Perforce Best Practices for Codeline Management
Introduction

• Introductions

• Class Schedule

• About the Exercises
Course audience

- Experienced Perforce users
- Individuals whose work includes:
  - branching and integrating
  - developing branching and integration policy
  - configuration management
  - or, builds and releases
Prerequisites

• Completion of “Perforce for Users and Administrators” or equivalent experience

• Experience with Perforce as both user and administrator and, specifically, solid knowledge of the Perforce commands
Codeline management objectives

To learn best practices for:

• Organizing your codelines
• Managing parallel development
• Performing codeline operations
• Maintaining server performance
Best Practices for Codeline Management

Topics

• Organizing Codelines
• Managing Parallel Development
• Branching Techniques
• Maintaining Server Performance
Best Practices for Codeline Management

Organizing Codelines
Module objectives

• Organize codelines using “mainline” model
• Establish naming conventions for codelines
• Optimize server performance
Organizing Codelines - Topics

- Terms
- Recommended Depot Structure
- Performance Tips
- Codeline Management Policy
Definitions

• **Product**
  - A named, deliverable set of files
• **Project**
  - Development effort that produces a product
• **Module**
  - Subset of related project files
• **Codeline**
  - Set of modules that evolve together
  - Same phase of development
  - Includes components, subsystems, etc.
The mainline model

- A common model of software development
- Perforce supports the mainline model
- Main is the “central” codeline
- Branch codelines from main for
  - Software releases
  - New development
  - Other purposes based on codeline policy
- Changes normally flow back to main
Release codelines branch from main

MAIN

REL1.0
REL2.0

Time
Development codelines branch from main

- Bug fixes flow to dev branches from main
- New development flows to main as completed
Other branching models supported

- Promotion common in Web development
Codeline relationships

- Branching lineage
- Relative stability
Codeline firmness/stability

- The “tofu” scale
  - Firm
    - Release codelines
    - Thoroughly tested
  - Medium
    - Main codeline
  - Soft
    - Development codelines
    - Unstable
    - Barely tested

Laura Wingerd, Practical Perforce pp. 176-177, O’REILLY 2006
The flow of change: mainline model
Organizing Codelines

Exercise

Organizing Codelines, Part 1
Organizing Codelines

Recommended Depot Structure
References

• Laura Wingerd, *Practical Perforce*, Chapter 8

• Perforce Website Knowledge Base Article, “Planning Your Server Installation”
Name depots using project names

- A unique name avoids namespace collision
  
  //Sim/…

- Performance tip:
  
  Map depots to a volume separate from P4ROOT
Map depot to a file system

**p4 depot Mars**

- **Depot:** Mars
- **Owner:** bruno
- **Date:** 2010/06/12 14:41:04
- **Description:** Mars topography model
- **Type:** local
- **Address:** local
- **Map:** /filer/perforce/Mars/...
Grouping by functionality

• Product development and release
  //Sim/Prod/...

• Private development
  //Sim/Priv/...

• Short lived tasks
  //Sim/Task/...

• Website development
  //Sim/Web/...

Naming CODELINES

• Release codeline
  //Sim/Prod/REL2.1/...

• Main codeline
  //Sim/Prod/MAIN/...

• Development codeline
  //Sim/Prod/DEV/...

• New feature codeline
  //Sim/Prod/FLAT/...
Group files into modules

//Sim/Prod/MAIN/src/…
//Sim/Prod/MAIN/tests/…
//Sim/Prod/MAIN/utils/…
Review of recommended depot structure

// Project / Functionality / CODELINE / module / file

//Sim/Prod/MAIN/src/sim.c
Creating a release codeline

`p4 integ -b SimMAINtoREL2.2`
```
//Sim/Prod/REL2.2/src/Build.com#1 - branch/sync
  from //Sim/Prod/MAIN/src/Build.com#1,#7
//Sim/Prod/REL2.2/src/Build.mpw#1 - branch/sync
  from //Sim/Prod/MAIN/src/Build.mpw#1,#2
//Sim/Prod/REL2.2/src/command.c#1 - branch/sync
  from //Sim/Prod/MAIN/src/command.c#1,#8
... etc.
```

