From_ServerID>] [-p] [-ssh_opts=\"opts\"] [-skip_ssh] [-L <log>] [-si] [-v<n>] [-n]
[-D]
```

```
or

 $THISSCRIPT [-h|-man|-V]
 "
    if [[ $style == -man ]]; then
       echo -e "
DESCRIPTION:
    This script creates makes a replica, and provides enough information to
    make it ready in all respects.


OPTIONS:
 -i <SDP_Instance>
    Specify the SDP Instance.


 -t <Type>
    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 values are:
    * ha:   High Availability mandatory standby replica, for 'p4 failover' (P4D
2018.2+)
    * ham:  High Availability metadata-only mandatory 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:

    $SiteTagsFile

    Current valid site tags defined in this file are:
    $(grep -v '^#' "$SiteTagsFile" 2>&1|grep -v '$^'|sed 's:^:\t:g')

-r <Replica_Host>
    Specify the target replica host.

-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.

    By default, this is determined dynamically checking the ServerID of the
    master server. 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'.

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

    By default, an error is displayed if the check fails, i.e. if super user
    access for the group $ServiceUsersGroup 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 $ServiceUsersGroup * //...

-ssh_opts=\"opts\"
    Specify '-ssh_opts' to pass paraemters on to the ssh command. For
    example, to specify ssh operation on non-standard port 2222, specify
    '-ssh_opts=\"-p 2222\"'.

-skip_ssh
    Specify '-skip_ssh' to skip the SSH access preflight check.

    This is useful if you only intend to do the metadata preparation phase
    of creating a new replica, prior to SSH being setup or perhaps even
    prior to the hardware being available.

-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.

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

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

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

  -D Set extreme debugging verbosity.

  -f Full Mode Setup: The completes an edge servers setup so no additional steps
        are required. This setup requires an ssh connection from the master to the
        edge to be in place first. It also requires the depot log journal and /p4
        mounts to be in place and setup as expected. This setup assumes a standard
        SDP setup.

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

DEPENDENCIES:
        This script depends on ssh keys being defined to allow the Perforce
        operating system user ($OSUSER) to ssh to any necessary machines
        without a password.

        This script assumes the replica host already has the SDP fully
        configured.

FILES:
        This Site Tags file defines the list of valid geographic site tags:
        $SiteTagsFile

EXAMPLES:
        Prepare an edge server to run on host syc-helix-04:
        $THISSCRIPT -i acme -t edge -s syd -r syc-helix-04
 "
```

6.2.3. Manual Steps

In the sample below, the replica name will be `replica1`, it is instance 1 on a particular host, the service user name is `svc_replica1`, and the master server's hostname is `svrmaster`.

The following sample commands illustrate how to setup a simple read-only replica.

First we ensure that journalPrefix is set appropriately for the master server (in this case we assume instance 1 rather than a named instance):

```
p4 configure set master#journalPrefix=/p4/1/checkpoints/p4_1
```

Then we set values for the replica itself:

```
p4 configure set replica1#P4TARGET=svrmaster:1667
p4 configure set "replica1#startup.1=pull -i 1"
p4 configure set "replica1#startup.2=pull -u -i 1"
p4 configure set "replica1#startup.3=pull -u -i 1"
p4 configure set "replica1#startup.4=pull -u -i 1"
p4 configure set "replica1#startup.5=pull -u -i 1"
p4 configure set "replica1#db.replication=readonly"
p4 configure set "replica1#lbr.replication=readonly"
p4 configure set replica1#serviceUser=svc_replica1
```

Then the following also need to be setup:

- Create a service user for the replica (Add the Type: service field to the user form before saving):

```
p4 user -f svc_replica1
```

- Set the service user's password:

```
p4 passwd svc_replica1
```

- Add the service user svc_replica1 to a specific group ServiceUsers which has a timeout value of unlimited:

```
p4 group ServiceUsers
```

- Make sure the ServiceUsers group has super access in protections table:

```
p4 protect
```

Now that the settings are in the master server, you need to create a checkpoint to seed the replica. Run:

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

When the checkpoint finishes, rsync the checkpoint plus the versioned files over to the replica:

© 2010-2020 Perforce Software, Inc.

