Perforce Helix Server Deployment Package (for Windows)

Perforce Professional Services

Version v2020.1, 2021-01-09

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. Windows SDP vs Unix SDP

The principles of the SDP are the same on both operating systems. The similarities are:

- Similar logical structure for file/directory layout (starting from `/p4` and `c:\p4` respectively)
- This logical structure can be mapped to flexible physical structure for desired performance and redundancy criteria
- Support for offline checkpointing (`root` vs `offline_db`) using an automated daily script
- Support for regular archive verification using an automated script
- Emailing of results for basic monitoring
- Some things like triggers written in Python or Perl are cross platform

The differences are:

- Unix scripts/tools are mainly written in Bash, whereas Windows mainly uses Powershell
- Windows Perforce Helix Core deployments tend to be simpler than Unix ones (fewer replicas etc), so there are more scripts for Unix SDP to manage replicas.

1.2. Downloading SDP

This is available: https://swarm.workshop.perforce.com/files/guest/perforce_software/sdp/downloads/sdp.Windows.zip

See Section 3.1, “Clean Installation” for where to put it after downloading.

Chapter 2. Configuring the Perforce Server

This chapter tells you how to configure a Perforce server machine and an instance of the Perforce Server. These topics are covered more fully in the Knowledge Base; this chapter 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 Perforce server instances. (In this guide, the term *server* refers to a Perforce server instance unless otherwise specified.) Each server instance is assigned a number. This guide uses instance number 1 in the example commands and procedures. Other instance numbers can be substituted as required.

This chapter also describes the general usage of SDP scripts and tools.

2.1. Volume Layout and Hardware

To ensure maximum data integrity and performance, use three different physical volumes for each server instance. Three volumes can be used for all instances hosted on one server machine, but using three volumes per instance reduces the chance of hardware failure affecting more than one instance.

While re recommend 3 volumes (drives), it is often practical to put all the files onto a single physical volume. We do NOT recommend the use of the C: drive (operating system root)!

- **Perforce metadata (database files):** Use the fastest volume possible, ideally RAID 1+0 on a dedicated controller with the maximum cache available on it. This volume is normally called `metadata`.
- **Journals and logs:** Use a fast volume, ideally RAID 1+0 on its own controller with the standard amount of cache on it. This volume is normally called `logs`. If a separate logs volume is not available, put the logs on the depotdata 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. This volume is the only volume that MUST be backed up (although we recommend also backing up `logs`). The backup scripts place the metadata snapshots on this volume. This volume can be backed up to tape or another long term backup device. This volume is normally called `depotdata`.

If three controllers are not available, put the logs and depotdata volumes on the same controller. Do not run anti-virus tools or back up tools against the metadata volume(s) or logs volume(s), because they can interfere with the operation of the Perforce server.

The SDP assumes (but does not require) the three volumes described above. It can easily be configured to use a single volume on which all data is stored.

View Figure 2: Volume Layout (below), 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.

Both Unix and Windows installation of the SDP now use symlinks (on Windows this is via the mklink tool).

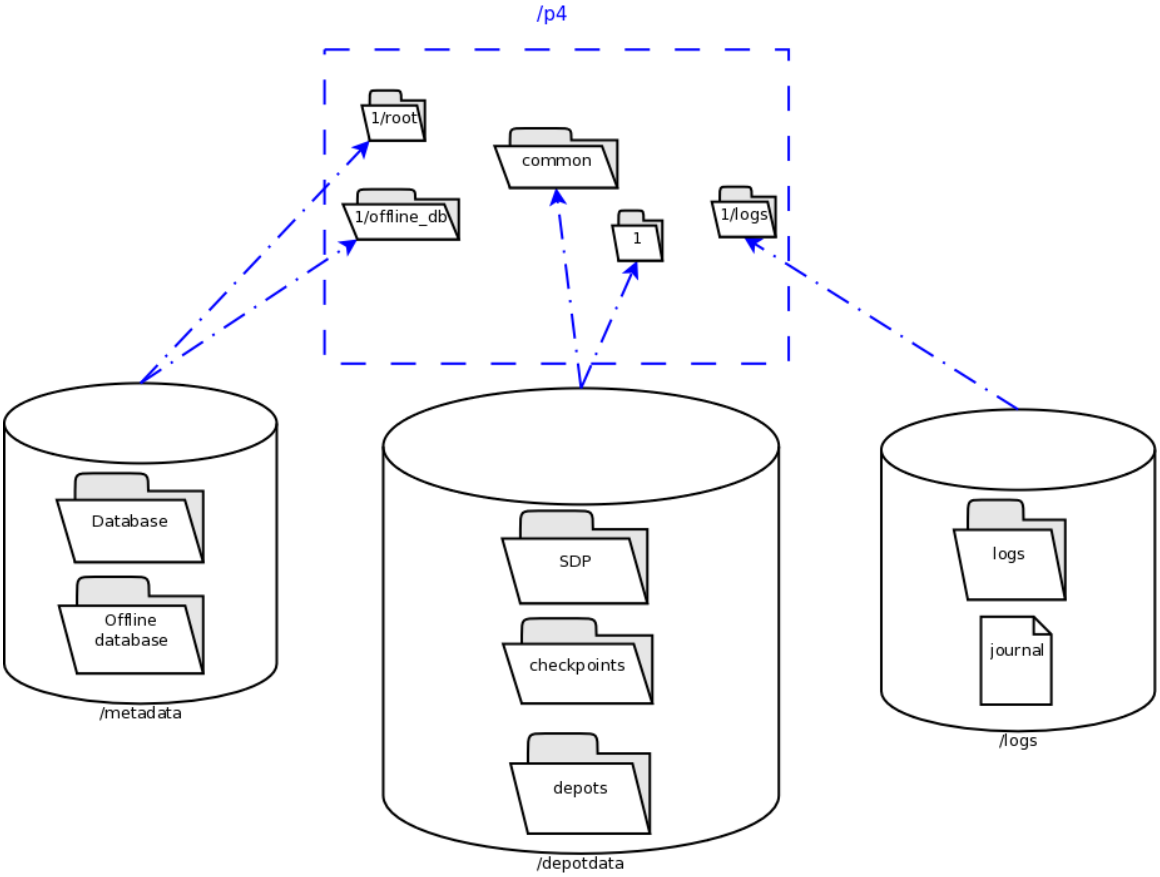


Figure 2: Volume Layout

The links are shown as <SYMLINKD> below on a Windows installation.

```
Directory of c:\p4
20/06/2020 15:05 <DIR> .
20/06/2020 15:05 <DIR> ..
20/06/2020 15:05 <SYMLINKD> common [f:\p4\common]
20/06/2020 15:05 <SYMLINKD> config [f:\p4\config]
20/06/2020 15:05 <SYMLINKD> 1 [f:\p4\1]
```



```
Directory of c:\p4\1
20/06/2020 15:05 <DIR> .
20/06/2020 15:05 <DIR> ..
20/06/2020 15:05 <DIR> bin
20/06/2020 15:05 <DIR> checkpoints
20/06/2020 15:05 <DIR> depots
20/06/2020 15:05 <SYMLINKD> logs [g:\p4\1\logs]
20/06/2020 15:05 <SYMLINKD> offline_db [e:\p4\1\offline_db]
20/06/2020 15:05 <SYMLINKD> root [e:\p4\M1\root]
20/06/2020 15:05 <DIR> ssl
20/06/2020 15:05 <DIR> tmp
```

2.2. Instance Names

Traditionally the SDP has used integers for instance names which show up in the paths above, for example C:\p4\ **1**\root.

However it is increasingly the case that alphanumeric names are used for instances, e.g. C:\p4\ ***Acme***\root. Commonly organizations strive to use a single Perforce instance, one logical data set, which may be replicated around the globe. Using a single instance optimize colloboration and simplifies code access for all development activity. When there is a single instance, the name '***1***' is as good as any. When there is more than one instance, e.g. if there are isolated silos of development activity, an alaphnumeric name may be more helpful than an integer for identifying the data set, such as ***Acme*** or perhaps ***LegacyApps***. Another instance is sometimes to develop and test things like Perforce trigger scripts before rolling them out to the live production instance, or to provide a standing internal training data set.

In any case it is worth thinking and planning your naming, particularly if you have multiple instances including replicas of different types and these are located on different hosts.