`p4 submit`
Perforce stores which revisions have been integrated

```
p4 filelog //Sim/Prod/REL2.2/src/command.c
... #1 change 876 branch on 2007/11/23 by ines@ines-rose
{text} 'Create Sim 2.2 Release branch. '
... ... branch from //Sim/Prod/MAIN/src/command.c#1,#8
```
Stored integration credits for a new branch

```
p4 filelog //Sim/Prod/REL2.2/src/glob.c
... #1 change 876 branch on 2007/11/23 by ines@ines-rose
  (text) 'Create Sim 2.2 Release branch. '
... ... branch from //Sim/Prod/MAIN/src/glob.c#1,#6
```

- All source file revisions (1-6) have credit
- Target file revision 1 has credit
Integration credits for glob.c

//Sim/Prod/REL2.2/src/glob.c

= Integration credit

Revisions with credit will not be reintegrated
Credits are used in future integrations

- To eliminate revisions already integrated
- To find a base for the integration

Perforce Website Knowledge Base Article, “Determining Revisions to Integrate”
Create codelines

• Based on sound policy
• Judiciously for future scalability
• Including only files actively under development
• With short paths
• In new server if:
  • Current projects are becoming inactive
  • Dependencies can be managed appropriately
Developing a codeline policy

• Does each codeline have a “curator”?  
• Are development responsibilities clear? 
• Does depot structure work for build tools?  
• Are protections appropriate?  
• Is a client workspace template available?
Organizing Codelines

Exercise

Organizing Codelines, Part 2
Best Practices for Codeline Management

Parallel Development
Module Objectives

• Branch codelines based on sound policy
• Follow best practices for full branches
• Select best branching method
• Use collaboration tools
Parallel Development - Topics

• Perforce Branching and Merging Concepts
• Full and Sparse Branches
• Branching Techniques
Parallel Development

Branching and Merging Concepts in Perforce
Integration

• “The business of accounting for every change…”
• Integrating codelines can involve
  • merging file contents
  • ignoring changes
  • propagating renamed files
  • copying files
  • propagating new files
  • propagating file deletions

p4 resolve result determines integration method

Laura Wingerd, *Practical Perforce* p. 87, 124.
When resolving integrations

p4 resolve
c:\raj-fir\Sim\Prod\MAIN\src\sim.c - merging //Sim/Prod/FLAT/src/sim.c#2
Diff chunks:
1 yours + 1 theirs + 0 both + 0 conflicting
Accept(a) Edit(e) Diff(d) Merge (m) Skip(s) Help(?) am:

Yours = Target
Theirs = Source
Base = Closest common ancestor
Resolve’s accepting actions

- **am** Accept merged – auto merge source and target
- **ae** Accept edited – you have edited the merged file
- **at** Accept theirs – copy source to target
- **ay** Accept yours – ignore source, keep target
Integration history of a branch

**p4 integrated** //Sim/Prod/MAIN/src/...
//Sim/Prod/MAIN/src/compile.c#1,#12 - branch into //Sim/Prod/FLAT/src/compile.c#1
//Sim/Prod/MAIN/src/compile.c#26,#29 - delete into //Sim/Prod/FLAT/src/compile.c#4
//Sim/Prod/MAIN/src/compile.h#6,#7 - edit into //Sim/Prod/FLAT/src/compile.h#4
//Sim/Prod/MAIN/src/execcmd.h#5 - delete from //Sim/Prod/FLAT/src/execcmd.h#2,#5
//Sim/Prod/MAIN/src/execmac.c#1 - merge from //Sim/Prod/FLAT/src/execmac.c#2,#3
//Sim/Prod/MAIN/src/execmac.c#4 - ignored //Sim/Prod/FLAT/src/execmac.c#3
//Sim/Prod/MAIN/src/execvms.c#3,#5 - merge into //Sim/Prod/FLAT/src/execvms.c#5
//Sim/Prod/MAIN/src/execvms.c#1,#7 - branch into //Sim/Prod/FLAT/src/execvms.c#1
## Decoding integration history

