Advanced Perforce Helix Administration

PERFORCE

Introduction

- Introductions
- Class Schedule
- GUI vs. CLI
- P4Admin Demonstrations
- About the Exercises

Course Contents

- Replication – Introduction
- Fully Distributed – Edge Servers
- Advanced Maintenance
- Offline Checkpoints
- Broker
- Security
- Advanced Tools
- Scripting

PERFORCE

Notation used herein

- **p4 command and flags or *variables:***

  p4 -p *port* command -f flags

- Items of note in output
- Examples of **commands** in text
- Sample output:

```
$ p4 ping -c 1000 -s 5120000
2.24s for 1000 messages of 5120000 characters
```

Advanced
Perforce Helix Administration

Replication - Introduction

PERFORCE

What is Replication?

- A separate Perforce Helix Server (p4d) instance which is continuously polling the master server for updates
  - Duplicating server data, typically in real time
- Has its own metadata (db.*)
  - can be filtered
  - can be fully distributed (Commit/Edge later)
- Usually has its own set of Versioned Files
  - can be filtered
  - can be shared

PERFORCE

Why Replication?

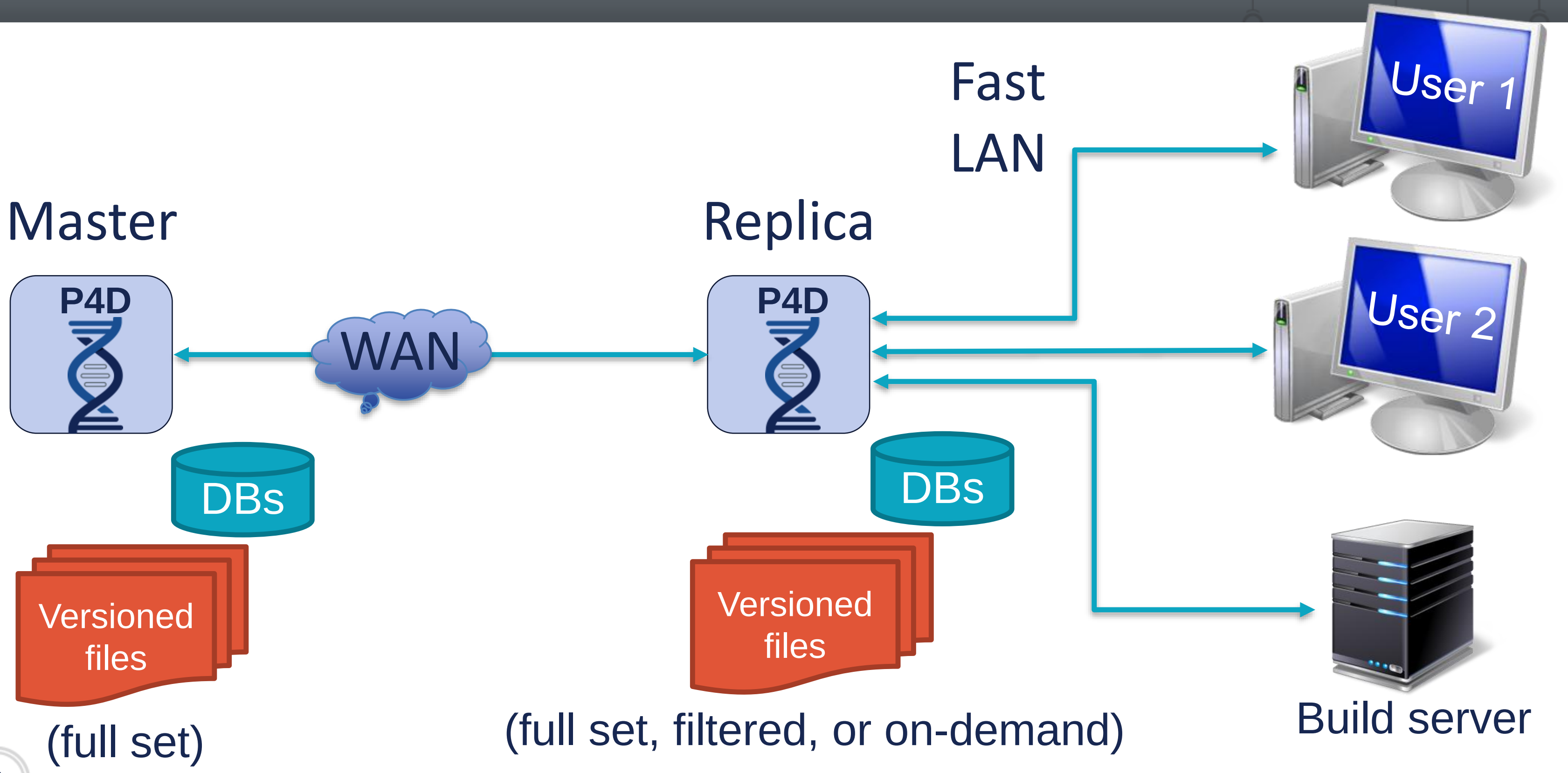
- Disaster Recovery
  - Possibly read-only
- Offloading intense server traffic
  - Reports
  - Builds
- Forwarding Replica (aka Smart Proxy)
- Edge / Commit server architecture (distributed working)

Replication - Implementation

- Server-to-Server replication
  - Asynchronous based on journal file
  - Supports both Metadata-only and Full Replication
  - No need for external scripts, complete solution
- Replicas must initially be seeded with a checkpoint (metadata)
  - Versioned files are required for full replication
    - Can be copied before setup using OS commands (e.g. rsync/robocopy)
    - Can be copied after setup using Helix Enterprise replication (p4 verify –qt)

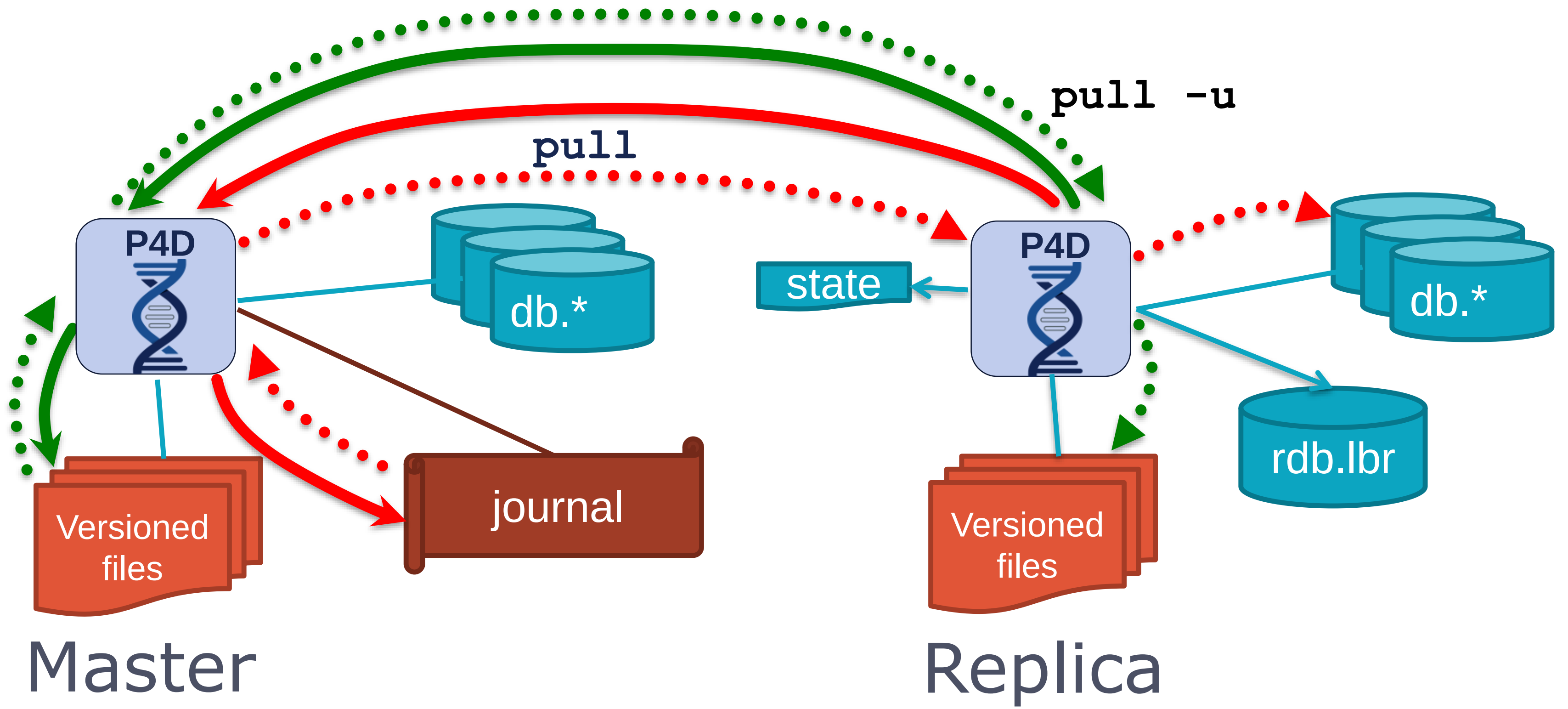
PERFORCE

Replication Architecture – General



PERFORCE

Replication Architecture – Detailed



PERFORCE

Overview of Replication Configuration

- In master repository:
  - Define serverid for master
  - Create server spec for replica (defines the server name)
  - Create replica service user in master repo
    - Assign to a group (for long timeout) and give super access
    - Create a password for user
  - Create other configurables for replica
- Create replica environment (directory structure etc)
  - Define serverid for replica
- Checkpoint master, transfer to replica, and replay
- Login to master as replica service user to create ticket
- Start replica and monitor