If you are using instance numbers, then an example configuration where there are 2 master server instances, each with a replica, might be:

Server hostname	Instance ID	Port
p4d-sfo-01	1	1666
sfo-p4d-01	2	2666
sfo-p4d-02	1	1666
sfo-p4d-02	2	2666

For consistency, instances with same ID should refer to the same logical data set, they just run on different machines.

Alternatively, alphanumeric names can be clearer and easier:

Server hostname	Instance ID	Port
sfo-p4d-01	Acme	5000

Server hostname	Instance ID	Port
sfo-p4d-01	Test	5999
sfo-p4d-02	Acme	5000
sfo-p4d-02	Test	5999

Some sites apply a convention to the port number to identify whether the P4PORT value is that of a master server, a broker, replica, edge server, or proxy. In such cases the first digit is reserved to identify the instance, and the remaining 3 digits identify the target service, e.g. 666 for a broker, 999 for a master server, 668 for a proxy.

Host naming conventions vary from site to site, and often have local naming preferences or constraints. These examples the the code of the nearest major airport, sfo in this case, as a location code. Using location in the hostname is merely an example of a site preference, not necessarily a best practice.

End user **P4PORT** values typically do not reference the actual machine names. Instead they reference an alias, e.g. perforce or sfo-p4d (without the -01). This helps make failover operations more transparent.

Chapter 3. Installing the Perforce Server and the SDP

This chapter tells you how to install a Perforce server instance in the SDP framework. For more details about server installation, refer to the Perforce System Administrator's Guide.

Many companies use a single Perforce Server to manage their files, while others use multiple servers. The choice depends on network topology, the geographic distribution of work, and the relationships among the files being managed. If multiple servers are run, assign each instance a number and use that number as part of the name assigned to depots, to make the relationship of depots and servers obvious. See the discussion above on Instance Names.

3.1. Clean Installation

In this section we describe the server and SDP installation process on Windows. The process consists of:

1. Initial setup of the file system and configuration files.
2. Running the SDP configuration script.
3. Starting the server and performing initial configuration.

3.1.1. Pre-requisites

The following are required (details mentioned below):

- Administrator account on the server
- Python installed (see below)
- Perforce Helix Core executables (p4.exe/p4d.exe - see below)
- Powershell 5.x or greater (default on Windows 10 or Windows Server 2016+)

Optional (but recommended):

- Perforce Helix Visual client (P4V - optional but very useful, together with P4Admin - the Admin tool)
- An editor (Notepad will do, but Download Notepad++)
- GOW (Gnu on Windows) - optional but very useful for parsing log files etc.

3.1.2. Configuring Powershell

The scripts now use Powershell rather than .BAT files due to improved error handling and options, and code re-use (via a single included module rather than duplication of functionality in every script). This also allows us to keep the scripts more closely aligned with the functionality of the Unix scripts.

It is important to enable local scripts to be run. The following command must be run within an

Powershell Administrator prompt:

```
get-executionpolicy
```

The result needs to be either `RemoteSigned` or `Unrestricted`. If not then set it as below.

For Windows 10, Windows Server 2016 or later, run the following commands as Administrator:

- *x86*
  Open `C:\Windows\SysWOW64\cmd.exe`
  Run the command:

  ```
  powershell Set-ExecutionPolicy RemoteSigned
  ```

- *x64*
  Open `C:\Windows\system32\cmd.exe`
  Run the command:

  ```
  powershell Set-ExecutionPolicy RemoteSigned
  ```

Use `get-executionpolicy` to check the policy has been updated. You may need to ensure that the various scripts are not "blocked" - right click in Windows Explorer and check Properties options.

3.1.3. Initial setup

Prior to installing the Perforce server, perform the following steps.

1. Mount the volumes for the three-volume configuration described in Volume Layout and Hardware. The procedure assumes the drives are mapped as follows:
   - Metadata on `e:`
   - Depotdata on `f:`
   - Logs on `g:`

If you do not have a logs volume, put the logs on the depot data volume. If you have only a single data volume, e.g. `d:` then set all the values to that volume.

2. Copy the SDP to the `f:\sdp` directory (let us call this `%SDP%`).

It is likely that Windows will have blocked the various scripts and files for security reasons. It is important to run the following command (in Powershell window as Administrator)

```
dir -Path f:\sdp -Recurse | Unblock-File
```

3. Customize the following for your environment. It requires you to identify the master server and all replicas that we need to setup for the SDP, including instance names, hostnames, etc. This information is all in a single file:

```
%SDP%\Server\Windows\setup\sdp_master_config.ini
```

4. Download and install Python, e.g. from www.python.org. We use Python 2.7.x (latest) and 3.7.x (latest). 64-bit version is fine. Typically we install to default dir, e.g. c:\python27. For initial installation we only require base Python. For subsequent scripting you may wish to install P4Python (e.g. using pip).
5. Other tools we find useful: Notepad++ and GOW (Gnu on Windows - Unix command line utilities such as wc, head, tail). These are recommended but not strictly required.
6. Download to directory `%SDP%\Server\Windows\setup` the desired release of p4.exe and p4d.exe. For example, for Helix Core for release 2020.1 on 64 bit Windows, use this URL:

```
http://ftp.perforce.com/perforce/r20.1/bin.ntx64
```

   From that directory listing, select `p4.exe` and then `p4d.exe` to download each of those files. If you are using 32 bit Windows (unusual these days), substitute `bin.ntx86` for `bin.ntx64` in the URL above. The following works within Powershell

```
Invoke-WebRequest "http://ftp.perforce.com/perforce/r20.1/bin.ntx64/p4.exe"
-OutFile "p4.exe"
Invoke-WebRequest "http://ftp.perforce.com/perforce/r20.1/bin.ntx64/p4d.exe"
-OutFile "p4d.exe"
```

3.1.4. Running Configuration script

The `create_env.py` script, available in `%SDP%\Server\Windows\setup`, sets up the basic directory structure used by the SDP. It creates .bat files to register the Perforce service as Windows services. It parses and validates the `sdp_master_config.ini` file in the same directory.

You need to customize this `sdp_master_config.ini` file. It contains lots of comments as to how to set the various configuration values.

The following shows a sample config after editing:

```
[DEFAULT]
SDP_P4SUPERUSER=perforce
SDP_P4SUPERUSER_PASSWORD=SomeRandomPassword
ADMIN_PASS_FILENAME=adminpass.txt

mailfrom=perforce@example.com
maillist=p4ra@example.com
mailhost=mail.example.com
mailhostport=25

EMAIL_PASS_FILENAME=emailpass.txt
EMAIL_PASSWORD=

KEEPCKPS=10
KEEPLOGS=20
LIMIT_ONE_DAILY_CHECKPOINT=false
SDP_GLOBAL_ROOT=c:

# Assuming output of `hostname` is this value, and we are using instance `1`
[1:perforce-svr-01]
SDP_SERVERID=Master
SDP_SERVICE_TYPE=standard
SDP_P4PORT_NUMBER=1666
# Everything on D: drive
METADATA_ROOT=D:
DEPOTDATA_ROOT=D:
LOGDATA_ROOT=D:
REMOTE_DEPOTDATA_ROOT=
```

Review the contents of `template_configure_new_server.bat` file which defines the recommended default configurable values for any server, and make any desired changes. This file will be parsed and used to create instance specific configuration files.

After updating the configuration file, run `create_env.py` from the same directory.