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Source</th>
<th>Target</th>
<th>Integration credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch</td>
<td>from</td>
<td>into</td>
<td>all source &amp; target revisions</td>
</tr>
<tr>
<td>copy</td>
<td>from</td>
<td>into</td>
<td>all source &amp; target revisions</td>
</tr>
<tr>
<td>add move</td>
<td>branch from</td>
<td>into</td>
<td>all source revisions</td>
</tr>
<tr>
<td></td>
<td>from</td>
<td>into</td>
<td>(source deleted)</td>
</tr>
<tr>
<td>merge</td>
<td>from</td>
<td>into</td>
<td>source revision range &amp; target</td>
</tr>
<tr>
<td>ignored</td>
<td></td>
<td>by</td>
<td>source revision range &amp; target</td>
</tr>
<tr>
<td>delete</td>
<td>from</td>
<td>into</td>
<td>source revision range &amp; target</td>
</tr>
<tr>
<td>edit</td>
<td>from</td>
<td>into</td>
<td>source revision range range</td>
</tr>
</tbody>
</table>
Integration credit when merging files

```
p4 integ -b SimMAINtoFLAT sim.c
p4 resolve -am

//Sim/Prod/MAIN/src/sim.c

//Sim/Prod/FLAT/src/sim.c

p4 submit
```
Integration credit when editing files

```
p4 integ -b SimMAINtoFLAT sim.h
p4 resolve -ae
//Sim/Prod/MAIN/src/sim.h
p4 submit
//Sim/Prod/FLAT/src/sim.h
```

Edit during resolve

```
//Sim/Prod/FLAT/src/sim.h
```
Complete integ credit when copying files

```
p4 integ -r -b SimMAINtoFLAT scan.h
p4 resolve -at
//Sim/Prod/MAIN/src/scan.h

//Sim/Prod/FLAT/src/scan.h
```

```
p4 submit
```

Copy
Incomplete integ credit when ignoring files

```
p4 integ -r -b SimMAINtoFLAT parse.c
p4 resolve -ay

//Sim/Prod/MAIN/src-parse.c

1  2  3  4  5

Ignore

1  2  3

//Sim/Prod/FLAT/src-parse.c

p4 submit
```
Integration credits are used to find a base

```
p4 integ -r -b SimMAINtoFLAT

//Sim/Prod/MAIN/src/glob.c
```

```
p4 resolve -am
p4 submit
```

```
//Sim/Prod/FLAT/src/glob.c
```
Finding a base with unintegrated revisions

```
p4 integ -r -b SimMAINtoFLAT sim.c
```

```
//Sim/Prod/MAIN/src/sim.c
```

```
p4 resolve -am
p4 submit
```

```
//Sim/Prod/FLAT/src/sim.c
```
Merging codelines

- Can propagate source files that have been:
  - edited
  - added
  - deleted
  - moved
  - modified by a file type change

- Codeline policy tip:
  - Store source/target pairs in branch spec
Parallel Development

Full and Sparse Branches
Full branches

- Mainline – All project files
- Release – Stabilization branch
- Development – Ongoing codeline changes
- Staging – Deployment for specific purpose
When working on a full branch

- Keep workspace current
- Submit changes as logical units of work
- Include in changelist description:
  - Files or directories renamed
  - Formatting changes
  - Structural changes
  - Explanation why files added or deleted
Integrating full branch changes

- Integrate often, keeping codelines in sync
- Integrate one changelist at a time
- Test merged result before submitting
Sparse branches

- Small number of files are branched for
  - Specific tasks
  - Editing by a single user
  - Short-lived development

- Advantages
  - Less metadata created on the server
  - Risky bug fixes can be isolated
  - Files reviewed before merging to parent
When to avoid using a sparse branch

- Code refactoring for improved design
- Changes involving more than one person
- When files are added or deleted in parent codeline
- Long term development
Perforce collaboration tools

- Changelist descriptions
- Shelving files
- Script-generated P4Web URL’s
- P4V:
  - Folder diff
  - Revision graph
  - Time-lapse view
- Email, CGI scripts, etc.
- P4Web using RSS feed
Change - 875: Misc sybase changes: split MAXL

Date: 2008/02/19 13:13
Client: bruno_ws
User: bruno
Description:

Misc sybase changes: split MAXLINE and CMDBUF so that they can be separate sizes on NT.

Affected files:
//Sim/Prod/MAIN/src/command.c#8 edit

Article:

http://localhost:8020/@md=c&cd=//Sim/Prod/MAIN/&pgc=y875?ac=10
External collaboration tools

- SmartBear Software
  - CodeCollaborator

- Atlassian
  - Crucible
  - Fisheye

- CodeStriker
  - CodeStriker
Parallel Development

Exercise
Parallel Development, Part 1
Parallel Development

Branching Techniques
Use appropriate methodology

- Sparse branch for a specific short-lived task
- Staging codeline for web site development
- Map virtual modules into workspace
- Private codeline for informal work
Using a sparse branch