PERFORCE

p4 pull

- Typically runs as a background task inside the replica server

Command	Effect
p4 pull	Retrieve missing journal entries, then terminate (CLI)
p4 pull -i <N>	Continuously pull every <N> seconds (server configurable)
p4 pull -u	Retrieve missing file revisions, then terminate (CLI)
p4 pull -u -i <N>	Continuously pull file revisions (server configurable)
p4 pull -l	List missing file revisions or errors (CLI)
p4 pull -l [-j | -s]	Replica reporting (CLI)

- `p4 pull -lj` Shows metadata replication status
- `p4 pull -ls` Shows content transfer status

How does ‘p4 pull’ keep track?

- **state** file
  - Text file normally located in the replica P4ROOT directory
  - Value/example:
    - journal / offset
    - `104/28398`
  - Allows replication to be interrupted
  - Master server can rotate journal file
    - Configure journalPrefix if master uses journal prefix for checkpoints
- **rdb.lbr** database
  - Binary file located in the replica P4ROOT directory
  - Contains information on missing archive revisions

Configuration

- ‘p4 pull’ is designed to be a background process
  - Started from the replica server
  - One process for retrieving metadata (>1 not allowed)
  - Several processes to retrieve archive data (typically 2-4)

PERFORCE

Journal rotation and Prefix

- **Master**
  - `p4 admin checkpoint/journal [-Z] [prefix]`
  - Do not use –z, use –Z (uppercase)
    - Compresses checkpoint but not rotated journal file
  - If you use a prefix, must use the same prefix for ‘p4 pull’
    - **Recommended:** Use ‘journalPrefix’ configurable instead (next slide)
- **Replica**
  - `p4 pull [-J prefix] [-i n]`
  - Journal will be rotated in sync with the master

journalPrefix

- Specify journalPrefix configurable for the master to…
  - Simplify checkpoint and journal rotation
  - Avoid having to specify ‘p4 pull –J prefix’ in the replica(s)

- Specify journalPrefix configurable for the replica to…
  - Automatically rotate journal to correct location when master rotates
  - Help to prevent replica running out of disk space
  - Without journalPrefix, replica will rotate journal in P4ROOT

```
p4 configure set repl_1#journalPrefix=/replica/checkpoints/repl_1
```

```
/replica/checkpoints/repl_1.ckp.100.gz
/replica/checkpoints/repl_1.jnl.101
```

Prepare in the Master

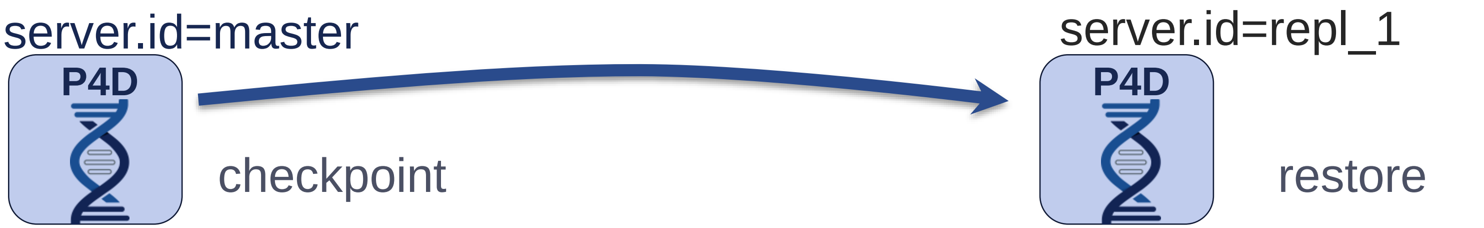
```
> p4 configure set monitor=1
For server 'any', configuration variable 'monitor' set to '1'

> p4 configure set master#net.tcpsize=512k
For server ‘master', configuration variable 'net.tcpsize' set to '512k’

> p4 configure set repl_1#P4TARGET=master:1666
For server ‘repl_1', configuration variable 'P4TARGET' set to 'master:1666'
```

PERFORCE

Prepare in the Master



p4 configure show allservers

```
any: monitor=1
master: net.tcpsize=512k
master: lbr.bufsize=64k
repl_1: P4TARGET=master:1666
repl_1: serviceUser=service
repl_1: db.replication=readonly
repl_1: lbr.replication=readonly
repl_1: startup.1=pull -i 0
repl_1: startup.2=pull -i 1 -u
```

p4 configure show allservers

```
any: monitor=1
master: net.tcpsize=512k
master: lbr.bufsize=64k
repl_1: P4TARGET=master:1666
repl_1: serviceUser=service
repl_1: db.replication=readonly
repl_1: lbr.replication=readonly
repl_1: startup.1=pull -i 0
repl_1: startup.2=pull -i 1 -u
```

server.id determines which configuration is active

Server environment settings

- Command line flags
  - `p4 configure set`
  - `p4d -cset`
- Environment variables
- (On Windows) registry variables

PERFORCE

Configuration parameters

Parameter	Sample Values
P4TARGET	svrmaster.example.com:1666
db.replication	readonly
lbr.replication	readonly
rpl.forward.all	1
serviceUser	repl_1_svc
startup.1	pull –i 0
startup.2	pull –u –i 1
startup.3	pull –u –i 1

PERFORCE

Active Replication Monitoring (CLI)

- `p4 pull -l [-j|-s]`
  - Reports pending transfers
- `p4 verify [-t]`
  - Option -t schedules content transfer of missing/damaged revision
- `p4 journaldbchecksums`
  - Run on master, check log on replica

Naming Servers

- All Helix Enterprise servers should
  - Be named
  - Have server specifications
  - **`p4 server`** ***`servername`***
- Server names…
  - Are used in replication and failover and other scenarios
  - Define server capabilities
  - Determine which configurables apply to a server
  - Enforce security
    - Require special service accounts for access by remote servers

Naming Servers

- **`p4 serverid`** ***`[serverID]`***
- **`p4d -xD`** ***`[serverID]`***
  - Sets/retrieves server.id file in server's root directory
- Tells server which configurables apply to it
- **`P4NAME`** – Environment variable
  - Required on Windows prior to 2015.2 release because server.id file is ignored.
  - Overrides server.id file
  - **Not** suggested for use on platforms other than Windows

Server Specifications

- `p4 server servername`
  - Creates or updates information about a server
  - Specifies information about a server – the most important is the type (services it provides)

Type	Definition
`standard`	Standard Helix Server
`replica`	Helix replica server
`broker`	Helix Broker
`proxy`	Helix Proxy
`forwarding-replica`	Helix smart proxy
`build-server`	Helix Build Server
`commit-server`	Helix Commit Server
`edge-server`	Helix Edge Server
`P4AUTH`	Helix Authentication Server
`P4CHANGE`	Helix Change Server

Configurables and Named Servers

- `p4 configure show`
  - Shows running configuration of queried server
- `p4 configure show allservers`
  - Shows stored configurables for all servers
- Use 'p4 configure set/show' for named servers
- `p4 configure show` ***SERVERID***
- `p4 configure set` ***SERVERID#variable=value***
- `p4 configure set repl_1#P4TARGET=192.168.1.1:1666`
- `p4 configure show repl_1`

PERFORCE

Configurables and Named Servers

```
> p4 configure show
P4ROOT=. (-r)
P4PORT=9876 (-p)
P4JOURNAL=journal (default)
auth.default.method=perforce (default)
```

```
➢ p4 configure show repl_1
repl_1: P4TARGET = 192.168.1.1:1666
repl_1: P4TICKETS = /path/to/replica1/.p4tickets
repl_1: db.replication = readonly
repl_1: lbr.replication = readonly
repl_1: startup.1 = pull -i 1
repl_1: startup.2 = pull -u -i 1
```

Service user

- Replication requires user of type **`service`**.
- This service user requires 'super' access.
- Add user to a group (e.g. service.g) group with unlimited timeout.