You must run this command from a CMD window which has **administrator** rights.

```
cd %SDP%\Server\Windows\setup
```

Edit and save changes:

```
notepad sdp_master_config.ini
```

Run the command to create the environment (by default it looks for the config file sdp_master_config.ini but this can be changed with -c option):

```
Create_env.py
```

The output will look something like this:

```
D:\sdp\Server\Windows\setup>create_env.py
INFO: Found the following sections: ['1:EC2AMAZ-LJ68A4I']
INFO: Config file written: sdp_config.ini
INFO: The following directories/links would be created with the -y/--yes flag
INFO: Creating target dir 'c:\p4'
INFO: Creating target dir 'D:\p4\1'
INFO: Creating target dir 'D:\p4\1'
INFO: Creating link 'c:\p4\1' to 'D:\p4\1'
INFO: Creating target dir 'D:\p4\common'
INFO: Creating link 'c:\p4\common' to 'D:\p4\common'
INFO: Creating target dir 'D:\p4\config'
INFO: Creating link 'c:\p4\config' to 'D:\p4\config'
INFO: Creating target dir 'c:\p4\common\bin'
INFO: Creating target dir 'c:\p4\common\bin\triggers'
INFO: Creating target dir 'c:\p4\1\bin'
INFO: Creating target dir 'c:\p4\1\tmp'
INFO: Creating target dir 'c:\p4\1\depots'
INFO: Creating target dir 'c:\p4\1\checkpoints'
INFO: Creating target dir 'c:\p4\1\ssl'
INFO: Creating target dir 'D:\p4\1\root'
INFO: Creating link 'c:\p4\1\root' to 'D:\p4\1\root'
INFO: Creating target dir 'c:\p4\1\root\save'
INFO: Creating target dir 'D:\p4\1\offline_db'
INFO: Creating link 'c:\p4\1\offline_db' to 'D:\p4\1\offline_db'
INFO: Creating target dir 'D:\p4\1\logs'
INFO: Creating link 'c:\p4\1\logs' to 'D:\p4\1\logs'
INFO: Copying 'D:\sdp\Server\Windows\p4\common\bin\create-filtered-edge-
checkpoint.ps1' to 'c:\p4\common\bin\create-filtered-edge-checkpoint.ps1'
INFO: Copying 'D:\sdp\Server\Windows\p4\common\bin\create-offline-db-from-
checkpoint.bat' to 'c:\p4\common\bin\create-offline-db-from-checkpoint.bat'
INFO: Copying 'D:\sdp\Server\Windows\p4\common\bin\create-offline-db-from-
checkpoint.ps1' to 'c:\p4\common\bin\create-offline-db-from-checkpoint.ps1'
:
INFO: Copying 'D:\sdp\Server\Windows\setup\p4.exe' to 'c:\p4\1\bin'
INFO: Copying 'D:\sdp\Server\Windows\setup\p4d.exe' to 'c:\p4\1\bin'
INFO: Copying 'D:\sdp\Server\Windows\setup\p4d.exe' to 'c:\p4\1\bin\p4s.exe'
INFO: Copying 'D:\sdp\Server\Windows\setup\sdp_config.ini' to 'c:\p4\config'
INFO: Copying 'D:\sdp\Server\Windows\setup\Master_server.id' to
'c:\p4\1\root\server.id'
INFO: Creating instance bat file 'c:\p4\1\bin\daily-backup.bat'
INFO: Creating instance bat file 'c:\p4\1\bin\p4verify.bat'
INFO: Creating instance bat file 'c:\p4\1\bin\replica-status.bat'
INFO: Creating service configure commands on 'ec2amaz-lj68a4i' for instance '1' in
install_services_ec2amaz-lj68a4i.bat
```

```
The following commands have been created - but you are in report mode so no
directories have been created
    install_services_ec2amaz-lj68a4i.bat
    configure_Master.bat
You will also need to seed the replicas from a checkpoint and run the appropriate
commands on those machines
INFO: Running in reporting mode: use -y or --yes to perform actions.
```

If the output looks correct then re-run the script with `-y` parameter to actually perform the copying of files and creation of directories and links.

```
create_env.py -y
```

3.1.5. sdp_config.ini

This file is written to `c:\p4\common\bin`. It will look something like this(note the value `EC2AMAZ-LJ68A4I` is the output of the `hostname` command):

```
[1:EC2AMAZ-LJ68A4I]
p4port=EC2AMAZ-LJ68A4I:1777
sdp_serverid=Master
sdp_p4serviceuser=
sdp_global_root=c:
sdp_p4superuser=perforce
admin_pass_filename=adminpass.txt
email_pass_filename=emailpass.txt
mailfrom=perforce@example.com
maillist=p4ra@example.com
mailhost=mail.example.com
mailhostport=25
python=None
remote_depotdata_root=
keepckps=10
keeplogs=20
limit_one_daily_checkpoint=false
remote_sdp_instance=
p4target=
```

The above are the values for a single master/commit server.

If you configure a replica then these fields should be set to appropriate values:

```
[1:EC2AMAZ-REPLICA]
:
sdp_p4serviceuser=svc_p4d_ha_aws
:
remote_depotdata_root=\\EC2AMAZ-LJ68A4I\d$
remote_sdp_instance=1
p4target=EC2AMAZ-LJ68A4I:1777
```

3.1.6. Installing service(s)

The above command will create a couple of files in that directory. The first is `install_services_<hostname>.bat`, so on a machine where the hostname is `svrp4master`, it will be `install_services_svrp4master.bat`

Validate the contents of this file and run it if it looks appropriate - this installs the service(s) with appropriate parameters. Please note that it is specific to the **hostname** that you specified inside `sdp_master_config.ini` - so it will only run on the correct host server. It should look something like:

```
D:\sdp\Server\Windows\setup>install_services_ec2amaz-lj68a4i.bat
D:\sdp\Server\Windows\setup>c:\p4\common\bin\instsrv.exe p4_1 "c:\p4\1\bin\p4s.exe"
The service was successfuly added!
```

```
Make sure that you go into the Control Panel and use
the Services applet to change the Account Name and
Password that this newly installed service will use
for its Security Context.
```

```
D:\sdp\Server\Windows\setup>c:\p4\1\bin\p4.exe set -S p4_1 P4ROOT=c:\p4\1\root
D:\sdp\Server\Windows\setup>c:\p4\1\bin\p4.exe set -S p4_1
P4JOURNAL=c:\p4\1\logs\journal
D:\sdp\Server\Windows\setup>c:\p4\1\bin\p4.exe set -S p4_1 P4NAME=Master
D:\sdp\Server\Windows\setup>c:\p4\1\bin\p4.exe set -S p4_1 P4PORT=1777
D:\sdp\Server\Windows\setup>c:\p4\1\bin\p4.exe set -S p4_1
P4LOG=c:\p4\1\logs\Master.log
```

Note that if you have defined multiple instances in `sdp_master_config.ini` to run on this same hostname, then they will all be installed by this .bat file.

3.1.7. Start the server to test

Having installed the service, we now test that it will start: `c:\p4\common\bin\svcinst start -n p4_<instance name>`, e.g.

```
c:\p4\common\bin\svcinst start -n p4_1
```

Or

```
c:\p4\common\bin\svcinst start -n p4_Master
```

If the service fails to start, then examine the log file for the reason (e.g. missing license file) in `c:\p4\<instance>\logs`.

Ensure the server is running (specify appropriate port):

```
p4 -p 1666 info
```

Use `c:\p4\common\bin\svcinst stop -n p4_<instance>` to stop the service if required.

3.1.8. Applying configurables to the server instance

For each instance defined in `sdp_master_config.ini`, a configuration .bat file will be created, called `configure_<instance>.bat`, so for instance master, it will be `configure_master.bat`.

Review the contents of the file and make any desired changes.

You will only be able to run the .bat file if you have started the server instance as per previous section.

If an instance is a replica (or similar), then you should apply the configurables to the master server and then checkpoint it before seeding the replica - see the Distributing Perforce guide.

3.1.9. Configuring the server

To configure the server, perform the following steps:

1. Make sure your server is running (specify appropriate port below):

   ```
   p4 -p 1666 info
   ```

2. Create your Perforce administrator account within the Perforce repository, using the user name and password specified in `sdp_master_config.ini`.
3. Optional. To create a Perforce stream depot called PerforceSDP and load the SDP, issue the following commands:

   ```
   p4 depot -t stream -o PerforceSDP | p4 depot -i
   p4 stream -t mainline -o //PerforceSDP/main | p4 stream -i
   cd /d C:\sdp
   p4 client -S //Perforce/main -o PerforceSDP_ws | p4 client -i
   p4 -c PerforceSDP_ws reconcile
   p4 -c PerforceSDP_ws submit -d "Added SDP."
   ```

4. Optional. To create a Perforce spec depot, issue the following commands:

```
p4 depot -t spec -o spec | p4 depot -i
```

   Then add the following to the Protections table, near the bottom (about super user entries), to hide specs which could have security implications:

```
list user * * -//spec/protect.p4s
list user * * -//spec/triggers.p4s
```

   Then update specs in the depot with this command:

```
p4 admin updatespecdepot -a
```

5. Optional. To create an unload depot, issue the following command:

```
p4 depot -t unload -o unload | p4 depot -i
```

6. Optional. To delete the default Perforce depot naned depot, issue the following command: `p4 depot -d depot`. Create one or more depots as required to store your files, following your site's directory naming conventions.