- Create a client workspace spec
- Integrate files using a branch spec
- Edit files
- Sync your client workspace for testing
- Get files reviewed
- Integrate approved files to parent codeline
Isolate a bug fix for review and testing

//Sim/Task/BUG892/...
Sparse branch client workspace view

p4 client ann-Bug892

Client: ann-BUG892
Root: c:\Fix-Bug892
View:

//Sim/Prod/REL1.0/src/...  //ann-BUG892/src/...
+//Sim/Task/BUG892/src/...  //ann-BUG892/src/...
Branch specification

p4 branch SimREL1.0toBug892

Branch: SimREL1.0toBUG892
Owner: ann

View:

//Sim/Prod/REL1.0/src/sim.c  //Sim/Task/BUG892/src/sim.c
//Sim/Prod/REL1.0/src/sim.h  //Sim/Task/BUG892/src/sim.h
//Sim/Prod/REL1.0/src/yyacc  //Sim/Task/BUG892/src/yyacc
Integrate files to the sparse branch

p4 integ -b SimREL1.0toBug892

p4 submit
Working on the sparse branch files

- Get all files required from server
  `p4 sync`

- List files in sparse branch
  `p4 files //Sim/Task/BUG892/src/...`

- Edit file in sparse branch
  `p4 edit //Sim/Task/BUG892/src/sim.c`
Merging approved changes into parent

p4 sync //Sim/Prod/REL1.0/src/...

p4 integ -r -b SimREL1.0toBUG892

p4 resolve

p4 submit
Using staging codelines

- Staging branch
  - Continuous integration to release codeline
  - Web site development a typical use

- Performance tip:
  - Reuse codeline namespace to prevent excessive branch proliferation
Using active and virtual modules

• **Active module**
  • Actively under development
  • Explicitly mapped into client workspace
  • Branch spec stores relation to parent

• **Virtual module**
  • Required in workspace, but unmodified
  • Mapped into client workspace from parent
  • Files are not branched

• **Use both in a codeline for performance**
  • Branch active modules
  • Map virtual modules into client workspace
Examples

- Active
  //Sim/Prod/MAIN/src/...
  //Sim/Prod/MAIN/tests/...

- Virtual
  //Sim/Prod/MAIN/utils/...

Laura Wingerd, *Practical Perforce*, p. 252.
Branch specification for active modules

p4 branch SimMAINtoDEV

Branch: SimMAINtoDEV
Owner: raj
Description:
   Sim Project Development branch.
   Utilities are not branched from MAIN.
Options: locked
View:
   //Sim/Prod/MAIN/src/... //Sim/Prod/DEV/src/...
   //Sim/Prod/MAIN/tests/... //Sim/Prod/DEV/tests/...
Example: development client workspace

`p4 client jan-dev`

Client: jan-dev
Owner: jan
Root: c:\Sim-Dev

View:

//Sim/Prod/DEV/src/... //jan-dev/src/...
//Sim/Prod/DEV/tests/... //jan-dev/tests/...
//Sim/Prod/MAIN/utils/... //jan-dev/utils/...
When to use private codelines

- For informal work
  //Sim/Priv/BRUNO/...

- For code review
  - Reviewers check files
  - Notify author when files are approved

- For working offline with P4V
Reviewing shelved files
Communicating via changelist

Submitted Changelist: 849 (3366, bruno)

Changelist: 849
Workspace: bruno-sim
Date: 3/7/2008 2:35:22 PM
User: bruno

Description:
Add global parameters for OSF.
Reviewed by: Earl Ashby
Code passed review, ready for integration into MAIN.

Files:
- //Sim/Priv/BRUNO/src/compile.c
- //Sim/Priv/BRUNO/src/expand.c
- //Sim/Priv/BRUNO/src/expand.h
- //Sim/Priv/BRUNO/src/glob.c

Expand All

Job | Status | Description
--- | --- | ---

Job: Add  Browse...  OK  Cancel  Apply
Working offline
Bringing workspace online
Parallel Development

Exercise

Parallel Development, Part 2
Best Practices for Codeline Management

Performing Codeline Operations
Module objectives

• Follow best practices for propagating renames
• “Merge down, copy up” to ensure convergence
• Exercise caution with “cherry-picked” integrations
• Handle added and deleted files appropriately
• Back out unwanted changes
Codeline Operations - Topics

• Propagating Renames
• Integration Policy
• “Cherry Picking”
• Creating Divergence
• Ensuring Convergence
• Handling Adds and Deletes
• Reversing Unwanted Changes
• Scripting Opportunities
Typical goals in codeline management