- On replica login as service user before starting replication
  - Define P4TICKETS location for the replica on command line
  - P4TICKETS should also be defined (same value) as a configurable for each server

```
set P4TICKETS=c:\p4\p4tickets.txt
p4 –u p4admin login repl_1_svc
```

PERFORCE

Replication set up – check master id

- Check master has a serverid
  - `p4 serverid`

```
Server ID: master
```

- If necessary, set it:
  - `p4 serverid master`
- Alternative:
  - `p4d -r . -xD`

Replication set up - master

- Set up the replica environment on the **master** server in metadata
- Create a server specification:
  - `p4 server repl_1`
  - Add `Services: forwarding-replica` to the spec and save it
- Create a replica service user:
  - `p4 user -f repl_1_svc`
  - Add `Type: service` to the spec and save it
  - `p4 passwd repl_1_svc`

Replication set up - master

- Add replica user to a group of service users (with no ticket timeout)
  - `p4 group service_users`
  - Add **`repl_1_svc`** to the spec in Users:
  - change Timeout: to unlimited
  - and save it
- Ensure group has super access:
  - `p4 protect`
  - Make sure there is a line with **`super group service_users`** present

Replication set up - master

- Set variables for the replica in the master:

```
p4 configure set server=3

p4 configure set repl_1#P4TARGET=192.168.1.1:1666

p4 configure set repl_1#P4TICKETS=/path/to/.p4tickets

p4 configure set "repl_1#startup.1=pull -i 1"

p4 configure set "repl_1#startup.2=pull -u -i 1"

p4 configure set repl_1#db.replication=readonly

p4 configure set repl_1#lbr.replication=readonly

p4 configure set repl_1#serviceUser=repl_1_svc

p4 configure set repl_1#server.depot.root=/path/to/replica/depots
```

Replication set up - master

- Verify settings on master:

```
> p4 configure show repl_1
repl_1: P4TARGET = 192.168.1.1:1666
repl_1: P4TICKETS = /path/to/replica/.p4tickets
repl_1: db.replication = readonly
repl_1: lbr.replication = readonly
repl_1: startup.1 = pull -i 1
repl_1: startup.2 = pull -u -i 1
repl_1: serviceUser = replica_svc_user
repl_1: server.depot.root = /path/to/replica/depots
```

- All okay? Take a checkpoint of the master (or rotate journal):

```
p4 admin checkpoint -Z
```

Replication set up - replica

- Setup environment on replica host (P4ROOT dir, P4LOGS, P4JOURNAL, binaries etc)
- Copy the checkpoint to the replica and restore
  - If you just rotated the journal on the master, then copy previous checkpoint and all rotated journals since then to replica and restore (this is a good option if a checkpoint takes many hours to run)
- Create the server.id file on the replica (in $P4ROOT dir):

```
p4d -r . -xD repl_1
Perforce server info:
   Server ID: repl_1
```

Replication set up - replica

- Log into the master from replica machine (with same value in P4TICKETS environment variable as is in relevant configurable):

```
export P4TICKETS=/path/to/replica/.p4tickets

p4 -p master-host:1666 -u repl_1_svc login
```

- Start the replica

Replication set up - replica

- Replication is working:

```
> p4 -p replica-host:1666 pull -lj
Current replica journal state is:      Journal 2,      Sequence 683.
Current master journal state is:       Journal 2,      Sequence 683.
The statefile was last modified at:    2014/10/30 14:27:56.
The replica server time is currently:  2014/10/30 14:28:38 -0700 PDT

> p4 -p master-host:1666 journaldbchecksums
Perforce server info:
      Table db.config checksums match. 2I li014/10/30 14:33:41 version 1: expected
Perforce server info:
      Table db.counters checksums match. 2014/10/30 14:33:41 version 1: expected
Perforce server info:
      Table db.nameval checksums empty. 2014/10/30 14:33:41 version 1: expected
```

Replication set up - troubleshooting

- Common errors:
  - Login ticket not set correctly
  - Permissions for replica user not correct
  - Typos in configuration parameters
- Look for errors in replica and master log files

```
tail -50 /path/to/master/log

tail -50 /path/to/replica/log

grep -2 "Perforce server error:" /path/to/master/log

grep -2 "Perforce server error:" /path/to/replica/log
```

Replication *live*

- Replication really is quite easy to configure
  - But you need to be precise and accurate
  - Carefully plan and review **all** configurables before taking a checkpoint of master to seed replica with
  - If you make a mistake and have to change configurables, consider rotating master journal copying only that across
- Demo: Setup and install a forwarding replica

Exercises

Lab Set E1: Replication

New commands in this chapter:

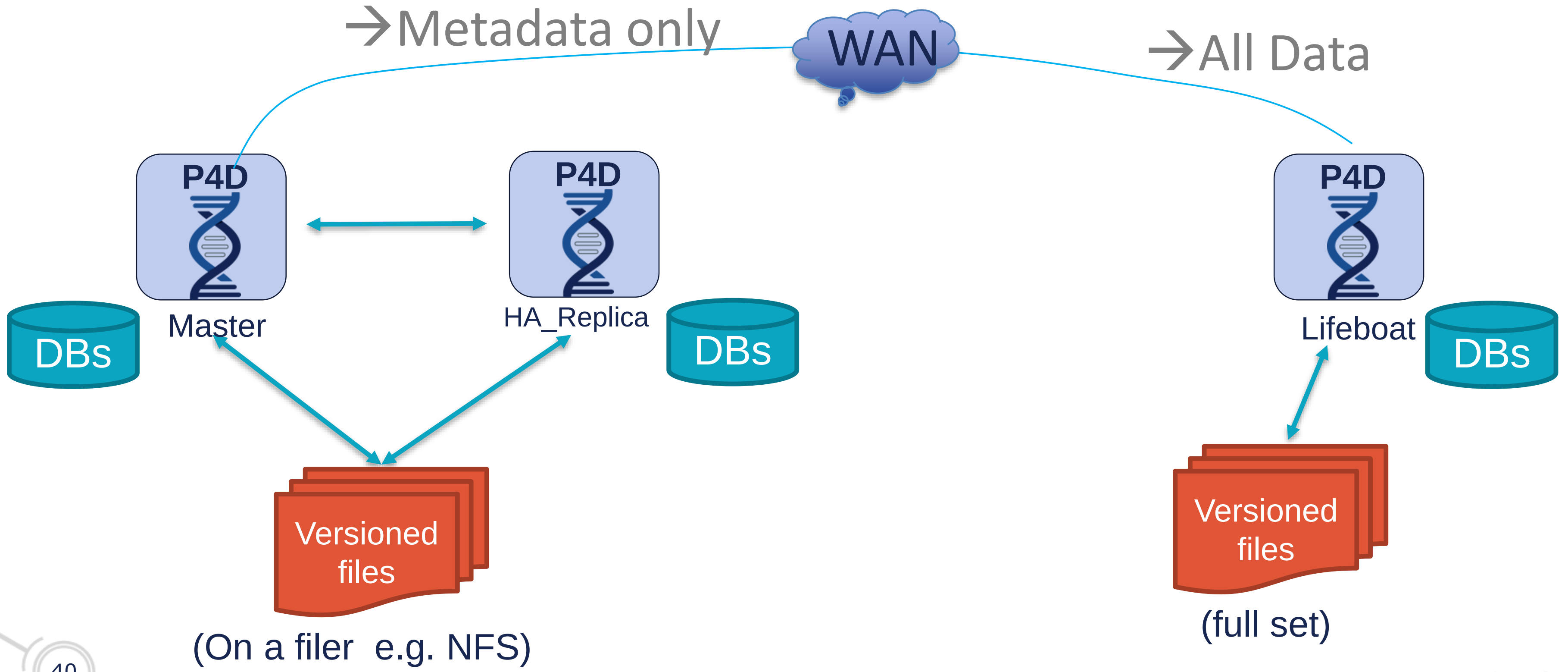
- `p4 configure set SERVERID#variable=value`
- `p4 configure show allservers`
- `p4 pull`
- `p4 pull -l [-j | -s]`
- `p4 journaldbchecksums`
- `p4 verify -t`
- `p4d -xD`
- `p4 server`

Advanced
Perforce Helix Administration

More Replication Options

PERFORCE

Replicas for HA and DR



PERFORCE

Prepare in the Master

```
p4 server Replica1
```

```
ServerID: Replica1
Name: Replica1
Type: server
Services: forwarding-replica
```

```
p4 configure set Replica1#db.replication=readonly
p4 configure set Replica1#lbr.replication=readonly
```

Equivalent value set via ‘p4 server’ specification:

```
p4 configure set Replica1#rpl.forward.all=1
```

PERFORCE

Replica filtering

- To exclude entire tables from a replica:

```
p4 pull -T db.have,db.client
```

- Detailed Filtering:

```
p4 server Replica1
ServerID: Replica1
:
ClientDataFilter:
  -//site2-ws-*
ArchiveDataFilter:
  //....c
  -//....mp4
```

```
p4 configure set
"Replica1#startup.1=pull -i
30 -P Replica1"
```