3.1.10. Verifying your server installation

To verify your installation, perform these steps:

1. Issue the p4 info command, after setting appropriate environment variables. If the server is running, it will display details about its settings.
2. Create a client workspace and verify that it is archived in the spec depot and written to the `c:\p4\1\depots\specs\client` (assuming instance `1`) directory.
3. Add a file to the server and verify that the archive file gets created in the corresponding directory under `c:\p4\1\depots`.

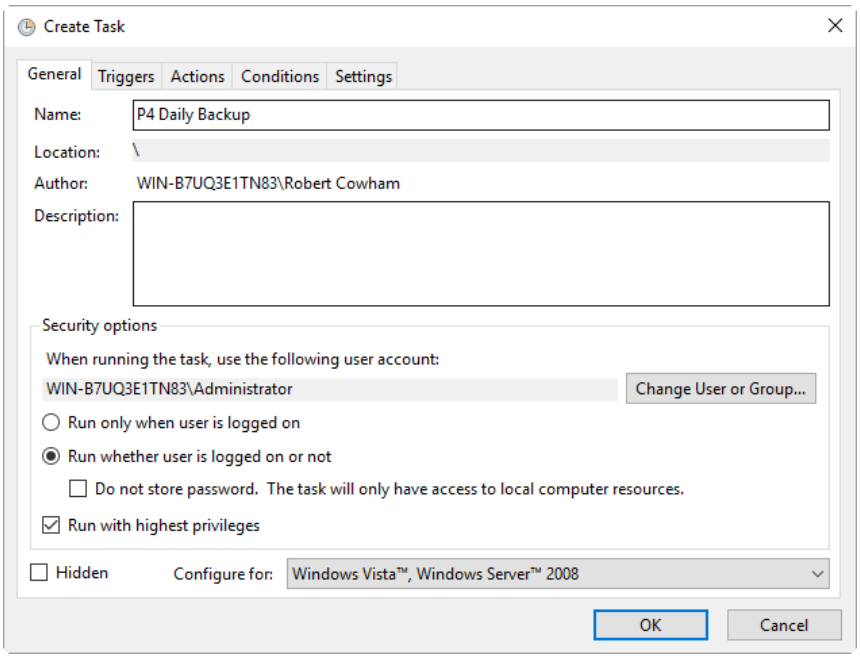
3.1.11. Scheduling maintenance scripts

In Windows 2012 or later you should use the Task Scheduler. We recommend that you create a folder called Perforce at the top level in which to create your tasks (otherwise they can be hard to find when you next look in Task scheduler!).

Note that the "schtasks" command can be useful from command line, or "taskschd.msc" to get the control panel equivalent.

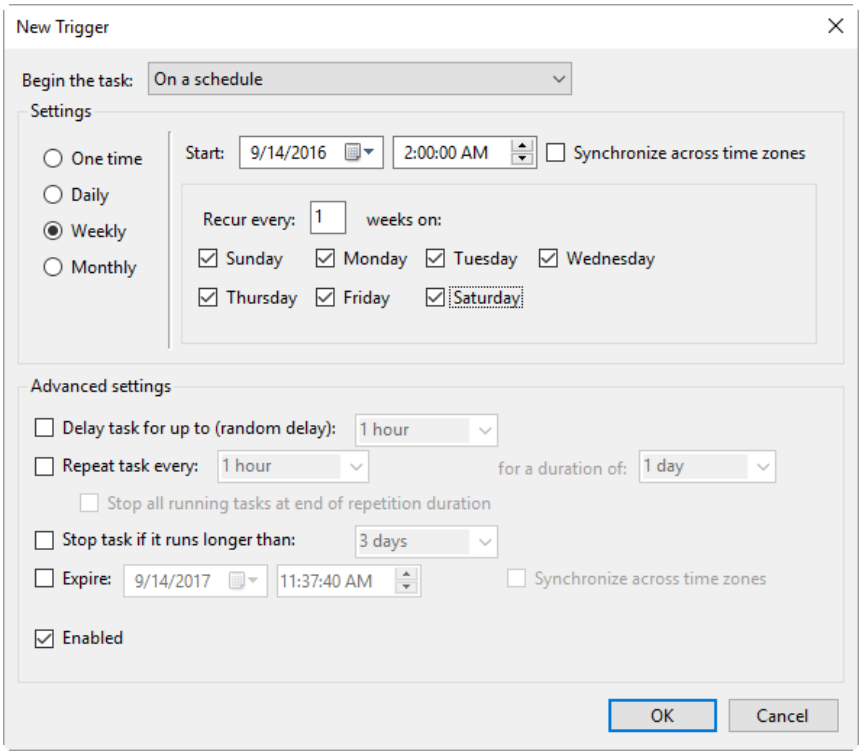
The recommendation is to run `daily-backup.bat` every day at say 2:00 am (or similar time).

Task Basics



Trigger screen

Set the time to run like this:



Action

Program: `c:\p4\master\bin\daily-backup.bat`

Where `master` is your instance name.

3.1.12. Saving your configuration files in Perforce

It is sensible to create a Perforce workspace and to store the configuration files in Perforce.

Typically the depot root might be something like `//perforce/sdp`. If you have many machines, then you might use `//perforce/sdp/<machine>` using either a physical or a logical name for the machine.

A typical workspace view (e.g. for workspace called p4admin.sdp and for instance master), might be:

```
Root: c:\p4
View:
    //perforce/sdp/p4/*1*/bin/... //p4admin.sdp/*1*/bin/...
    //perforce/sdp/p4/common/bin/... //p4admin.sdp/common/bin/...
    //perforce/sdp/p4/config/... //p4admin.sdp/config/...
```

You would have appropriate workspaces for each machine, and appropriate lines for each instance on that machine.

3.1.13. Archiving configuration files

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

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

3.1.14. Configuring a New Instance on an existing machine

It is possible to add a new instance to an existing machine.

Edit the `sdp_master_config.ini` and add a new section for the new instance.

The run `create_env.py`, specifying to just create the new instance.

```
cd %SDP%\Server\Windows\setup
```

```
notepad sdp_master_config.ini
```

```
create_env.py -c sdp_master_config.ini --instance Replica2
```

If the output looks correct then re-run the script with `-y` parameter to actually perform the copying of files and creation of directories and links.

3.1.15. Upgrading an existing (non SDP) Windows installation

The easiest way to upgrade a service instance is:

1. Create new `sdp_master_config.ini` file to describe the existing installations.
2. Run `create_env.py` to create the new environment
3. Run `install_services_<hostname>.bat` to create new services.
    a. Stop existing services
    b. Manually move the following files from their existing to new locations:
        i. `db.*` files
        ii. `license`
        iii. log file(s)
        iv. `journal`
        v. checkpoints and archived journals
    c. Start new service and check it runs successfully
    d. Adjust the depot root paths (with 2014.1 or greater use the configurable `server.depot.root`, otherwise manually edit depot specs and install the appropriate trigger for new depot specs)

Simple reporting commands to compare before/after include:

```
p4 changes -m20 -l -t > changes.txt
p4 depots > depots.txt
p4 verify -q //...@yyyy/mm/dd,#head (specifying a few days before the cutover)
```

3.1.16. Upgrading an older Windows SDP installation

Older versions of the Windows SDP (pre June 2014) stored configuration values for each instance in a p4env.bat file within the `p4\common\bat` directory.

They also didn't link all directories from `c:\p4`, but instead used drives such as `E:`, `F:` and `G:` and paths on those drives.