- Merging correctly without losing changes
- Copying files without losing work
- Preserving divergence
- Encouraging convergence
Renaming compile.h to recompile.h

What are the consequences using the “-d” flag?

```
p4 integ -d MAIN/src/... FLAT/src/...
//Sim/Prod/FLAT/src/compile.h#3 - delete from //Sim/Prod/MAIN/src/compile.h#6
//Sim/Prod/FLAT/src/recompile.h#1 - branch/sync from //Sim/Prod/MAIN/src/recompile.h#1,#2
```
Propagating a renamed file correctly (old)

```
p4 integ FLAT/src/compile.h FLAT/src/recompile.h
p4 delete FLAT/src/compile.h
p4 submit
p4 integ MAIN/src/recompile.h FLAT/src/recompile.h
p4 resolve
p4 submit
```
Propagating a renamed file correctly (new)

```
p4 edit  FLAT/src/compile.h
p4 move  FLAT/src/compile.h FLAT/src/recompile.h
p4 submit
p4 integ -3 MAIN/src/recompile.h FLAT/src/recompile.h
p4 resolve  (if needed)
p4 submit
```
Integration policy

- Merge “down” into softer codeline
  - Resolve edited files
  - Add new files
  - Delete obsolete files

- Copy “up” into firmer codeline
  - First, merge outstanding edits from firm codeline
  - “Freeze” firm codeline
  - Integrate and resolve, accepting “theirs”
p4 integ -r -b SimMAINtoREL1.0
p4 resolve -am
p4 submit
Merge “down”, copy “up”

Freeze MAIN

p4 integ -b SimMAINtoDEVZ
p4 resolve -am
p4 submit
p4 integ -r -b SimMAINtoDEVZ
p4 resolve -at
p4 submit
Freezing a codeline

• Why freeze a codeline?
• How can Perforce be used to enforce freeze?
• How can a freeze be announced?
• Are there alternatives to freezing a codeline?
• What are disadvantages of a freeze?
• A freeze should be short-lived
Using protections to enforce a freeze

Protections:

write group simdev  *  //Sim/Prod/MAIN/src/...
write user ann  *  //Sim/Prod/MAIN/src/...

p4 protect

Change “write” to “open”; afterward change back.
Example of “cherry picking”

```
p4 integ -r -b SimMAINtoFLAT //...glob.c#4,4
p4 resolve -am
p4 submit
```
“Cherry picked” integration credits

```
p4 integ -r -b SimMAINtoFLAT //...glob.c#4,4
p4 resolve -am

//Sim/Prod/MAIN/src/glob.c
```

```
p4 submit
```

```
//Sim/Prod/FLAT/src/glob.c
```
Example of “cherry-picked” merge.
How “accepting yours” creates divergence

```
p4 integ -r -b SimMAINtoFLAT //...glob.c
p4 resolve -ay
p4 submit
```

```
p4 integ -r -b SimMAINtoFLAT
//Sim/Prod/FLAT/src/glob.c - all revision(s) already integrated.
```

```
p4 diff2 -q -b SimMAINtoFLAT

==== //Sim/Prod/MAIN/src/glob.c#6 (text) - //Sim/Prod/FLAT/src/glob.c#4 (text) ==== content
```
Factors that create codeline divergence

- Files may be added to one branch only
- Files may be deleted from one branch only
- “Cherry picking” and “accept yours” resolves can cause codeline divergence
- Renames may not get propagated
- Divergent codelines may
  - continue to diverge
  - converge to a common base
How to ensure codeline convergence

- Merge “down”
- “Copy” up
- Copy files that differ

First: merge “down”

- Check for merge “down” candidates
  ```
  p4 integ -n -b SimMAINtoFLAT
  //Sim/Prod/FLAT/src/rules.c#1 – integrate from //Sim/Prod/MAIN/src/rules.c#6
  ```

- Merge outstanding edits into source
  ```
  p4 integ -b SimMAINtoFLAT
  //Sim/Prod/FLAT/src/rules.c#1 – integrate from //Sim/Prod/MAIN/src/rules.c#6
  p4 resolve -am
  p4 submit
  ```
Next: copy “up”

- Perform standard integration
  
p4 integ -r -b SimMAINtoFLAT
  //Sim/Prod/MAIN/src/glob.c#5 - integrate from //Sim/Prod/FLAT/src/glob.c#21