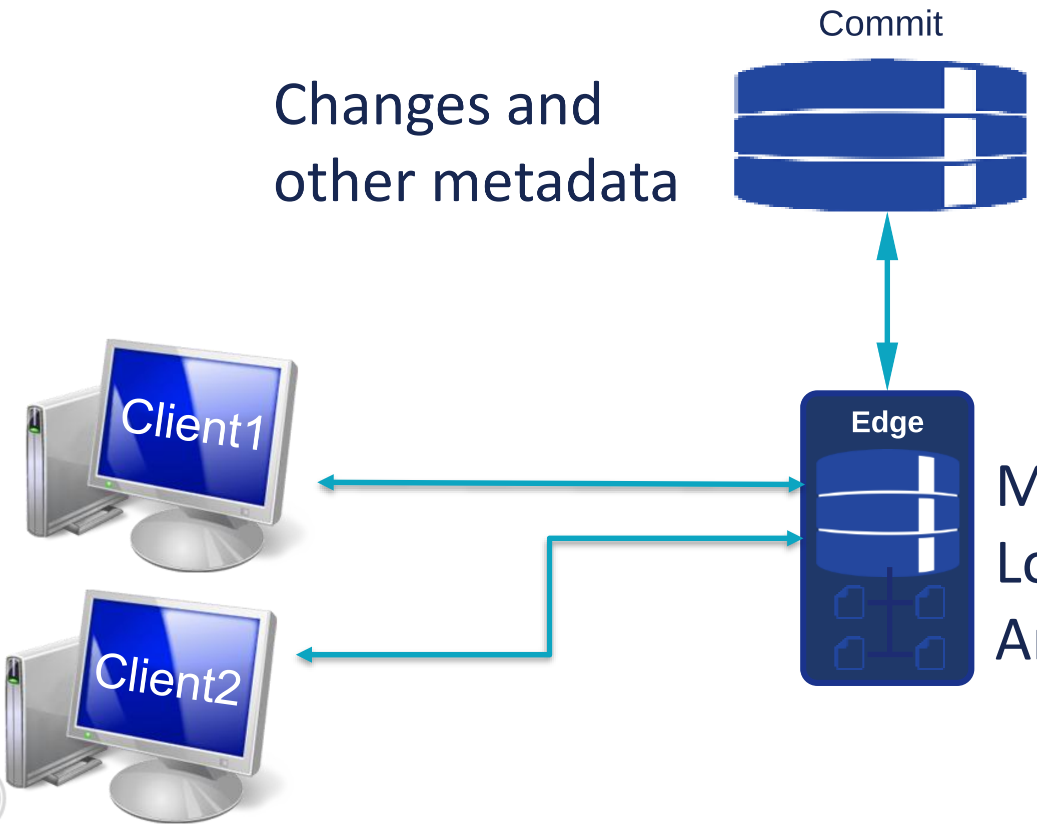
PERFORCE

Advanced
Perforce Helix Administration

Fully Distributed

PERFORCE

Edge/Commit Server Architecture

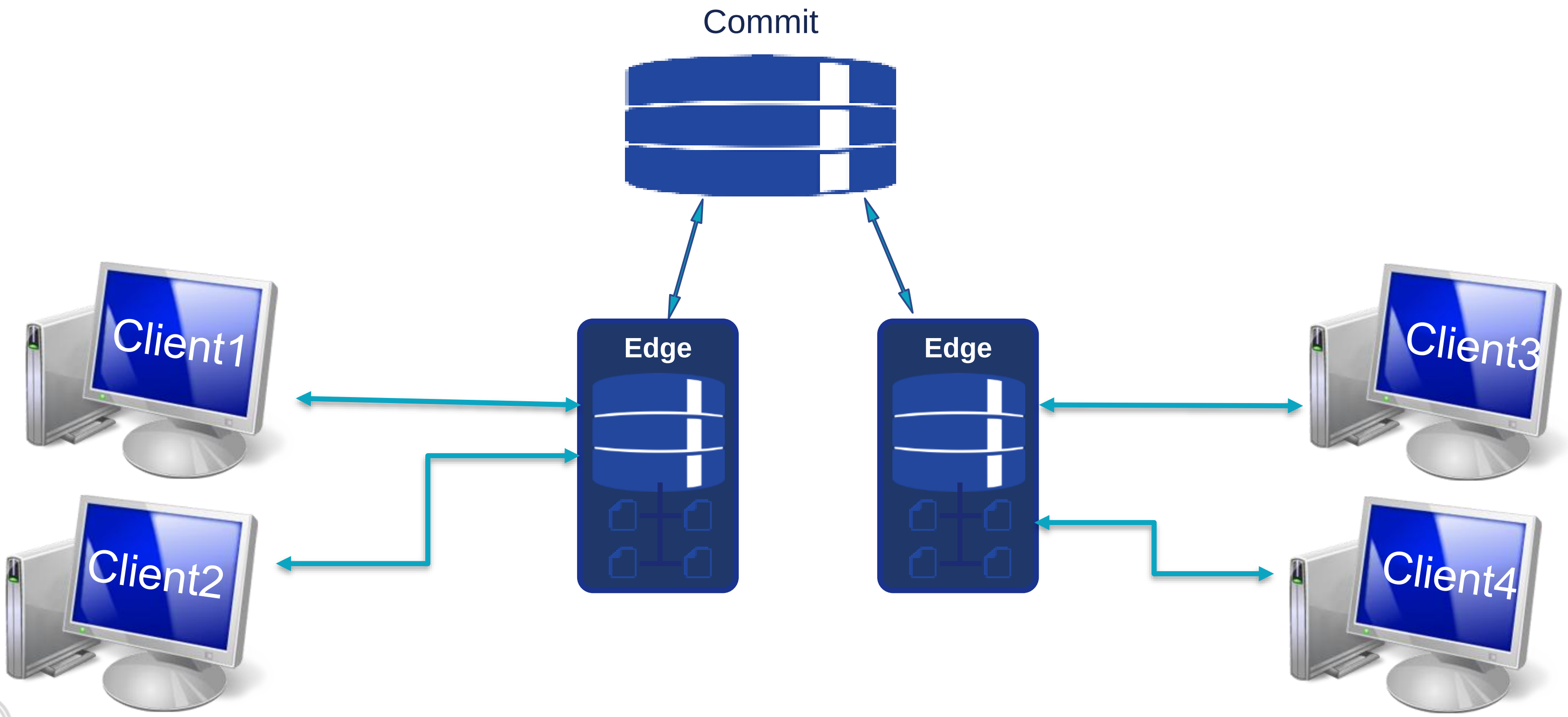


Changes and
other metadata

Metadata from Commit
Local Workspace metadata
Archive files

PERFORCE

Edge/Commit Server Architecture



PERFORCE

Prepare in the Master

```
p4 server Edge1
```

```
ServerID: Edge1
Name: Edge1
Type: server
Services: edge-server
```

```
p4 configure set Edge1#db.replication=readonly
p4 configure set Edge1#lbr.replication=readonly
```

Equivalent value set via ‘p4 server’ specification:

```
p4 configure set Edge1#rpl.forward.all=1
```

Configuring Edge workspaces

```
p4 client build-ws-9201
```

```
Client: build-ws-9201
:
ServerID: Edge1
View:
:
```

PERFORCE

Edge/Commit Considerations

- Edge servers contain locally-unique data
  - Generally require backup/recovery
- Information is distributed
  - You may need to interrogate all edge servers
- Forwarding replicas are simpler
  - Address many needs
  - large db.have is better handled with Edge servers
- Overall user performance is better with Edge servers

Build-Edge/Commit Considerations

- Edge servers for build farms don’t generally require backup
- Build data is inherently transient
- Faster to rebuild from master than to rebuild from scratch
  - Workspaces stored on master
  - ‘Have’ data stored local to Edge
  - Local ‘have’ data not valuable after build is complete

Exercises

Lab set E2: Forwarding and edge server

Advanced
Perforce Helix Administration

Advanced Maintenance

PERFORCE

Topics

- Recover a Stored Spec Revision
- Lazy Copies
- Archive/Restore

Spec Depot

- **Goal**
  - Recover specs such as clients and protection table
  - Keep history of changes to specs
  - Identify user who changed a spec

- **Implementation**
  - Separate spec depot automatically maintained by Helix Enterprise
  - Specs are stored as form files, which can be printed or synced
    - Grouped into directories by type, such as *client* or *label*

PERFORCE

Spec Depot Usage

- Spec depot stores specs like clients and protection table (not change)
- Tracing of changes by a user

```
p4 print -q //specs/label/lastbuild.p4s#1
# The form data below was edited by bruno
```

- Optional: controlling which specs are versioned

```
p4 depot specs
 SpecMap:
   //specs/...
   -//specs/client/build_ws_*
```

PERFORCE

Recovering a Stored Spec Revision

- List revisions in the spec depot

```
p4 filelog //specs/client/bruno_ws.p4s
... #4 default change edit on 2014/11/01
... #3 default change edit on 2014/10/17
... #2 default change edit on 2014/07/01
... #1 default change add on 2013/11/20
```