The easiest way to upgrade (most of the work can be done without stopping the service) is:

1. Ensure that existing instance files are checked in to Perforce (`instance\bin` and `common\bin` files), for example in workspace p4admin.sdp.orig (use a root directory of %DEPOTDATA% - see next step).
2. Extract existing values from `p4env.bat` such as mailfrom, mailhost, mailto, and also METADATA, LOGDATA and DEPOTDATA
3. Edit `sdp_master_config.ini` and set the appropriate values using extracted ones, and appropriate instance specific values.
4. Set the values for `METADATA_ROOT`, `DEPOTDATA_ROOT` and `LOGDATA_ROOT` to the same (dummy) value, e.g. `c:\p4assets`

5. Run 'create_env.py` to generate the new structure.
6. Manually edit `c:\p4assets\p4\config\sdp_config.ini` and set the ROOT values to the existing values taken from step 2.
7. Using a different but similar workspace p4admin.sdp.new, which has a root directory of `c:\p4`, run `p4 sync -k`, then do a `p4 reconcile` to identify all the changed files - this will include most of the .bat files, but it shouldn't include `p4d.exe` or `p4s.exe` as we are not updating these files.
8. Submit the new changes.
9. In workspace `p4admin.sdp.orig`, carefully check the updated files that need to be synced (recommend you review diffs one by one), and then sync them.
10. Manually remove and recreate the links (using `del` and `mklink /d`) for directories under `c:\p4` so that they point to the existing directories on `e:`, `f:` or `g:` (the original DEPOTDATA).
11. Review existing configurables and adjust as appropriate.
12. Setup the scheduled tasks for daily/weekly backup and verify as appropriate. Validate that the daily backup works (typically wait until the next day)
13. At an appropriate point, stop the existing service, adjust the service paths to use the new paths starting from `c:\p4`.

3.1.17. 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 your typemap to override Perforce's default behavior.

   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`

5. For large, generated text files (e.g. postscript files), assign the text+C file type, to avoid causing server memory issues.

Perforce provides most IT required password management practices internally. It is recommend to use internal passwords over LDAP/AD to avoid exposing LDAP/AD passwords to the Perforce admin via the auth trigger.

3.2. General SDP Usage

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

Most tools reside in c:\p4\common\bin. The directory c:\p4*<instance>*\bin contains scripts and executables that are specific to a server instance, such as the p4.exe client. The scripts in c:\p4*<instance>*\bin generally set the environment for an instance correctly, then invoke the corresponding script in c:\p4\common\bin.

Run important administrative commands using the scripts in c:\p4*<instance>*\bin, when available. Then, use the p4.exe executable located in c:\p4*<instance>*\bin.

Below are some usage examples for instance 1 or instance master.

Example	**Remarks**
c:\p4\common\bin\live-checkpoint.ps1 1	Take a checkpoint of the live database on instance 1
c:\p4\common\bin\daily-backup.ps1 master	A daily checkpoint of the *master* instance.

Chapter 4. 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 depotdata volume.

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

4.1. Typical Backup Procedure

The SDP's maintenance scripts, run as *cron* tasks on Unix/Linux or as Windows *scheduled tasks*, periodically back up the metadata. The weekly sequence is described below.

Seven nights a week, perform the following tasks.

1. Rotate/truncate the active journal.
2. Replay the journal to the offline database. (Refer to Figure 2: 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 depots.

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

4.2. Planning for HA and DR

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

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

See also Replica types and use cases

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

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

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

Unplanned covers risks you are seeking to mitigate with failover:

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

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

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

- taking a backup once per 24 hours and requiring maybe an hour or two to restore it. Thus you might lose up to 24 hours of work for an unplanned failure, and require several hours to restore.
- having a high availability replica which is a mirror of the server hardware and ready to take over within minutes if required

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

Typically we would define:

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

4.2.1. Further Resources

- High Reliability Solutions

4.2.2. Creating a Failover Replica for Commit or Edge Server

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

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

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

4.2.3. What is a Failover Replica?

As of 2018.2 release, p4d supports a `p4 failover` command that performs a failover to a `standby` replica (i.e. a replica with `Services:` field value set to `standby` or `forwarding-standby`). Such a replica performs a `journalcopy` replication of metadata, with a local pull thread to update its `db.*` files.

See also: Configuring a Helix Core Standby.

4.2.4. Mandatory vs Non-mandatory Standbys

You can modify the server spec of a `standby` replica to make it `mandatory`.

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

Thus downstream servers can simply be re-directed to point to the standby and will carry on working without problems.

If a server which is marked as `mandatory` goes offline for any reason, the replication to other replicas will stop replicating. In this scenario, the server spec of the replica can be changed to `nomandatory`, and then replication will immediately resume (so long as the replication has not been offline for too long, typically several days or weeks depending on the KEEPJNLS setting).

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

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

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

4.2.5. Server host naming conventions

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

- Use a name that does not indicate switchable roles, e.g. don't indicate in the name whether a host is a master/primary or backup, or edge server and its backup. This might otherwise lead to confusion once you have performed a failover and the host name is no longer appropriate.
- Use names ending numeric designators, e.g. -01 or -05. The goal is to avoid being in a post-failover situation where a machine with `master` or `primary` is actually the backup. Also, the assumption is that host names will never need to change.
- While you don't want switchable roles baked into the hostname, you can have static roles, e.g. use p4d vs. p4p in the host name (as those generally don't change). The p4d could be primary, standby, edge, edge's standby (switchable roles).
- Using a short geographic site is sometimes helpful/desirable. If used, use the same site tag used in the ServerID, e.g. aus.
- Using a short tag to indicate the major OS version is **sometimes** helpful/desirable, eg. c7 for CentOS 7, or r8 for RHEL 8. This is based on the idea that when the major OS is upgraded, you either move to new hardware, or change the host name (an exception to the rule above about never changing the hostname). This option maybe overkill for many sites.
- End users should reference a DNS name that may include the site tag, but would exclude the number, OS indicator, and server type (`p4d`/`p4p`/`p4broker`), replacing all that with just `perforce` or optionally just `p4`. General idea is that users needn't be bothered by under-the-covers tech of whether something is a proxy or replica.
- For edge servers, it is advisable to include `edge` in both the host and DNS name, as users and admins needs to be aware of the functional differences due to a server being an edge server.

Examples:

- `p4d-aus-r7-03`, a master in Austin on RHEL 7, pointed to by a DNS name like `p4-aus`.

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

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

4.2.6. Pre-requisites for Failover

These are vital as part of your planning.

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

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

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

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

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

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

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

4.3. Full One-Way Replication

Perforce supports a full one-way replication of data from a master server to a replica, including versioned files. The p4 pull command is the replication mechanism, and a replica server can be configured to know it is a replica and use the replication command. The p4 pull mechanism

requires very little configuration and no additional scripting. As this replication mechanism is simple and effective, we recommend it as the preferred replication technique. Replica servers can also be configured to only contain metadata, which can be useful for reporting or offline checkpointing purposes. See the Distributing Perforce Guide for details on setting up replica servers.

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

4.3.1. Replication Setup

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

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

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

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

4.4. Replication Setup Details

Note, it is required that you set P4TICKETS for the service and for the users on the machine to a common location for replication to work. To set this up, run the following on both the master and the replica:

```
p4 set -s P4TICKETS=c:\p4\1\p4tickets.txt
p4 set -S p4_1 P4TICKETS=c:\p4\1\p4tickets.txt
```

Once the machine and SDP install is in place, you need to configure the master server for replication. We will assume the following for the setup:

The replica name will be `p4d_ha_lon`, the service user name is `svc_rp4d_ha_lon`, and the master server's name is `master`, and the metadata volume is `e:`, the depotdata volume is `f:`, and the logs volume is `g:`. You will run the following commands on the master server:

```
p4 configure set P4TICKETS=c:\p4\1\p4tickets.txt
p4 configure set p4d_ha_lon#P4PORT=1667
p4 configure set p4d_ha_lon#P4TARGET=master:1667
p4 configure set p4d_ha_lon#journalPrefix=c:\p4\1\checkpoints\p4_1
p4 configure set p4d_ha_lon#server=3
p4 configure set "p4d_ha_lon#startup.1=pull -i 1"
p4 configure set "p4d_ha_lon#startup.2=pull -u -i 1"
p4 configure set "p4d_ha_lon#startup.3=pull -u -i 1"
p4 configure set "p4d_ha_lon#startup.4=pull -u -i 1"
p4 configure set "p4d_ha_lon#startup.5=pull -u -i 1"
p4 configure set "p4d_ha_lon#db.replication=readonly"
p4 configure set "p4d_ha_lon#lbr.replication=readonly"
p4 configure set p4d_ha_lon#serviceUser=svc_p4d_ha_lon
```

The following commands will also need to be run:

- `p4 user -f svc_p4d_ha_lon` (You need to add the `Type: service` field to the user form before saving)
- `p4 passwd svc_p4d_ha_lon` (Set the service user's password)
- `p4 group ServiceUsers` (Add the service user to the `Users:` section and set the `Timeout:` to unlimited.)
- `p4 protect` (Give `super` rights to the group `ServiceUsers` to `//…`)

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

```
c:\p4\common\bin\daily-backup.ps1 1
```

When the checkpoint finishes, copy the checkpoint plus the versioned files over to the replica server. You can use xcopy or something like robocopy for this step.

```
xcopy c:\p4\1\checkpoints\p4_1.ckp.###.gz replica_f_drive:\p4\1\checkpoints
```

```
xcopy c:\p4\1\depots replica_f_drive:\p4\1\depots /S
```

(# is the checkpoint number created by the daily backup)

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

```
c:\p4\1\bin\p4d -r c:\p4\1\root -jr -z c:\p4\1\checkpoints\p4_1.ckp.###.gz
c:\p4\1\bin\p4 -p master:1667 -u svc_p4d_ha_lon login (enter the service user's
password)
c:\p4\common\bin\svcinst start -n p4_1
```

Now, you check the log on the master server (`c:\p4\1\logs\log`) to look for the rmt-Journal entries that show you the replication is running. If you see those entries, then you can make some changes on the master server, and then go to the replica server and check to see that they changes were replicated across. For example, you can submit a change to the master server, then go to the replica server and check to see that the change was replicated over to the replica by running p4 describe on the changelist against the replica server.

The final steps for setting up the replica server are to set up the task scheduler to run the replica sync scripts. This has to be done via task scheduler running as a regular AD user so that the scripts can access the network in order to get to the drives on the replica machine.

You need to configure a task to run `c:\p4\common\bin\sync-replica.ps1 <instance>` every day. The task should be set up to run after the master server finishes running `daily-backup.ps1`. Be sure to give it some buffer for the length of time it takes the master to run that script is likely to become gradually longer over time.

4.5. 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 depotdata volume using your normal backup utilities.

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

4.5.1. Recovering from a checkpoint and journal(s)

The checkpoint files are stored in the `c:\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 c:\p4\<instance>\checkpoints, and the journal file from c:\p4\<instance>\logs.

To recover the server database manually, perform the following steps from the root directory of the server (`c:\p4\<instance>\root`). In the examples below we assume the <instance> is 1.

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

```
c:\p4\1\bin\p4 admin stop
```

2. Delete the old database files in `c:\p4\1\root\save` directory (note there may not be any files there as they will typically be cleared out **after** successful completion of the previous invocation of the recovery process - see below).
3. Move the live database files (db.*) to the save directory.
4. Use the following command to restore from the most recent checkpoint.

```
c:\p4\1\bin\p4d -r c:\p4\1\root -jr -z
```

```
c:\p4\1\checkpoints\p4_1.ckp.<most recent #>.gz
```

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

```
c:\p4\1\bin\p4d -r c:\p4\1\root -jr c:\p4\1\logs\journal
```

6. Restart your Perforce server.

If the Perforce service starts without errors, delete the old database files from `c:\p4\1\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 back up and recovery, refer to the Perforce System Administrator's Guide.

4.5.2. 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 depotdata 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 depotdata volume from your backup tape.
2. As a super-user, reinstall and enable the Windows services that run the Perforce instance.
3. Find the last available checkpoint, under `c:\p4\<instance>\checkpoints`.
4. Recover the latest checkpoint by running:

```
c:\p4\<instance>\bin\p4d_<instance> -r c:\p4\<instance>\root -jr -z _last_ckp_file_
```

5. Recover the checkpoint (as shown in the preceding step) into the offline_db directory rather than the root directory.

c:\p4\<instance>\bin\p4d_<instance> -r c:\p4\<instance>\offline_db -jr -z *last_ckp_file*

6. Reinstall the Perforce server license to the server root directory.
7. Start the Perforce service.
8. Verify that the server instance is running.
9. Reinstall the server crontab or scheduled tasks.
10. Perform any other initial server machine configuration.
11. Verify the database and versioned files by running the p4verify 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.

4.5.3. Failover to a replicated standby machine

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

Chapter 5. Server Maintenance

This section describes typical maintenance tasks and best practices for administering server machines. The directory `c:\p4\sdp\Unsupported` contains scripts for several common maintenance tasks.

The user running the maintenance scripts must have administrative access to Perforce for most activities. All of these scripts can be run from any client machine.

5.1. 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 from ftp.perforce.com and place them in `c:\p4\common\bin`
- Run `c:\p4\common\bin\upgrade.ps1 <instance>`, e.g.

```
powershell -f c:\p4\common\bin\upgrade.ps1 1
```

If upgrading a pre-2013.3 server, then this will require a checkpoint restore and the script cannot be used. Contact Perforce Support if in doubt.

5.1.1. Database Modifications

Occasionally modifications are made to the Perforce database. 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:

```
powershell -f C:\p4\common\bin\live-checkpoint.ps1 <instance>
```

E.g.

```
powershell -f C:\p4\common\bin\live-checkpoint.ps1 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 must 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.

5.1.2. Unloading and Reloading labels

Archiving labels is a best practice for large installations, with hundreds of users and Perforce checkpoints that are gigabytes in size. Smaller sites need not necessarily concern themselves with archiving labels to maintain performance, though doing so will minimize database size if labels are used extensively.

To use the `p4 unload` and `p4 reload` commands for archiving clients and labels, you must first create an unload depot using the p4 depot command. Run:

```
p4 depot unload
```

Set the type of the depot to unload and save the form.

After the depot is created, you can use the following command to archive all the clients and labels that have been accessed since the given date:

```
p4 unload -f -L -z -a -d <date>
```

For example, to unload all clients and labels that haven't been accessed since Jan. 1, 2019, you would run:

```
p4 unload -f -L -z -a -d 2019/01/01
```

Users can reload their own clients/labels using the reload command. They can run:

```
p4 reload -c <clientname>
```

or

```
p4 reload -l <labelname>
```

As a super user, you can reload and unloaded item by adding the -f flag to the reload command as follows:

```
p4 reload -f -c|l <specname>
```

In addition, you can avoid having to unload/reload labels by creating a trigger to set the autoreload option as the default on all new labels. That will cause the server to use the unload depot for storing the labels rather than storing them in db.label. This helps with performance of the server by not increasing the size of the database for label storage.

You can automate these tasks with $SDP/Maintenance/unload_clients.py and $SDP/Maintenance/unload_labels.py

5.1.3. Workspace management

The simplest option is to use create a template client workspace (usual name is `template.client`) and then set configurable `template.client` to that name. This will mean that all new client workspaces created after that time will have the same options and view, unless otherwise explictly updated.

```
p4 client template.client     [edit and save]
```

```
p4 configure set template.client=template.client
```

Alternatively the old fashioned way is to install a trigger from the Unsupported/Samples/triggers folder.

The form-out trigger $SDP/Unsupported/Sample/triggers/SetWsOptions.py contains default workspace options, such as leaveunchanged instead of submitunchanged.

To use the trigger, first copy it to /p4/common/bin/triggers

To enable the trigger, first modify the OPTIONS variable in the script, providing the set of desired options. Then insert an entry in the trigger table like the following:

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

The form-save trigger $SDP/Server/common/p4/common/bin/triggers/PreventWsNonAscii.py enforces the policy that no workspaces may contain non-ASCII characters.

To use the trigger, first copy it to /p4/common/bin/triggers

To enable the trigger, insert an entry in the trigger table like the following:

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

5.1.4. Removing empty changelists

To delete empty pending changelists, run python remove_empty_pending_changes.py.

Chapter 6. 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.

6.1. 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. The weekly checkpoint script used as part of the normal server maintenance automates this task.

To minimize the size of back up files and maximize server performance, minimize the size of the db.have and db.label files. The scripts described in Unloading and Reloading labels, **Deleting users**, and

6.2. Managing server load

6.2.1. 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.

6.2.2. 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.

P4P does not require special hardware because it doesn't use much processing power, and 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.

6.3. P4V performance settings

At large sites with hundreds or thousands of simultaneous users, the P4V data retrieval settings can help prevent P4V requests from impacting server performance. As of the 2010.1 release, P4V settings that affect performance can be centrally managed for all users or specific groups of users, using the JavaScript API (P4JsApi).

The SDP includes a sample P4V settings file, along with the P4JsApi centralsettings file that enables it. These files are located in //Perforce/sdp/JsApi.

Follow these steps to provide P4V performance settings for your users.

1. Determine whether you want P4V settings common to all users, or different settings for different groups. If the latter, make a unique copy of //Perforce/sdp/JsApi/p4vsettings.xml for each group of users. For example, you may create //Perforce/sdp/JsApi/p4vsettings_dev.xml for developers and //Perforce/sdp/JsApi/p4vsettings_qa.xml for QA.
2. Review and set the performance limits in //Perforce/sdp/JsApi/p4vsettings.xml, or in each copy of this file. (The file contains suggested default values.) The available settings are:
    a. The ServerRefresh interval in minutes, which defines how often P4V attempts to get updated information from the server.
    b. The MaxFiles that P4V will retrieve for one fetch command.
    c. The MaxFilePreviewSize in kilobytes.
    d. The FetchCount, which affects the number of forms fetched for some operations.
3. If using common settings for all users, proceed with this step; otherwise proceed to the next step. Install the centralsettings file by adding a line to the protections table like:

```
list group All.G centralsettings //Perforce/sdp/JsApi/centralsettings.js
```

(This line assumes that you have a group called All.G that represents all users.)

4. (Skip this step if using common settings for all users.) If using different settings for different groups, create a copy of //Perforce/sdp/JsApi/centralsettings.js for each group of users. For example, you may create //Perforce/sdp/JsApi/centralsettings_dev.js for developers and //Perforce/sdp/JsApi/centralsettings_qa.js for QA. Modify the line that references p4vsettings.xml to reference the copy for the group.
5. (Skip this step if using common settings for all users.) Install each copy of centralsettings.js in the protections table. In our example with separate copies for developers and QA, we would use lines like:

```
list group Dev.G centralsettings //Perforce/sdp/JsApi/centralsettings_dev.js
list group QA.G centralsettings //Perforce/sdp/JsApi/centralsettings_qa.js
```

6. Each P4V user must follow the instructions in the P4JsApi manual to enable P4V extensions.

Of course, the P4JsApi provides many other valuable features. If you choose to use these features, you can use the same centralsettings files for your groups to enable them. Refer to the P4JsApi manual for details.

Chapter 7. Tools and Scripts

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

Scripts are located typically in the following directory unless otherwise specified:

```
c:\p4\common\bin
```

The following sections describe the scripts in detail.

7.1. Standard scripts

The scripts are implemented in **Powershell**, and usually have a simple `.bat` wrapper script. Note that for historical reasons there will often be 2 versions of the `.bat` which both call the same underlying Powershell script. For example, one uses '-' and one '_' as seperators:

```
daily-backup.bat
daily_backup.bat
```

Both are effectively identical and call the same Powershell script like this:

```
powershell -file c:\p4\common\bin\daily-backup.ps1 %1
```

In the sub-sections below we refer to the `.ps1` scripts. Please assume the `.bat` calling wrappers are present.

7.2. Core scripts

7.2.1. daily-backup.ps1

This script is configured to run seven days a week using the Windows scheduler. The script truncates the journal, replays it into the `offline_db` directory, creates a new checkpoint from the resulting database files, then recreates the `offline_db` directory from the new checkpoint.

This procedure rebalances and compresses the database files in the offline_db directory. These can be rotated into the live database directory on an occasional (e.g. monthly) basis using Section 7.3.7, “recreate-live-from-offline-db.ps1”.

Usage

```
<#
    .Synopsis
        Daily_Backup.ps1 performs journal rotation to offline database and creates
offline checkpoint

    .Description
        Admin access is required.
        Also recovers from the offline checkpoint to ensure that it is good, and that
offline
        DB files are unfragmented.

    .Parameter sdp-instance
        The specified instance to backup

    .Example
        daily_backup.ps1 Master

    .Example
        daily_backup.ps1 1
#>
```

7.2.2. p4verify.ps1

This script verifies the integrity of the depot files. This script is run by Windows scheduler, usually on a weekly basis, e.g. Saturday morning. It emails the resulting report - please check for any errors contained.

for larger repositories this can take many hours to run, and places some load on the server. Run it at the weekends when this is less of a problem.

Usage

```
<#
    .Synopsis
        p4verify.ps1 performs p4d archive verification on all appropriate depots

    .Description
        Runs "p4 verify -qz //depot/..." as appropriate to the depot type.
        For replicas it adds the "-t" flag to transfer missing revisions.
        Sends an error message with errors if BAD! or MISSING! is found in the output.

    .Parameter sdp-instance
        The specified SDP instance to verify

    .Example
        p4verify.ps1 Master

    .Example
        p4verify.ps1 1
#>
```

7.3. Other scripts and tools

7.3.1. create-filtered-edge-checkpoint.ps1

This script creates a checkpoint from the offline_db files, filtered for use with an edge server. Note restrictions below.

See also partner script Section 7.3.6, “recover-edge.ps1” which replays the checkpoint on the edge server machine.

Usage

```
<#
    .Synopsis
        Creates a filtered edge checkpoint from a master offline database.
        The resulting checkpoint can be copied to the remote edge server
        and restored using recover-edge.ps1

    .Description
        Create filtered checkpoint using the server spec (output of
        'p4 server -o', and in particular the fields RevisionDataFilter:
        and  ArchiveDataFilter: which specify filtering).

        IMPORTANT NOTE:
        Because this uses the offline database, you can not just edit an
        existing server spec to change the filter and have it picked up by
        this script.

        After changing the server spec for the live server, you must ensure that the
metadata
        changes are reflected in the offline_db - which is easy to do by
        running daily-backup.ps1 as normal (this rotates the journal and
        applies it to offline_db).

        Alternatively you can wait until the following day to run this
        script, by which time a scheduled daily backup should have run!

        Output of this script:
        A gzipped (filtered) checkpoint file, reflecting the name of the
        current instance. E.g. if current instance is Master, then result
        will be c:\p4\Master\checkpoints\p4_Master.ckp.filtered-edge.193.gz
        where 193 is the latest numbered checkpoint in that directory.

    .Parameter SDPInstance
        The specified instance to process, e.g. 1 or Master

    .Parameter EdgeServer
        The specified id of edge server (a server spec visible in output of
        'p4 servers' command).

    .Example
        create-filtered-edge-checkpoint.ps1 Master Edge-server

#>
```

7.3.2. create-offline-db-from-checkpoint.ps1

This script recreates offline db from the latest checkpoint found.

Usage

```
<#
    .Synopsis
        Create-offline-db-from-checkpoint.ps1 recreates offline_db using latest
checkpoint found.

    .Description
        This script recreates offline_db files from the latest checkpoint. If it
fails, then
        check to see if the most recent checkpoint in the c:\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 live-checkpoint.ps1 script (which locks
database while
        it runs).

    .Parameter sdp-instance
        The specified instance to process

    .Example
        Create-offline-db-from-checkpoint.ps1 Master

    .Example
        Create-offline-db-from-checkpoint.ps1 1
#>
```

7.3.3. grep.exe

Windows version of Unix `grep` command. Useful for searching inside files.

There is a Windows equivalent which is `findstr` (although not as powerful).

7.3.4. gzip.exe

Windows version of Unix `gzip` command - useful for checkpoint (de)compression.

7.3.5. live-checkpoint.ps1

This script stops the server, creates a checkpoint from the live database files, recovers from that checkpoint to rebalance and compress the files, then recovers the checkpoint in the offline_db directory to ensure that the database files are optimized.

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

Usage

```
<#
    .Synopsis
        Live_checkpoint.ps1 checkpoints the live database and creates offline
checkpoint

    .Description
        Admin access is required.
        This will lock the database for the duration which can be hours for large
        repositories!

    .Parameter sdp-instance
        The specified instance to checkpoint

    .Example
        live_checkpoint.ps1 Master

    .Example
        live_checkpoint.ps1 1
#>
```

7.3.6. recover-edge.ps1

This script recreates an edge server from create-filtered-edge-checkpoint, maintaining local data such as workspaces (in edge specific db.have table) plus the other 6+ edge tables.

Partner script to Section 7.3.1, “create-filtered-edge-checkpoint.ps1”

Usage

```
<#
    .Synopsis
        Recovers an edge server from specified Master checkpoint (which may be
filtered)
        Particularly intended for use with filtered edge servers, but handles an
        unfiltered edge server too.