```
rsync -avz /p4/1/checkpoints/p4_1.ckp.###.gz perforce@replica:/p4/1/checkpoints/.
```

```
rsync -avz /p4/1/depots/ perforce@replica:/p4/1/depots/
```

(Assuming perforce is the OS user name and replica is the name of the replica server in the commands above, and that <mark>#</mark> is the checkpoint number created by the daily backup.)

Once the rsync finishes, go to the replica machine run the following:

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

Login as the service user (specifying appropriate password when prompted), and making sure that the login ticket generated is stored in the same place as specified in the P4TICKETS configurable value set above for the replica (the following uses bash syntax):

```
source /p4/common/bin/p4_vars 1
/p4/1/bin/p4_1 -p svrmaster:1667 -u svc_replica1 login
```

Start the replica instance (either using _init script or systemctl if on system):

```
/p4/1/bin/p4d_1_init start
```

Now, you can log into the replica server itself and run p4 pull -lj to check to see if replication is working. If you see any numbers with a negative sign in front of them, replication is not working. The most likely cause of this is that the service user is not logged in. Rerun the steps above to login the service user and check again. If replication still is not working, check /p4/1/logs/log on the replica, and also look for authentication failures in the log for the master instance on svrmaster.

The final steps for setting up the replica server are to set up the crontab for the replica server.

To configure the ssh trust:

On both the master and replica servers, go to the perforce user's home directory and run:

```
ssh-keygen -t rsa
```

Just use the defaults for the questions it asks.

Now from the master, run:

```
rsync -avz ~/.ssh/id_rsa.pub perforce@replica:~/.ssh/authorized_keys
```

and from the replica, run:

```
rsync -avz ~/.ssh/id_rsa.pub perforce@master:~/.ssh/authorized_keys
```

The crontab (/p4/p4.crontab) contains several lines which are prefixed by /p4/common/bin/run_if_replica.sh or run_if_edge.sh or run_if_master.sh

These can be tested to make sure all is valid with:

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

If "yes" is output then SDP thinks the current hostname with instance 1 is a replica server. Similarly for edge/master.

The log files will be in /p4/1/logs, so you can check for any errors from each script.

6.3. Recovery Procedures

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

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

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

6.3.1. 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.

6.3.2. 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 [_sync_replica_sh].

See the steps in the script [_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).

6.3.3. Recovering from a tape backup

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

To recover from a tape backup, perform the following steps.

1. Recover the hxdepots volume from your backup tape.

2. Create the /p4 convenience directory on the OS volume.

3. Create the directories /metadata/p4/instance/root/save and /metadata/p4/instance/offline_db.

4. Change ownership of these directories to the OS account that runs the Perforce processes.

5. Switch to the Perforce OS account, and create a link in the /p4 directory to /depotadata/p4/instance.

6. Create a link in the /p4 directory to /hxdepots/p4/common.

7. As a super-user, reinstall and enable the init.d scripts

8. Find the last available checkpoint, under /p4/instance/checkpoints

9. Recover the latest checkpoint by running:

```
/p4/_instance_/bin/p4d__instance_ -r /p4/_instance_/root -jr -z _last_ckp_file_
```

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

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

11. Reinstall the Perforce server license to the server root directory.

12. Start the perforce service by running 1/p4/1/bin/p4d_1_init start`

13. Verify that the server instance is running.

14. Reinstall the server crontab or scheduled tasks.

15. Perform any other initial server machine configuration.

16. 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.

6.3.4. Failover to a replicated standby machine

See DR-Failover-Steps-Linux.docx

Chapter 7. Server Maintenance

This section describes typical maintenance tasks and best practices for administering server machines.

7.1. Server upgrades

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

- Download the new p4 and p4d executables for your OS from ftp.perforce.com and place them in `/p4/common/bin`

- Run:

```
/p4/common/bin/upgrade.sh <instance>
```

  e.g.

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

- If you are running replicas, upgrade the replicas first, and then the master (outside → in)

Please refer to details for Section 9.2.1, "upgrade.sh"

7.2. Database Modifications

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

When upgrading the server, replaying a journal patch, or performing any activity that modifies the db.* files, you must restart the offline checkpoint process so that the files in the offline_db directory match the ones in the live server directory. The easiest way to restart the offline checkpoint process is to run the live_checkpoint script after modifying the db.* files, as follows:

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

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

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

Chapter 8. Maximizing Server Performance

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

8.1. Optimizing the database files

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

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

8.2. Proactive Performance Maintenance

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

8.2.1. 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.

8.2.2. 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 9. Tools and Scripts

This section describes the various scripts and files provided as part of the SDP package. To run main scripts, the machine must have Python 2.7, and a few scripts require Perl 5. The Maintenance scripts can be run from the server machine or from client machines.

The following various scripts.

9.1. Core Scripts

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

9.1.1. p4_vars

Defines the environment variables required by the Perforce server. 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.

Location: /p4/common/bin

9.1.2. p4_<instance>.vars

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

For instance `1`:

```
p4_1.vars
```

For instance `art`:

```
p4_art.vars
```

Location: /p4/common/config

9.1.3. p4master_run

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

This script is somewhat historical, in that most scripts now directly `source` the `p4_vars` script directly. It is still occasionally useful.

Location: /p4/common/bin

9.1.4. daily_checkpoint.sh

This script is configured to run six days a week using crontab. The script:

- truncates the journal

- replays it into the `offline_db` directory

- creates a new checkpoint from the resulting database files

- recreates the `offline_db` databsae from the new checkpoint.

This procedure rebalances and compresses the database files in the `offline_db` directory. These are rotated into the live (`root`) databse, by the script [_refresh_P4ROOT_from_offline_db_sh]

Location: /p4/common/bin

9.1.5. 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 9.1.6, "live_checkpoint.sh", as the latter will stop the server while it is running, which can take hours for a large installation.

Run this script if an error occurs while replaying a journal during daily checkpoint process.

This script recreates offline_db files from the latest checkpoint. If it fails, then check to see if the most recent checkpoint in the `/p4/<instance>/checkpoints` directory is bad (ie doesn't look like the right size compared to the others), and if so, delete it and rerun this script. If the error you are getting is that the journal replay failed, then the only option is to run Section 9.1.6, "live_checkpoint.sh" script.

Location: /p4/common/bin

9.1.6. live_checkpoint.sh

This is a fallback option for use when you suspect that the `offline_db` has become corrupt.

This performs the following actions:

- Stops the server

- Creates a checkpoint from the live database files

- Recovers the `offline_db` database from that checkpoint to rebalance and compress the files

Run this script when creating the server and if an error occurs while replaying a journal during the off-line checkpoint process.

> ❗ Be aware it locks live database for the duration of the checkpoint which can take hours for a large installation (please check the `/p4/1/logs/checkpoint.log` for the most recent output of `daily_backup.sh` to see how long checkpoints take to create/restore).

Location: /p4/common/bin

9.1.7. p4verify.sh

Verifies the integrity of the depot files. This script is run by crontab on a regular basis.

It verifies both shelves and ordinary 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@perforce.com

Our recommendation is that you should expect this to be without error, and you should address errors sooner rather than later. This could involved 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 (includ edge servers) up-to-date.

Location: /p4/common/bin

9.1.8. p4login

Executes a p4 login command, using the administration password configured in `mkdirs.cfg` and subsequently stored in a text file: `/p4/common/config/.p4passwd .p4_<instance>.admin`

Location: /p4/common/bin

Usage

```
    echo "USAGE for p4login v$Version:

 p4login [<instance>] [-p <port> | -service] [-automation] [-all]

    or

 p4login -h|-man
 "
    if [[ "$style" == -man ]]; then
       echo -e "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
$P4CCFG/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 behaviour is affected by external factors:
1. P4AUTH, if defined, affects login behavior on replicas.

2. The auth.id setting, if defined, affects login behaviors (and
generally simplifies them).

3. The \$SDP_ALWAYS_LOGIN variable. If set to 1, this causes p4login
to always execute a 'p4 login' command to generate a login ticket,
even if a 'p4 login -s' test indicates none is needed. By default,
the login is skipped if a 'p4 login -s' test indicates a long-term
ticket is available that expires 31+days in the future.
Add \"export SDP_ALWYAYS_LOGIN=1\" to $P4CCFG/p4_N.vars to
change the default for an instance, or to $P4CBIN/p4_vars to
change it globally. If unset, the default is 0.

4. If the P4PORT contains an ssl: prefix, the P4TRUST relationship
is checked, and if necessary, a p4 trust -f -y is done to establish
trust.

OPTIONS:
<instance>
    Specify the SDP instances. If not specified, the SDP_INSTANCE
    environment variable is used instead. If the instance is not
    defined by a parameter and SDP_INSTANCE is not defined, p4login
    exists immediately with an error message.

 -service
    Specify -service when run on a replica or edge server to login
    the super user and the replication service user.

    This option is not compatible with '-p <port>'.