- Display content of revisions

```
p4 print -a //specs/client/bruno_ws.p4s
```

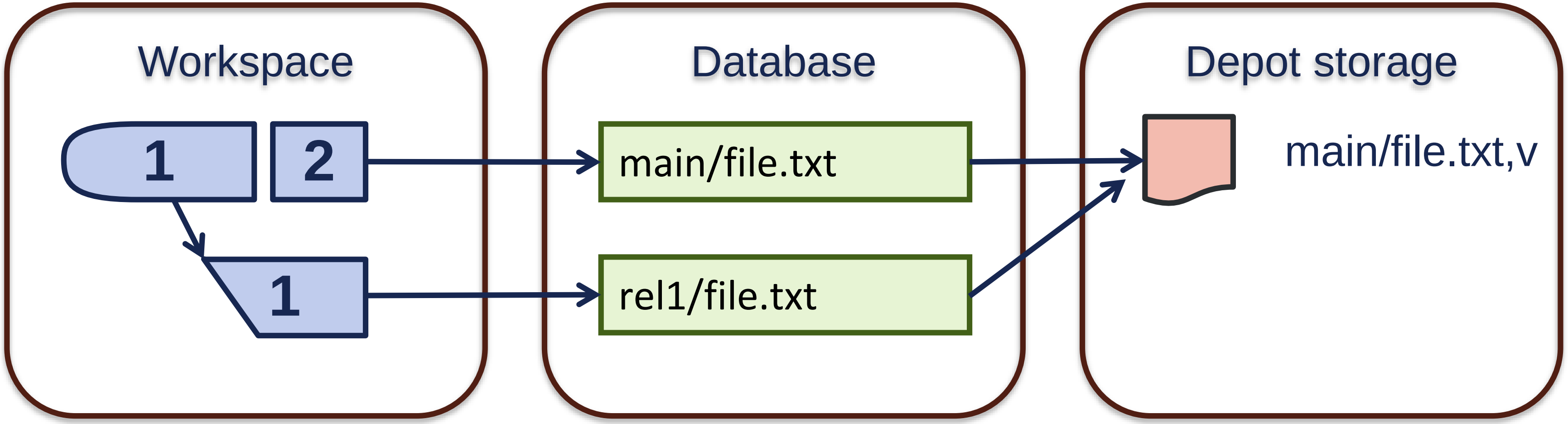
- Replace spec with earlier version

```
p4 print -q //specs/client/bruno_ws.p4s#3 | p4 client -i
```

PERFORCE

Branching and Lazy Copies

- Files branched or copied only create metadata in the db
  - Retain reference to original file location ➔ lazy copy



PERFORCE

Lazy Copies and Snap

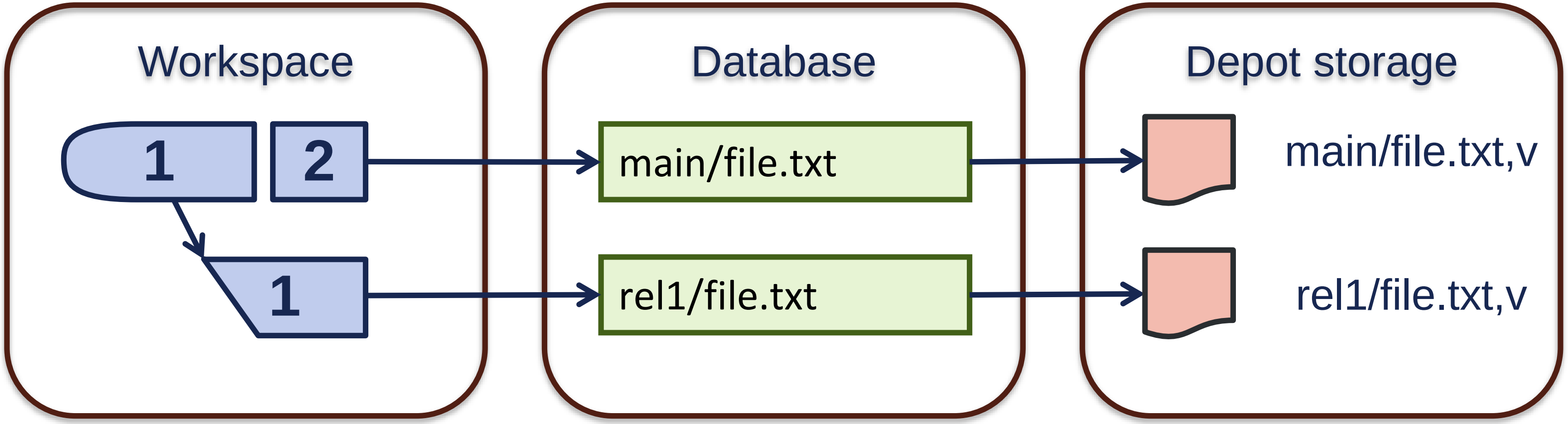
```
p4 fstat -Oc //depot/Jam/REL2.0/src/jam.c
...
... lbrFile //depot/Jam/MAIN/src/jam.c
... lbrRev 1.30
... lbrType text
... lbrIsLazy 1
```

(undocumented)

```
p4 snap //depot/Jam/REL2.0/src/jam.c
//depot/Jam/REL2.0/src/jam.c#1 - copy from //depot/Jam/MAIN/src/jam.c 1.30
```

After Snap

- Files in the depot storage are duplicated
- Useful when cleaning up depots with *obliterate*



PERFORCE

Archiving and Restoring

- Goal:
  - Free up space in active depots
  - Speed up backup and verify
  - Preserve history
  - Simple restore
- Implementation:
  - Separate archive depots (typically located on cheap storage)
  - Files can be archived and restored at individual revisions

Archiving and Restoring

- Files *not branched* can be archived
  - Requires at least one depot of type *archive*
  - Preserves history

```
p4 archive -D archives //assets/...
```

- To archive files stored in delta format, use the -t option.

```
p4 archive -D archives -t //assets/text/readme.txt#9,9
```

- Restore files as needed

```
p4 restore -D archives //assets/images/myimage.jpg#3
```

Archiving – Listing and Purging

- Files in original depot are marked as *archive*

```
p4 files //assets/...
//assets/images/myimage.gif#1 - archive change 865 (ubinary)
...
```

- List files in archive depot

```
p4 files -A //archives/...
//archives/assets/images/myimage.gif#1
...
```

- Purge unneeded archived files (cannot be undone)

```
p4 archive -D archives -p //assets/...@2012/01/01
```

Exercises

Lab Set E3: Advanced Maintenance

New commands in this chapter (samples):

- `p4 archive`
- `p4 restore`
- `p4 snap (undoc)`

Advanced Perforce Helix Administration

Offline Checkpoints

PERFORCE

Topics

- Offline Checkpoints
  - Usage
  - Upgrades
  - Switch offline_db/root

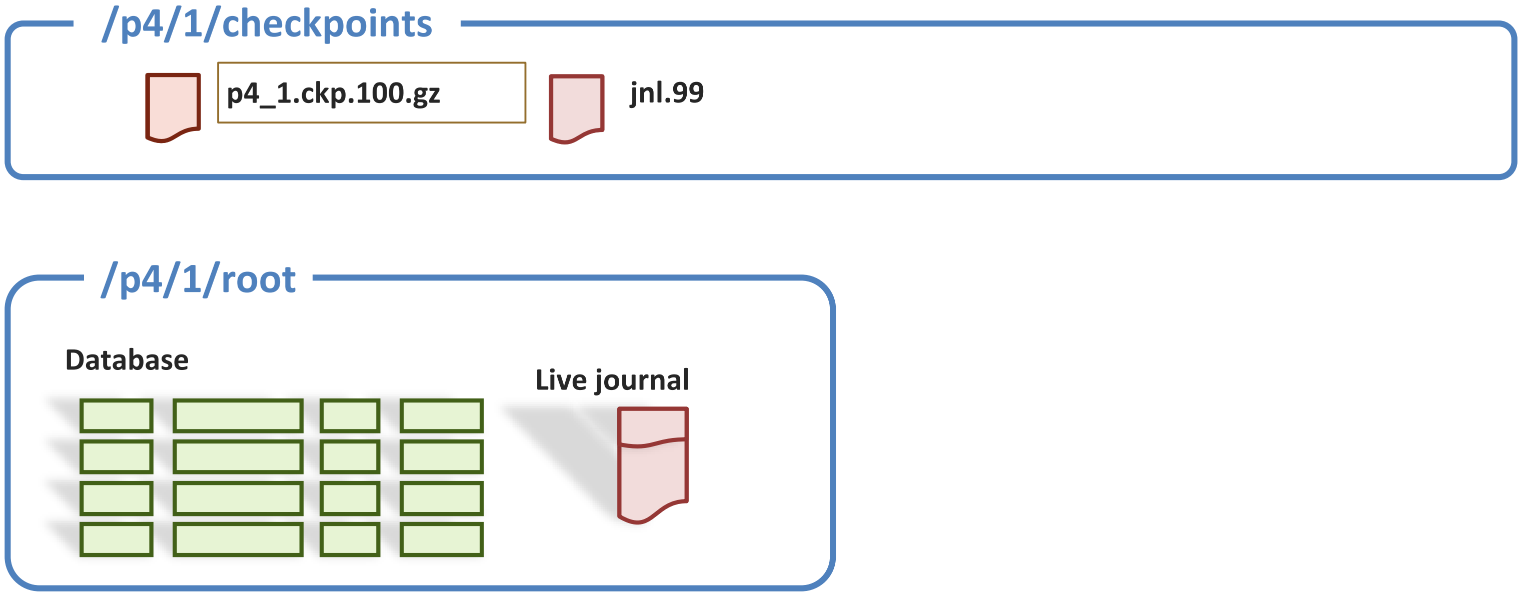
Offline Checkpoint

- **Goal**
  - Checkpoint without any downtime
  - Easy and fast recovery
  - Optional: regular database restoration