    .Description
        Recover the edge server from the latest commit server checkpoint, while
keeping
        any local edge server specific state such as db.have/db.working etc.

        The normal expectation is that you will use a checkpoint created by the
        script create-filtered-edge-checkpoint.ps1

        If you want the edge server to be filtered, then you MUST use that script.
        If you just use the latest Master/Commit server checkpoint as input, then
        the edge will be assumed to be unfiltered - this script will still work.

        NOTE:
        This script will stop and restart the edge server while it is running - so
        it will be unavailable for periods while this script is running.

        This script also resets the offline_db directory for the edge server.


    .Parameter SDPInstance
        The specified instance to process

    .Parameter CkpFile
        The specified (master/commit server) checkpoint file to recover from
        (assumed to be gzipped).

    .Example
        recover-edge.ps1 Edge1 p4_1.ckp.filtered-edge.1234.gz

        Will recover for SDP instance Edge1.
#>
```

7.3.7. recreate-live-from-offline-db.ps1

This script can be scheduled to run every few months - it used to be run weekly, but that is no longer best practice since database files are not fragemented as they used to be. It will move the db.* files from offline to live root (so requires stopping the service).

Usage

```
<#
    .Synopsis
        Recreate-live-from-offline-db.ps1 updates the offline database and then swaps
it over to
        replace the live database.

    .Description
        Admin access is required.

    .Parameter sdp-instance
        The specified instance to process

    .Example
        Recreate-live-from-offline-db.ps1 Master

    .Example
        Recreate-live-from-offline-db.ps1 1
#>
```

7.3.8. replica-status.ps1

This script sends an email with the results of the latest `p4 pull -lj`. Useful for basic monitoring services.

Usage

```
<#
    .Synopsis
        replica-status.ps1 emails the latest state of the replica status using "p4
pull -lj"

    .Description
        Normally set up to run once per day.

    .Parameter sdp-instance
        The specified instance to backup

    .Example
        replica-status.ps1 Master

    .Example
        replica-status.ps1 1
#>
```

7.3.9. rotate-log-files.ps1

This script is intended to be run nightly on a replica which may not have any other scheduled tasks

running. It ensures that the log files are appropriately rotated and old logs (and journals) are deleted according the settings of KEEP_CKPS in sdp_config.ini

Usage

```
<#
    .Synopsis
        rotate-log-files.ps1 rotates key log files for service.

    .Description
        Rotates all log files found and zips them.
        Useful for replicas which may not otherwise have scheduled tasks set.
        For use in Windows Task Scheduler.

    .Parameter sdp-instance
        The specified SDP instance to verify

    .Example
        rotate-log-files.ps1 Master

    .Example
        rotate-log-files.ps1 1
#>
```

7.3.10. SDP-functions.ps1

This script is the main repository of all shared functions used by other scripts.

They each source the file and then call individual functions as required. This is not intended by be called directly by the user - just sourced by other scripts.

E.g.

```
# Source the SDP Functions shared between scripts
$SDPFunctionsPath = Split-Path -parent $MyInvocation.MyCommand.Path | Join-Path
-childpath "SDP-Functions.ps1"
. $SDPFunctionsPath
```

It understands how to parse config files, start/stop instances, rotate log files etc.

7.3.11. send-test-email.ps1

This script is useful for debugging the setup of the sending of emails by the various scripts.

It sends a test email using the values found in `c:\p4\config\sdp_config.ini`

If successful then it shows other script emails will work correctly.

See Section A.2, “Emails not being sent” in Appendix A if having problems.

7.3.12. svcinst.exe

This is used for the Windows service to:

- start
- stop
- create
- remove

It is vital that you should use this utility rather than `net stop p4_1` or other `net` commands. This utility ensures that the service is shut down cleanly, and it will not timeout.

Usage

```
C:\p4\common\bin> svcinst.exe

Perforce Service Manager Utility:

SVCINST action [-d] [-n name] [-e exe] [-a]

  -d    enables debug messages, (use as first flag)

  actions - info, create, start, stop, remove
  info
    -n name   Specify the name of the service.
  create
    -n name   Specify the name of the service.
    -e exe    Specify the executable for the service, required.
    -a        The service is to be autostart on boot, optional.
  start
    -n name   Specify the name of the service.
  stop
    -n name   Specify the name of the service.
  remove
    -n name   Specify the name of the service.
```

So example usage would be:

```
svcinst stop -n p4_1
```

7.3.13. sync-replica.ps1

This script copies checkpoint files from master to the current replica.

Usage

```
<#
    .Synopsis
        sync-replica.ps1 copies checkpoint files from master to replica to ensure
        that they are local in case of replica reseeding being required.

    .Description
        Admin access is required.
        This script rebuilds the offline_db from the copied most recent checkpoint.

    .Parameter sdp-instance
        The specified instance to process

    .Example
        sync-replica.ps1 Master

    .Example
        sync-replica.ps1 1
#>
```

7.3.14. upgrade.ps1

This script upgrades the `p4d.exe` and related files (including Windows service `p4s.exe`), performing the appropriate `-xu` to upgraded the database.

Before running this script you should ensure the new versions of p4.exe and p4d.exe have been downloaded into `c:\p4\common\bin`

Usage

```
<#
    .Synopsis
        upgrade.ps1 performs upgrades the specified Perforce SDP instance to a new
version of p4d

    .Description
        Rotates the journal, stops the live service, updates the executable,
        updates the offline and root databases, and restarts the service.

        REQUIRED: prior to running, download new p4d.exe to c:\p4\common\bin

        This script is aware of p4d versions up to 20.1 (so recognised 19.1+ new
db.storage)

    .Parameter sdp-instance
        The specified SDP instance to upgrade

    .Example
        upgrade.ps1 Master

    .Example
        upgrade.ps1 1
#>
```

Appendix A: Frequently Asked Questions

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

A.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. This error can be fixed by running the create-offline-db-from-checkpoint (or if that doesn't work then live-checkpoint script - which blocks live server), as described in Server upgrades.

A.2. Emails not being sent

The Powershell function `send-email` in Section 7.3.10, "SDP-functions.ps1" is the one used.

Problems have been observed with some SMTP providers, for example on port 465 which is implicit SSL. There are 2 possibilities:

- **explicit SSL** - this means that the client first connects to the server using an unsecure channel, requests that conversations be moved to a secure channel, and then both server and client switch to a secure connection and the rest of the communication is encrypted. Though this sounds somewhat lengthy, it's the standard procedure for setting up an SSL connection (see RFC 2228). Gmail handles explicit SSL without any difficulties, as do many other mail servers; Gmail's explicit SSL server runs on port 587.
- **implicit SSL** - In contrast, implicit SSL drops the SSL negotiation and jumps right into the SSL connection to begin with. Often, this is done through a connection to a specific port that only accepts secure connections. There is no official standard for this mode of communication, though it's widely implemented; Gmail also handles implicit SSL, this time on port 465.

A.2.1. Explicit SSL

If the server is **explicit SSL** then the script will just work, e.g. the relevant entries in Section 3.1.5, "sdp_config.ini":

```
mailhost=smtp.gmail.com
mailhostport=587
```

A.2.2. Implicit SSL

The easiest solution for **implicit SSL** is to run a local copy of stunnel which configures a local port and knows how to talk to the remote server (for example Rackspace, or Gmail).

The relevant section in the example Windows config file:

```
[gmail-smtp]
client = yes
accept = 127.0.0.1:25
connect = smtp.gmail.com:465
verifyChain = yes
CAfile = ca-certs.pem
checkHost = smtp.gmail.com
OCSPaia = yes
```

Using the above we can set the relevant entries in Section 3.1.5, “sdp_config.ini”:

```
mailhost=localhost
mailhostport=25
```

Since `stunnel` wil forward local port 25 to remote port 465.

Appendix B: SDP Package Contents and Planning

B.1. Memory and CPU

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

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 with a lower number of cores provide better performance for Perforce. Quick bursts of computational speed are more important to Perforce's performance than the number of processors, but have a minimum of two processors so that the offline checkpoint and back up processes do not interfere with your Perforce server.

B.1.1. 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.