 -p <port>
    Specify a P4PORT value to login to, overriding the default
    defined by P4PORT setting in the environment. If operating
    on a host other than the master, and auth.id is set, this
    flag is ignored; the P4TARGET for the replica is used
    instead.

```
       This option is not compatible with '-service'.

   -automation
       Specify -automation to login external automation users defined
       by the \$SDP_AUTOMATION_USERS variable.

   -v      Show ouptput of login attempts, which is suppressed by default.
       Setting SDP_SHOW_LOG=1 in the shell environment has the same
       effect as -v.

   -L <log>
       Specify the log file to use.  The default is /p4/N/logs/p4login.log

   -d      Set debugging verbosity.

   -D      Set extreme debugging verbosity.

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

 EXAMPLES:
       1. Typical usage for automation, with instance SDP_INSTANCE defined
       in the environment by sourcing p4_vars, and logging in only the super
       user P4USER to P4PORT:
       source $P4CBIN/p4_vars abc
       p4login

       Login in only P4USER to the specified port, P4MASTERPORT in this example:
       p4login -p \$P4MASTERPORT

       Login the super user P4USER, and then login the replication serviceUser
       for the current ServerID:
       p4login -service

       Login external automation users (see SDP_AUTOMATION_USERS above):
       p4login -automation

       Login all users:
       p4login -all

       Or: p4login -service -automation

 LOGGING:
       This script generates no output by default.  All (stdout and stderr) is
       logged to /p4/N/logs/p4login.log.

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

       If the '-v' flag is used, the contents of the log are displayed to
```

```
        stdout at the end of processing.

EXIT CODES:
        An exit code of 0 indicates a valid login ticket exists, while a
        non-zero exit code indicates a failure to login.
    "
```

9.1.9. p4d_<instance>_init

Starts the Perforce server.

> ❗ 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 9.3.10, "p4d_base").

> ℹ️ In clustered environments, put this script in the `/p4/<instance>/bin` directory and configure your cluster software to launch it from this location.

Location: `/p4/<instance>/bin` with a symlink to it from `/etc/init.d` (or a copy in `/etc/init.d` in a clustered environments). Templates for init scripts for other Perforce server products exist in `/p4/common/etc/init.d`

9.1.10. refresh_P4ROOT_from_offline_db.sh

This script is intended to be used every 1-3 months to 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.

Location: /p4/common/bin

9.1.11. run_if_master.sh

See Section 9.1.14, "run_if_master/edge/replica.sh"

9.1.12. run_if_edge.sh

See Section 9.1.14, "run_if_master/edge/replica.sh"

9.1.13. run_if_replica.sh

See Section 9.1.14, "run_if_master/edge/replica.sh"

9.1.14. run_if_master/edge/replica.sh

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

```
/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`)

9.2. More Server Scripts

These scripts are helpful components of the SDP that run on the server, but are not included in the default crontab schedules.

9.2.1. upgrade.sh

Runs a typical upgrade process, once new p4 and p4d binaries are available in `/p4/common/bin` - saved as `p4` and `p4d` respectively (overwriting any existing files with those names).

This script will:

- Rotate the journal (to provide a clean recovery point)
- Apply all necessary journals to offline_db
- Stop the server
- Create an appropriately versioned link for new p4/p4d/p4broker etc
- Link those into `/p4/1/bin` (per instance)
- Run `p4d -xu` on live and offline_db to perform database upgrades (in a version aware manner, for example pre 2018.2 servers are treated differently to 2018.2 or later servers)
- Restart server instance

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
```

> ℹ️ For a case insenstive server, `p4d_1` is not a link, but instead a very simple bash script which ensures that the parameter `-C1` is always supplied when the base `p4d` is executed. This is important to avoid possible mismatch in case sensitivity commands which risk corrupting the database.

Case insensitive `p4d_1` is not a link but executes:

```
#!/bin/bash
P4D=/p4/common/bin/p4d_1_bin
exec $P4D -C1 "$@"
```

Note that the above executes the same `p4d_1_bin` as below.

If we look in `/p4/common/bin` we see:

```
p4d_1_bin -> p4d_2020.1_bin
p4d_2020.1_bin -> p4d_2020.1.1991450
p4d_2020.1.1991450
```

The naming of the last comes from:

```
./p4d_2020.1.1991450 -V
:
Rev. P4D/LINUX26X86_64/2020.1/1991450 (2020/07/31).
:
```

So we see the build number `1991450` being included in the name of the p4d executable.

While this may seem an unecessarily complex series of links, it allows implementation scenarios where there are multiple instances on the same server machine, and where each instance is upgraded independently.

> ℹ️ it is not recommended to do the linking manually (although of course possible, but surprisingly easy to get wrong!).

Location: /p4/common/bin

9.2.2. p4.crontab

Contains crontab entries to run the server maintenance scripts.

Location: /p4/sdp/Server/Unix/p4/common/etc/cron.d

9.2.3. verify_sdp.sh

Does basic verification of SDP setup.