- Copy the source files to the target
  
p4 resolve -at
  p4 submit
Finally: copy files that differ

- Use `diff2` to list files that differ
  
  ```bash
  p4 diff2 -q -b SimMAINtoFLAT
  //Sim/Prod/MAIN/src/command.c#10 (text)
  //Sim/Prod/FLAT/src/command.c#3(text) content
  ...
  etc.
  ```

- Then force integrate the files
  
  ```bash
  p4 integ -f -r -b SimMAINtoFLAT command.c
  ...
  etc.
  ```

- Copy source files to target
  
  ```bash
  p4 resolve -at
  p4 submit
  ```
Codeline Operations

Exercise
Codeline Operations, Part 1
Handling added or deleted files

• Use a branch spec to maintain divergence
  • Exclude adds, deletes or renamed files
  • Exclude incompatible file types
  • Map differing file names

• Convergence can be forced with p4 integ -d
  • Be aware edits may be lost
  • Can either delete or re-add a file
Branch specification for divergent files

p4 branch SimMAINtoFLAT

Branch: SimMAINtoFLAT
Owner: raj
Description:
    Sim Project FLATLAND branch.
Options: locked
View:
    //Sim/Prod/MAIN/src/... //Sim/Prod/FLAT/src/...
    -//Sim/Prod/MAIN/src/scan.c //Sim/Prod/FLAT/src/scan.c
    -//Sim/Prod/MAIN/src/fscan.c //Sim/Prod/FLAT/src/fscan.c
    //Sim/Prod/MAIN/src/flame.h //Sim/Prod/FLAT/src/fire.h
Codeline Operations

Reversing Unwanted Changes
Reversing a changelist

• Deleting added files
• Adding deleted files
• Reversing edits
• Reversing integrations
View changelist and populate workspace

**p4 files @826,826**

- //Sim/Prod/FLAT/src/Simbase#2 - delete change 826
- //Sim/Prod/FLAT/src/mksim.c#2 - edit change 826
- //Sim/Prod/FLAT/src/simbase.c#2 - edit change 826
- //Sim/Prod/FLAT/src/simbase.h#2 - edit change 826
- //Sim/Prod/FLAT/src/simrbase#1 - add change 826

**p4 sync #none**

**p4 sync @826,826**
Open files for delete or add

```
p4 files simrbase@826  
//Sim/Prod/FLAT/src/simrbase#1 - add change 826
```
```
p4 delete simrbase
```
```
p4 files Simbase@826  
//Sim/Prod/FLAT/src/Simbase#2 - delete change 826
```
```
p4 sync Simbase@825
```
```
p4 add Simbase
```
Reverse edits and submit

```
p4 sync //Sim/Prod/FLAT/src/mksim.c@825
p4 sync //Sim/Prod/FLAT/src/simbase.c@825
p4 sync //Sim/Prod/FLAT/src/simbase.h@825
p4 edit *
p4 sync ...@826
p4 resolve -ay
p4 submit
```
Reversing integrations

• List past integrations

**p4 integrated //Sim/Prod/FLAT/src/mksim.c**

- //Sim/Prod/MAIN/src/mksim.c#1,#4 - branch into //Sim/Prod/FLAT/src/mksim.c#1
- //Sim/Prod/REL2.1/src/mksim.c#3 - ignored //Sim/Prod/FLAT/src/mksim.c#2,#4
- //Sim/Prod/MAIN/src/mksim.c#5 - merge from //Sim/Prod/FLAT/src/mksim.c#4

... etc.

• Should edits be reversed in other codelines?
Codeline Operations

Scripting Opportunities
Candidate operations

- Enforcing codeline policy
- Maintaining divergent codelines
- Ensuring convergent codelines
- Integration techniques
Scripting questions

- What routine activities can be scripted?
- What problems can be avoided with scripting?
- Can scripting provide safety net for users?
Codeline management – key points

• Organize codelines for collaboration
• Branch for parallel development
• Common operations
  • Insight into Perforce integration records
  • Recipes for performing standard tasks
  • Best practices recommendations
• Tips for maintaining server performance
• Incorporate these ideas into your codeline policy
Codeline Operations

Exercise
Codeline Operations, Part 2
The End

Please fill out class evaluations.

All Perforce manuals and technical notes are available on our web page, www.perforce.com.

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

Share tips and ideas with other users on perforce-user@perforce.com