    Restored databases are smaller than original, but contain equivalent data
    (Removes empty data pages and rebalances the b-tree indexes)
- **Implementation**
  - Separate offline database created from checkpoint
  - Regular updates through rotated journal
  - Offline database dumped into checkpoint

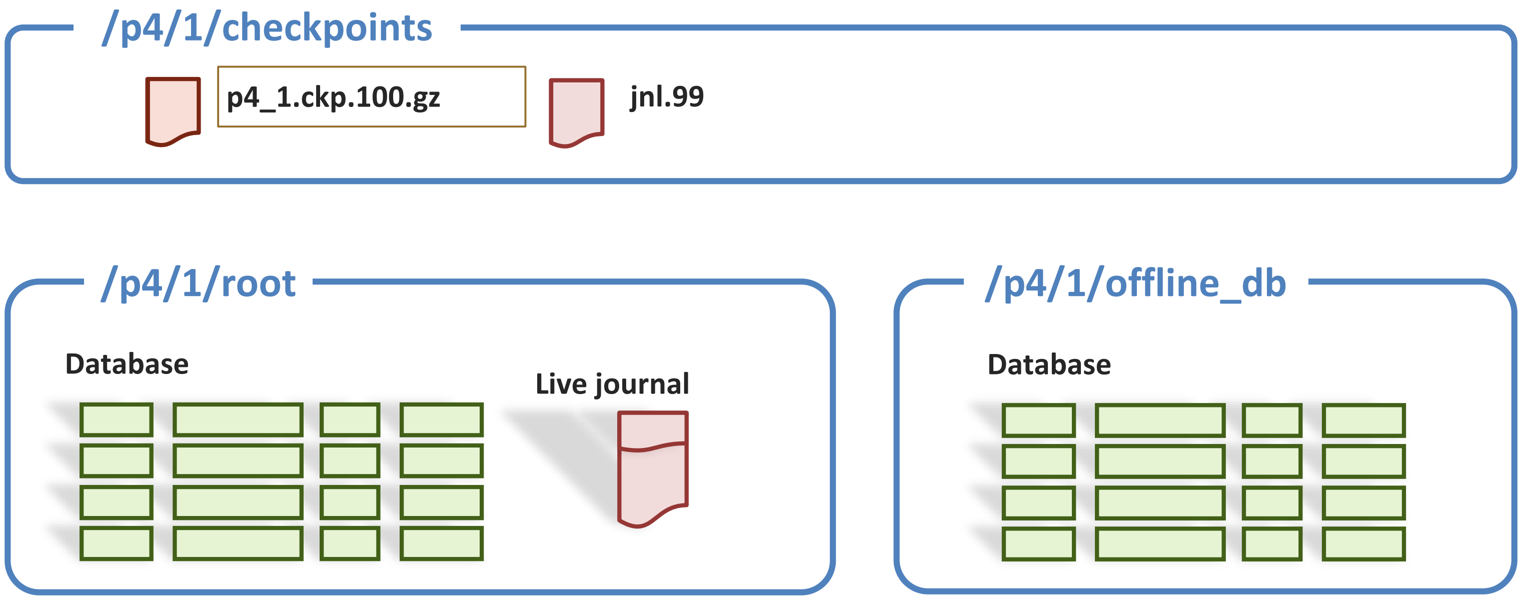
Prep Offline Checkpoint – Create Seed

```
p4d -r /p4/1/root -jc -Z /p4/1/checkpoints/p4_1
```



Prep Offline Checkpoint – Apply Seed

```
p4d -r /p4/1/offline_db -jr -z /p4/1/checkpoints/p4_1.ckp.100.gz
```



PERFORCE

Offline Checkpoint

- Nightly:
  - Truncate journal on live database

```
p4d -r /p4/1/root -J /p4/1/logs/journal -jj /p4/1/checkpoints/p4_1
```

  - Restore journal to offline directory

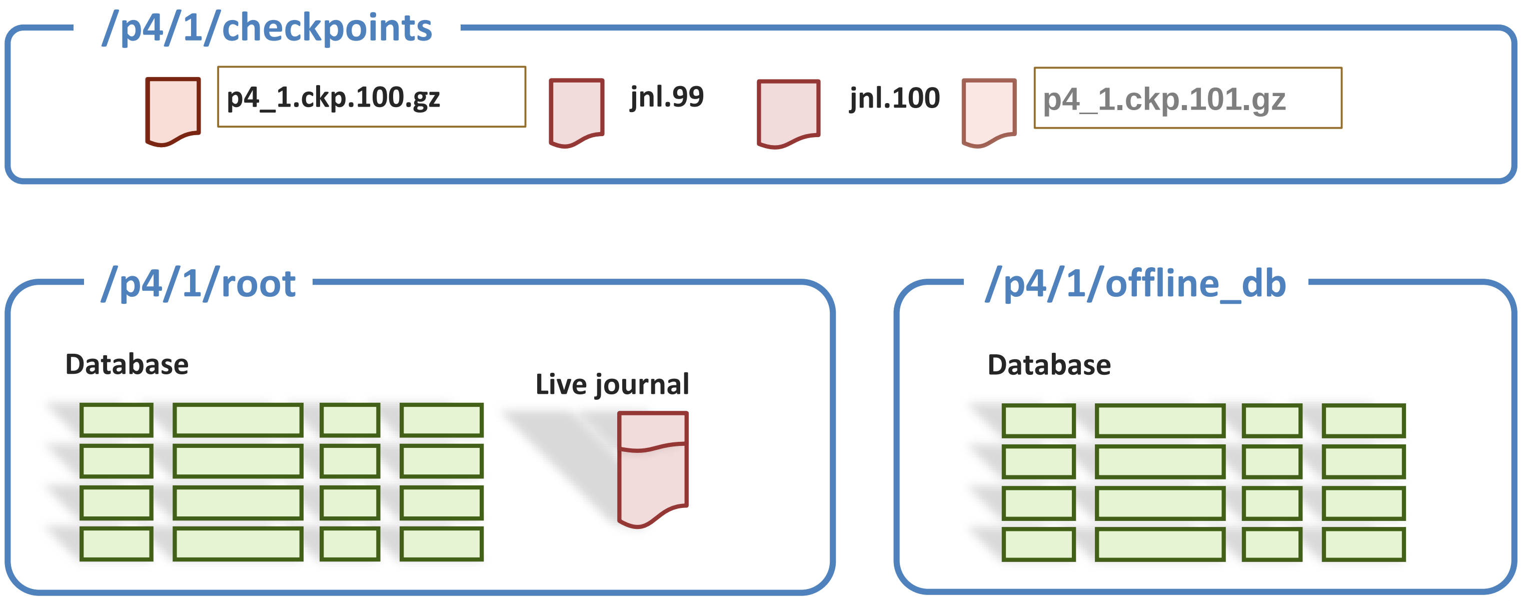
```
p4d -r /p4/1/offline_db -jr /p4/1/checkpoints/p4_1.jnl.100
```

  - Dump the offline database to make a new checkpoint

```
p4d -r /p4/1/offline_db -jd -z /p4/1/checkpoints/p4_1.ckp.101.gz
```

Offline Checkpoint

Truncate journal `-jj` ❯ Restore journal `-jr` ❯ Dump checkpoint `-jd`



PERFORCE

Recreate Offline Database

- Recreate the offline database from the new checkpoint

```
rm -f /p4/1/offline_db/db.*
p4d -r /p4/1/offline_db -jr -z /p4/1/checkpoints/p4_1.ckp.101.gz
```

Switch Offline Database/Root

- Stop the production server
- Rotate the journal
- Replay the journal to the offline_db
- Move **`/p4/1/root/db.* /p4/1/root/save/`**
- Move **`/p4/1/offline_db/db.* /p4/1/root/`**
- Restart the master server
- Delete the files in **`/p4/1/root/save/`**
- Recover the most recent checkpoint into **`/p4/1/offline_db`**
- Recover the journals following the checkpoint into **`/p4/1/offline_db`**
- Dump a checkpoint from **`/p4/1/offline_db`**
- Recreate the offline database from the new checkpoint