Location: /p4/common/bin

Usage

```
    echo "USAGE for verify_sdp.sh v$Version:

verify_sdp.sh [<instance>] [-online] [-si] [-L <log>|off ] [-D]

    or

verify_sdp.sh -h|-man
"
      echo -e "DESCRIPTION:

    This script verifies the current SDP setup for the specified instance.

    Useful if you change anything, particularly after an SDP upgrade.

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 require P4D to be online.

 -si    Silent mode.  Does not display stdout at the end of processing.

 -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.

 -D Set extreme debugging verbosity.

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

EXAMPLES:
    This script is typically called after SDP update with only the instance
    name or number as an argument, e.g.:

    verify_sdp.sh 1

    Or, if used from automation already doing its own logging, use -L off:
```

```
    verify_sdp.sh 1 -L off

LOGGING:
    This script generates a log file and also displays it to stdout at the
    end of processing.  By default, the log is:
    /p4/N/logs/verify_sdp.log.

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

    If the '-si' (silent) flag is used, the log is generated, but its
    contents are not displayed to stdout at the end of processing.

EXIT CODES:
    An exit code of 0 indicates no errors were encounted attempting to
    perform verifications, and that all checks verified cleanly.
"
```

9.3. 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.

File	Location	Remarks
dummy_ip.txt	`$SDP/Server/config`	Instructions for using a license on more than one machine. Typically used to enable a standby server. Contact Perforce Licensing before using.
Perl Modules (*.pm files)	`/p4/common/lib`	Modules used by some Perl scripts.

9.3.1. backup_functions.sh

This contains lots of standard Bash functions which are used in other scripts.

It is **sourced** (`source /p4/common/bin/backup_functions.sh`) by most of the other scripts in order to use the common shared functions and to avoid duplication.

It is not intendend to be called directly by the user.

9.3.2. broker_rotate.sh

This script rotates the broker log file on an instance that only has the broker running.

It can be adde to a crontab for e.g. daily log rotation.

Usage:

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

9.3.3. edge_dump.sh

This 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.

Run example:

```
./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 9.3.23, "recover_edge.sh"

9.3.4. edge_vars

This 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.

It is not intended for users to call directly.

9.3.5. edge_shelf_replicate.sh

This 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 support/consulting.

9.3.6. load_checkpoint.sh

Loads a checkpoint for commit/edge/replica instance.

Location: /p4/common/bin

Usage

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

 $THISSCRIPT <checkpoint> [-i <instance>] [-s <ServerID>] [-c] [-l] [-r] [-b] [-y] [-L
 <log>] [-si] [-v<n>] [-D]
```

```
or

$THISSCRIPT [-h|-man|-V]
"
    if [[ $style == -man ]]; then
        echo -e "
DESCRIPTION:
    This script loads a specified checkpoint into /p4/N/root and /p4/N/offline_db,
    where 'N' is the SDP instance name.

    At the start of processing, preflight checks are done. Preflight checks
    include:
    * The specified checkpoint and corresponding *.md5 file must exist.
    * The \$P4ROOT/server.id file must exist, unless '-s' is specified.
    * The \$P4ROOT/license file must exist, unless '-l' is specified.
    * Basic SDP structure and key files must exist.

    If the preflight passes, the p4d_N service is shutdown, and also the
    p4broker_N service is shutdown if configured.

    Next, the specified checkpoint is loaded. Upon completion, the Helix Core
    server process, p4d_N, is started.

    If the server to be started is a replica, the serviceUser configured for
    the replica is logged into the P4TARGET server.  Any needed 'p4 trust' and
    'p4 login' commands are done to enable replication.

    Note that this part of the processing will fail if the correct super user
    password is not stored in the standard SDP password file,

    /p4/common/config/.p4passwd.p4_N.admin

    After starting the server, a local 'p4 trust' is done if needed, and then
    a 'p4login -service -v' and 'p4login -v'.

    By default, the p4d_N service is started, but the p4broker_N service is not.
    Specify '-b' to restart both services.

ARGUMENTS AND OPTIONS:
 <checkpoint>
    Specify the path to the checkpoint file to load.

    The file may be a compressed or uncompressed checkpoint, and it may bea 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.

 -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 the \$P4ROOT/server.id file exists, value in the file must match the
     value specified with this flag.

     If no \$P4ROOT/server.id file exists, this flag is required.

  -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 availalbe, 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.

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

  -b Specify '-b' to start the a p4broker process (if configured).  By default
     the p4d process is started after loading the checkpoint, but the p4broker
     process is not. This can be useful to ensure the human administrator has
     an opportunity to do sanity checks before enabling the broker to allow
     access by end users (if the broker is deployed for this usage).

  -y Use the '-y' flag to bypass an interactive warning and confirmation
     prompt.

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

  -L <log>
     Specify the path to a log file.  By default, all output (stdout and stderr)
     goes to:
     /p4/<instance>/logs/${THISSCRIPT%.sh}.<timestamp>.log

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

  -si    Operate silently.  All output (stdout and stderr) is redirected to the log
     only; no output appears on the terminal.

  -D     Set extreme debugging verbosity.

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

```
     -V Dispay version info for this script and its libraries.