PERFORCE

Exercises

- Lab Set E4: Offline Checkpoints

Advanced
Perforce Helix Administration

Broker

PERFORCE

P4Broker

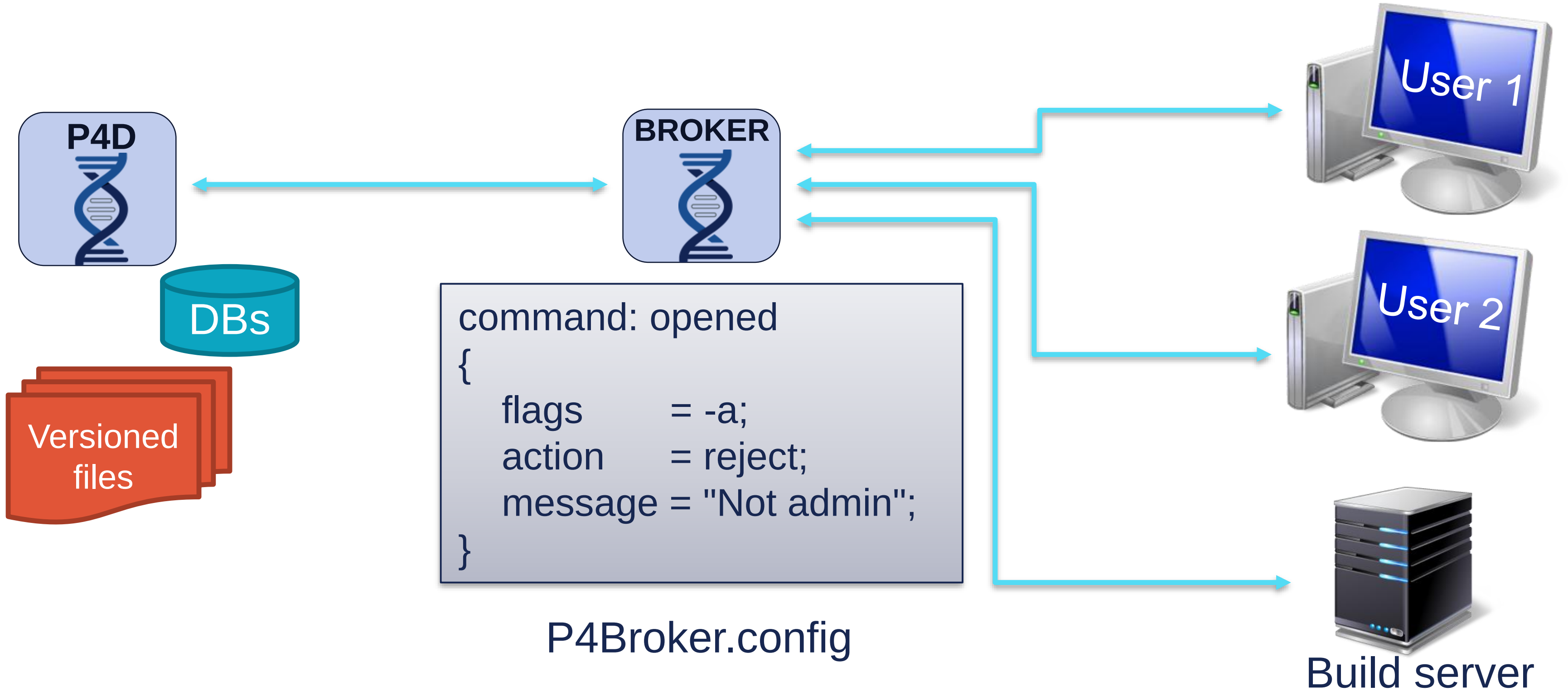
- Intercepts all incoming Helix Enterprise commands
- Command handling support:
  - Redirection
  - Blocking
  - Rewriting (undocumented)
- Great for notifying users when the server is down for maint.
- Sometimes used as part of HA/DR strategies to avoid DNS change delay.

P4Broker Use Cases

- Policy Customizations
  - different capabilities than triggers
- Traffic Redirection for Load Distribution
  - not “load balancing”
- Traffic Redirection for execution of automated failover operations
  - advanced/custom usage

PERFORCE

Helix Broker



```
command: opened
{
    flags      = -a;
    action     = reject;
    message = "Not admin";
}
```

P4Broker.config

PERFORCE

Redirection

- Selective – The default setting
  - Redirection allowed, but after the first command in a session hits the default server, all others in the same session use the default server and are not redirected.

- Pedantic – All redirected commands are redirected
  - Can cause the GUI to not update the icons correctly.

Filter Scripts

- When the action for a command handler is “filter”:
  - Broker executes the program or script
  - Performs the action returned by the program.
- Broker invokes filter program
- Passes in all the information about the command via stdin.
  - Filter program must read data from stdin before additional processing.
- The filter program responds on stdout with one of these:
  - action: PASS/REJECT/REDIRECT/RESPOND
  - message: Some message for the user

Mechanics: Helix Broker Setup

- Define an operating server.
- Generate a preliminary broker configuration file.
- Adjust the broker configuration to your needs.
- Set broker config file location.
- Initiate as a Windows service or Unix/Linux daemon.
- Documentation:
  - Latest Release Helix Broker Notes
  - Multi-Site Deployment Manual

Exercises

- Lab Set E5: P4Broker

Advanced
Perforce Helix Administration

Security

PERFORCE

Setting Server Security Level

- Security settings determine how Helix Server enforces passwords

- Display security counter value

```
p4 configure show security

security=3 (configure)
```

- Set security counter

```
p4 configure set security=3
```

0	**No password required, any password allowed (default)**
1	Strong password is required, can be stored in Windows registry
2	Strong password is required, cannot be stored in registry
3	p4 login tickets only, no password stored anywhere
4	Level 3 + Edge, replica, proxy & brokers must connect using a service user

Server Security

- Server security levels (0-4)
  - p4 configure set security=4
- Turn off auto user creation; require authorization for user list
  - p4 configure set dm.user.noautocreate=2
  - p4 configure set run.users.authorize=1
- Set changelists to restricted by default
  - p4 configure set defaultChangeType=restricted

Connection Protocols

- TCP
  - Default protocol

```
P4PORT=tcp:p4server:1666
```

- RSH
  - Starts up the server for each request
  - Useful for testing and inetd support

```
P4PORT=rsh:/usr/local/bin/p4d -r $P4ROOT -L $P4LOG -i
```

- SSL
  - SSL encrypted connection when using “ssl:” prefix

```
P4PORT=ssl:p4server:1667
```

RSH connection

- Starts up a server on client request
- No TCP/IP connection to server
  - Uses stdout/stdin bound to client (with -i option)

- Usage examples:
  - Sidetrack server (specify different log file)
  - Test environments (P4Python, P4Ruby, P4Perl)

SSL Encryption

- Helix Server, Helix Proxy, Helix Broker
- Consider implications with 3rd party integrations
- If enabled, all clients require SSL connection.
  - Run two P4Ds to offer SSL and non-SSL (one with "ssl:", one without)
- Client needs fingerprint in its P4TRUST file

```
p4 trust
```

p4 trust

- Client-side command for handling fingerprints
- Uses P4TRUST environment variable (default $Home/.p4trust)

```
p4 trust -h
```

`p4 trust -y`	Accept the fingerprint
`p4 trust -n`	Reject the fingerprint
`p4 trust -f`	Force overwriting of the fingerprint
`p4 trust -l`	List accepted fingerprints
`p4 trust -d`	Delete a fingerprint

SSL Setup

- P4SSLDIR -> directory with key and certificate

```
cd $P4ROOT
mkdir ssl          # optionally create config.txt
chmod 700 ssl      # drwx------
export P4SSLDIR=ssl
p4d -r . -Gc       # key and certificate
p4d -r . -p ssl:1667
```

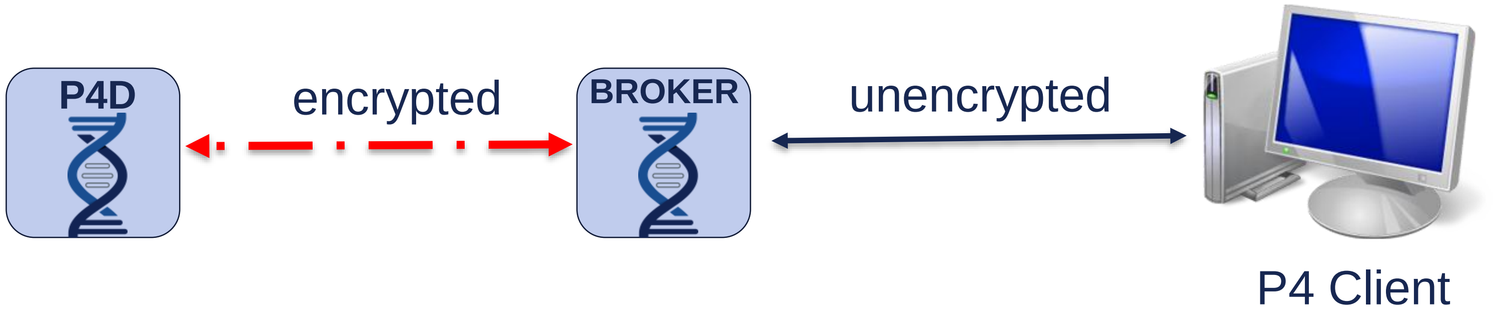
- Client needs to accept fingerprint

```
p4 -p ssl:p4server:1667 trust -y
```