 FILES:

 EXAMPLES:
     Sample Usage (bash syntax):
     nohup $P4CBIN/load_checkpoint.sh /p4/1/checkpoints/p4_1.ckp.4025.gz -i 1 -y -si <
/dev/null > /dev/null 2>&1 &

     Then, monitor with:
     tail -f \$(ls -t \$LOGS/load_checkpoint.*.log|head -1)\\n
 "
```

9.3.7. gen_default_broker_cfg.sh

Generate 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
```

9.3.8. journal_watch.sh

This script will check diskspace available to P4JOURNAL and trigger a journal rotation based on specified thresholds. This is useful in case you are in danger of running out of disk space and your rotated journal files are stored on a separate partition than the active journal.

This script is using the following external variables:

- SDP_INSTANCE - The instance of Perforce that is being backed up. If not set in environment, pass in as argument to script.

- P4JOURNALWARN - Amount of space left (K,M,G,%) before min journal space where an email alert is sent

- P4JOURNALWARNALERT - Send an alert if warn threshold is reached (true/false, default: false)

- P4JOURNALROTATE - Amount of space left (K,M,G,%) before min journal space to trigger a journal rotation

- P4OVERRIDEKEEPJNL - Allow script to temporarily override KEEPJNL to retain enough journals to replay against oldest checkpoint (true/false, default: false)

Usage:

```
./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
```

9.3.9. kill_idle.sh

Runs `p4 monitor terminate` on all processes showing in the output of `p4 monitor show` that are in the IDLE state.

9.3.10. p4d_base

This is the script to start/stop/restart the `p4d` instance.

It is called by `p4d_1_init` script (and thus also `systemctl` on systemd Linux distributions)

It ensures appropriate parameters are specified for journal/log and other variables.

9.3.11. p4broker_base

Very similar to Section 9.3.10, "p4d_base" but for the `p4broker` service instance.

See p4broker in SysAdmin Guide

9.3.12. p4ftpd_base

Very similar to Section 9.3.10, "p4d_base" but for the `p4ftp` service instance.

This product is very seldom used these days!

See P4FTP Installation Guide.

9.3.13. p4login-super.sh

Very simple utility script to login the SDP defined super user.

© 2010-2020 Perforce Software, Inc.

9.3.14. p4p_base

Very similar to Section 9.3.10, "p4d_base" but for the p4p (P4 Proxy) service instance.

See p4proxy in SysAdmin Guide

9.3.15. p4pcm.pl

This utility removes files in the proxy cache if the amount of free disk space falls below the low threshold.

Usage

```
    print "\nUsage:\n
    $main::ThisScript [-d \"proxy cache dir\"] [-tlow <low_threshold>] [-thigh
<high_threshold>] [-n]
or
    $main::ThisScript -h

This utility removes files in the proxy cache if the amount of free disk space
falls below the low threshold (default 10GB).  It removes files (oldest first)
until the high threshold is (default 20GB) is reached. Specify the thresholds
in kilobyte units (kb).

The '-d \"proxy cache dir\"' argument is required unless \$P4PCACHE is defined,
in which case it is used.

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

Use '-n' to show what files would be removed.
";
```

9.3.16. p4review.py

Sends out email containing the change descriptions to users who are configured as reviewers for affected files (done by setting the Reviews: field in the user specification). This script is a version of the p4review.py script that is available on the Perforce Web site, but has been modified to use the server instance number. It relies on a configuration file in /p4/common/config, called p4_<instance>.p4review.cfg. On Windows, a driver called run_p4review.cmd, located in the same directory, allows you to run the review daemon through the Windows scheduler.

This is not required if you have installed Swarm which also performs notification functions and is easier for users to configure.

Location: /p4/common/bin

9.3.17. p4review2.py

Enhanced version of Section 9.3.16, "p4review.py"

1. Run p4review2.py --sample-config > p4review.conf

2. Edit the file p4review.conf

3. Add a crontab similar to this:

    ◦ * * * * python2.7 /path/to/p4review2.py -c /path/to/p4review.conf

Features:

- Prevent multiple copies running concurrently with a simple lock file.

- Logging support built-in.

- Takes command-line options.

- Configurable subject and email templates.

- Can (optionally) include URLs for changelists/jobs. Examples for P4Web included.

- 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 P4 auth (optional, not recommended!).

9.3.18. p4sanity_check.sh

This is a simple script to run:

- p4 set

- p4 info

- p4 changes -m 10

9.3.19. p4web_base

Very similar to Section 9.3.10, "p4d_base" but for the p4web service instance.

This product is very seldom used these days - since it has been replaced by Swarm.

9.3.20. p4dstate.sh

This 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).

Example:

```
./p4dstate.sh 1
```

9.3.21. ps_functions.sh

Common functions for using 'ps' to check on process ids. Not intended to be called directly but just to be sourced by other scripts.

```
get_pids ($exe)
```

Usage: Call with an exe name, e.g. /p4/1/bin/p4web_1

Example:

```
p4web_pids=$(get_pids $P4WEBBIN)
p4broker_pids=$(get_pids $P4BROKERBIN)
```

9.3.22. purge_revisions.sh

This 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.

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: ./purge_revisions.sh <SDP_INSTANCE> <P4_ARCHIVE_CONFIG> <P4_ARCHIVE_DEPOT> <P4_ARCHIVE_REPORT_MODE (Optional)> 4_ARCHIVE_TEXT (Optional)> <P4_PURGE_MODE (Optional)>

EXAMPLES:

Run from CLI that will archive files as defined in the config file

```
./purge_revisions.sh 1 /p4/common/config/p4_1.p4purge.cfg archive FALSE
```

Cron job that will will archive files as defined in the config file, including text files

```
30 0 * * * [ -e /p4/common/bin ] && /p4/common/bin/run_if_master.sh ${INSTANCE}
/p4/common/bin/purge_revisions.sh $NSTANCE} /p4/common/config/p4_1.p4purge.cfg archive
FALSE FALSE
```

9.3.23. recover_edge.sh

This 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 9.3.3, "edge_dump.sh", to the edge server before running this script. (Alternately, a full checkpoint from the master can be used so long as the edge server spec does not specify any filtering, e.g. does not

use ArchiveDataFilter.)

Then run this script on the Edge server with the instance number and full path of the master seed checkpoint as parameters.

Run example:

```
./recover_edge.sh 1 /p4/1/checkpoints/p4_1.edge_syd.seed.ckp.9188.gz
```

9.3.24. replica_cleanup.sh

This script performs the following actions for a replica:

- rotate logs
- remove old checkpoints and journals
- remove old logs

It is a convenenience script for occasional use.

9.3.25. replica_status.sh

This script is regularly run by crontab on a replica or edge (using Section 9.1.13, "run_if_replica.sh")

```
0 8 * * * [ -e /p4/common/bin ] && /p4/common/bin/run_if_replica.sh ${INSTANCE}
/p4/common/bin/replica_status.sh ${INSTANCE} > /dev/null
0 8 * * * [ -e /p4/common/bin ] && /p4/common/bin/run_if_edge.sh ${INSTANCE}
/p4/common/bin/replica_status.sh ${INSTANCE} > /dev/null
```

It performs a `p4 pull -lj` command on the replica to report current replication status, and emails this to the standard SDP administrator email on a daily basis. This is useful for monitoring purposes to detect replica lag or similar problems.

If you are using enhance monitoring such as p4prometheus then this script may not be required.

9.3.26. request_replica_checkpoint.sh

This script is intended to be run on a standby replica. It essentially just calls 'p4 admin checkpoint -Z' to reqeust a checkpoint and exits. The actual checkpoint is created on the next journal rotation on the master.

9.3.27. rotate_journal.sh

This script is a convenience script to perform the following actions for the specified instance (single parameter):

- rotate live journal
- replay it to the `offline_db`

- rotate logs files according to the settings in `p4_vars` for things like `KEEP_LOGS`

It is not often used.

9.3.28. 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.

9.3.29. update_limits.py

This is a Python script which is intended to be called from a crontab entry one per hour.

It ensures that all current users are added to the `limits` group. This makes it easy for an administrator to configure global limits on values such as MaxScanRows, MaxSearchResults etc. This can reduce load on a heavily loaded instance.

For more information:

- [Maximising Perforce Helix Core Performance](#)
- [Multiple MaxScanRows and similar values](#)

Chapter 10. SDP Package Contents

The directory structure of the SDP is shown below in Figure 1 - SDP Package Directory Structure. This includes all SDP files, including documentation and maintenance scripts. A subset of these files are deployed to server machines during the installation process.

```
sdp
    doc
    Server (Core SDP Files)
        Unix
            setup (unix specific setup)
            p4
                common
                    bin (Backup scripts, etc)
                        triggers (Example triggers)
                    config
                    etc
                        cron.d
                        init.d
                        lib
                        test
    setup (cross platform setup - typemap, configure, etc)
    test (automated test scripts)
```

Figure 1 - SDP Package Directory Structure

10.1. Volume Layout and Server Planning

Figure 2: SDP Runtime Structure and Volume Layout, viewed from the top down, displays a Perforce *application* administrator's view of the system, which shows how to navigate the directory structure to find databases, log files, and versioned files in the depots. Viewed from the bottom up, it displays a Perforce *system* administrator's view, emphasizing the physical volume where Perforce data is stored.