PERFORCE

Phasing-in SSL encryption with P4Broker

- Use P4Broker
  - P4D runs with SSL encryption enabled
  - P4Broker itself runs unencrypted
  - Allows phasing-in of encrypted connections



Exercises

Lab Set E6: Security

New commands in this chapter:

- `p4d -Gc`
- `p4 trust`

Advanced Perforce Helix Administration

Advanced Tools

PERFORCE

Advanced Tools

- perfmerge
- perfsplit
- p4-migrate
- Checkpoint surgery
- Conversions – ftp://ftp.perforce.com/perforce/tools

perfmerge

- Goal
  - Merge two Helix Servers into a single Helix Server

- Implementation
  - Perfmerge tool reads both databases
  - Choice on change merging
    - Append
    - Intersperse and order in time
    - Append with offset

PERFORCE

perfsplit

- Goal
  - Extract data from a main server with its exact revision history
  - Split a Helix Server into two separate Helix Servers

- Implementation
  - Perfsplit reads directly from an existing Helix Server
  - It uses a splitmap to determine which files are split
    - Same syntax as the label view map
  - Only creates metadata, depot files need to be copied separately
  - You should run “p4 snap” on the directory or directories being split first

PERFORCE

p4migrate

- Goal
  - Migrate a Helix Server from a case-insensitive to a case-sensitive platform

- Implementation
  - Reads a checkpoint to find case inconsistencies
  - Generates a case-fix map
  - Use the map to correct the checkpoint
  - Once the checkpoint is case-consistent it can be used for migration
  - Tool can also be used to rename depot paths
  - Migration from case-sensitive to case-insensitive is not supported

Checkpoint/Journal Format

- Text file containing journal records
- Each record has a type
  - Checkpoint only has @pv@ entries
- Strings are surrounded by @ symbol
- Each value record refers to
  - A database table
  - The table version

Record	Type
@pv@	Put value = insert
@dv@	Delete value = delete
@rv@	Replace value = update
@vv@	Verify value = select
@ex@	commit
@mx@	flush
@nx@	Journal note

- http://www.perforce.com/perforce/doc.current/schema/

Log Analysis and Reporting

- Standard Log
  - Log Analyzer
    - Upload your logs
    - Download our tools
  - Track2SQL
- Structured Logs
- Performance monitoring using the log
- Metrics with P4toDB (replication technology)
- Discovering overall trends

PERFORCE

Structured Log

#	Structured Logs	Description
1	all	All loggable events (commands, errors, audit, etc...)
2	commands	Command events (command start, compute, and end)
3	errors	Error events (errors-failed, errors-fatal)
4	audit	Audit events (audit, purge)
5	track	Command tracking (track-usage, track-rpc, track-db)
6	user	User events; one record every time a user runs p4 logappend.
7	events	Server events (startup, shutdown, checkpoint, journal rotation, etc.)
8	integrity	Replication errors/events

Structured Logs

- Enable specific structured logs with:

```
p4 configure set serverlog.file.n=logtag.csv
p4 configure set serverlog.maxmb.n=1024
p4 configure set serverlog.retain.n=45
```

- Enabling all structured logging files can consume considerable space and impact performance.
- Structured logs are automatically rotated
  - Checkpoint or journal rotation
  - Exceeding size limit
  - When ‘p4 logrotate’ is run.

Conclusion

- Database schema is public
- Some tools use the checkpoint or the database directly
- Handle with care

- Ask Perforce support or consulting if you are not sure

Exercises

Lab Set E7: Structured Logs

New commands in this chapter (samples):

- `p4 configure set serverlog.file.`*`n`*`=errors.csv`
- `p4 configure set serverlog.maxmb.`*`n=30Mb`*
- `p4 configure set serverlog.retain.`*`n=45`*
- `p4 logappend`
- `p4 logrotate`

Advanced
Perforce Helix Administration

Scripting

PERFORCE

Preliminary Decisions

- Uses of scripts
- Choosing the interface
- Setting Environment Variables
- User Authentication

PERFORCE

Uses of scripts

- Reporting tools
- Daemons and recurring processes
- Wrappers for Helix Enterprise commands
- Triggers
- Workflow and policy enforcement
- P4V customization (P4JsAPI)
- P4Broker
- Legacy SCM data import

Typical workings of a script

- Data processing in batches
  - Retrieve information such as files or changes
  - Process the data in the script
  - Potentially update Helix Server

- Form handling
  - Retrieve a form such as a client workspace
  - Modify the form in the script
  - Update the form in Helix Server

Workflow and Policy enforcement

- Triggers
  - Submit/Shelving triggers
  - Authentication triggers
  - Form triggers
  - Archive triggers
  - Fix triggers
- P4Broker
  - Block, redirect or modify commands

PERFORCE

Choosing the interface

- Wrap P4 command
  - + Simple solution that will run everywhere
  - + Batch scripting built into the OS and requires no installation
  - • Requires parsing of output
- APIs
  - + Language-specific integration
  - + Extendable
  - + Performance (reduced connection overhead)
  - • Requires installation (and/or build/compilation)

API’s Available for Scripting

- Programming Languages and APIs
  - C++
  - P4Java
  - Objective-C
  - .NET

- Derived APIs (C++ API wrappers)
  - P4Python
  - P4Perl
  - P4Ruby
  - P4PHP

http://www.perforce.com/product/components/apis

Wrapping the command line client P4

- Command line returns lines of text

```
p4 describe -s 13
Change 13 by sknop@alita on 2015/03/02 12:58:51

 Branching foo from bar.
 Test branch only.

Affected files ...

... //depot/tests/foo#1 branch
```

PERFORCE

Capture errors, warnings and messages

- Use -s to precede each output line with “info” or “error”

```
p4 -s sync ...
info: //depot/foo#3 – updating /client/foo
error: Can't clobber writable file /client/foo
exit: 1
```

Tagging output: Command line and API

- Format output by using -ztag

```
p4 -ztag clients
... client bruno_ws
... Update 1104271684
... Access 1104340062
... etc.
```

- Helix Server API based on tagged data output

Form handling: bypassing an editor

- Redirect to standard output

```
p4 change -o
```

- Read from standard input

```
p4 submit -i
```

- Submit without invoking an editor

```
p4 submit -d "Fixed off-by-one error."
```

- Example: Create a client workspace without an editor

```
p4 client -o | p4 client -i
```

Setting the environment for scripts

- Command line flags

```
p4 -p server:1666 -u script_user -c script_ws info
```

- P4CONFIG (next slide)
- Environment and registry variables

- Recommendation:
  - Use P4CONFIG
  - Set P4CONFIG in the scripts to make sure it is set in the environment
  - Keeps scripts independent of Helix Server and current directory location

P4CONFIG

- P4CONFIG points to a file name

```
p4 set P4CONFIG=P4Config.txt

export P4CONFIG=/p4/scripts/.p4config
```

- File usually located in the workspace root or scripts folder

- File contains the Helix Server variables

```
  P4PORT=server:1666
  P4CLIENT=script_ws
  P4USER=script_user
```

User authentication for scripts

- `p4 login`
  - Works for all Helix Server security levels
  - Works if Helix Server is integrated with AD
  - Works if Helix Server is integrated with SSO
- Either: Store password locally (hidden/restricted) file

```
p4 login < /p4/scripts/.password
```

- Or: Use ever-lasting ticket (ideally in separate P4TICKETS file)

PERFORCE

Use a group to extend session

```
p4 group scripts
```

```
Group:   scripts
MaxResults:     1000000
Maxscanrows:    5000000
MaxLockTime:      30000
Timeout:      unlimited
Subgroups:
Owners:
              bruno
Users:
              script_user
```

PERFORCE

P4TICKETS

- P4TICKETS points to a ticket file

```
export P4TICKETS=/p4/scripts/.script_p4tickets
```

- Important when scripts may be run as a different user (default value is home directory which is different per user)
- Provides safety from accidently logging out a script user
  - Beware of `p4 -u script_user logout -a`
    - Invalidates all tickets for this user

Questions?

The End

All Perforce manuals and technical notes are available at
www.perforce.com

Follow and participate with the Perforce Community and Forums at

www.perforce.com/community

workshop.perforce.com

Report problems and get technical help from
support@perforce.com

PERFORCE