10.1.1. Memory and CPU

Make sure the server has enough memory to cache the **db.rev** database file and to prevent the server from paging during user queries. Maximum performance is obtained if the server has enough memory to keep all of the database files in memory.

Below are some approximate guidelines for allocating memory.

- 1.5 kilobyte of RAM per file stored in the server.
- 32 MB of RAM per user.

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 support/consulting for details).

10.1.2. Directory Structure Configuration Script for Linux/Unix

This script describes the steps performed by the mkdirs.sh script on Linux/Unix platforms. Please review this appendix carefully before running these steps manually. Assuming the three-volume configuration described in the Volume Layout and Hardware section are used, the following directories are created. The following examples are illustrated with "1" as the server instance number.

Directory	Remarks
`/p4`	Must be under root (`/`) on the OS volume
`/hxdepots/p4/1/bin`	Files in here are generated by the mkdirs.sh script.
`/hxdepots/p4/1/depots`	
`/hxdepots/p4/1/tmp`	
`/hxdepots/p4/common/config`	Contains p4_<instance>.vars file, e.g. `p4_1.vars`
`/hxdepots/p4/common/bin`	Files from `$SDP/Server/Unix/p4/common/bin`.
`/hxdepots/p4/common/etc`	Contains `init.d` and `cron.d`.
`/hxlogs/p4/1/logs/old`	
`/hxmetadata2/p4/1/db2`	Contains offline copy of main server databases (linked by `/p4/1/offline_db`.
`/hxmetadata1/p4/1/db1/save`	Used only during running of `recreate_db_checkpoint.sh` for extra redundancy.

Next, `mkdirs.sh` creates the following symlinks in the `/hxdepots/p4/1` directory:

Link source	Link target	Command
`/hxmetadata1/p4/1/db1`	`/p4/1/root`	`ln -s /hxmetadata1/p4/1/root`
`/hxmetadata2/p4/1/db2`	`/p4/1/offline_db`	`ln -s /hxmetadata1/p4/1/offline_db`
`/hxlogs/p4/1/logs`	`/p4/1/logs`	`ln -s /hxlogs/p4/1/logs`

Then these symlinks are created in the /p4 directory:

Link source	Link target	Command
`/hxdepots/p4/1`	`/p4/1`	`ln -s /hxdepots/p4/1 /p4/1`
`/hxdepots/p4/common`	`/p4/common`	`ln -s /hxdepots/p4/common /p4/common`

Next, mkdirs.sh renames the Perforce binaries to include version and build number, and then

creates appropriate symlinks.

The structure is shown in this example, illustrating values for two instances, with instance #1 using
Perforce 2018.1 and instance #2 using 2018.2.

In `/p4/common/bin`:

```
p4_2018.1.685046
p4_2018.1_bin -> p4_2018.1.685046
p4d_2018.1.685046
p4d_2018.1_bin -> p4d_2018.1.685046
p4d_2018.2.700949
p4_2018.2_bin -> p4_2018.2.700949
p4d_2018.2_bin -> p4d_2018.2.700949
p4_1_bin -> p4_2018.1_bin
p4d_1_bin -> p4d_2018.1_bin
p4_2_bin -> p4_2018.2_bin
p4d_2_bin -> p4d_2018.2_bin
```

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
```

Chapter 11. Frequently Asked Questions/Troubleshooting

This appendix lists common questions and problems encountered by SDP users. Do not hesitate to contact consulting@perforce.com if additional assistance is required.

11.1. Journal out of sequence

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 running the Section 9.1.6, "live_checkpoint.sh" script. Alternatively, if you know that the checkpoints created from previous runs of Section 9.1.4, "daily_checkpoint.sh" are correct, then restore the `offline_db` from the last known good checkpoint.

11.2. Unexpected end of file in replica daily sync

Check the start time and duration of the Section 9.1.4, "daily_checkpoint.sh" cron job on the master. If this overlaps with the start time of the [_sync_replica_sh] cron job on a replica, a truncated checkpoint may be rsync'd to the replica and replaying this will result in an error.

Adjust the replica's cronjob to start later to resolve this.

Default cron job times, as installed by the SDP are initial estimates, and should be adjusted to suit your production environment.