



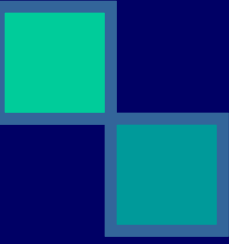

“From the files of a Perforce  
Consultant...”

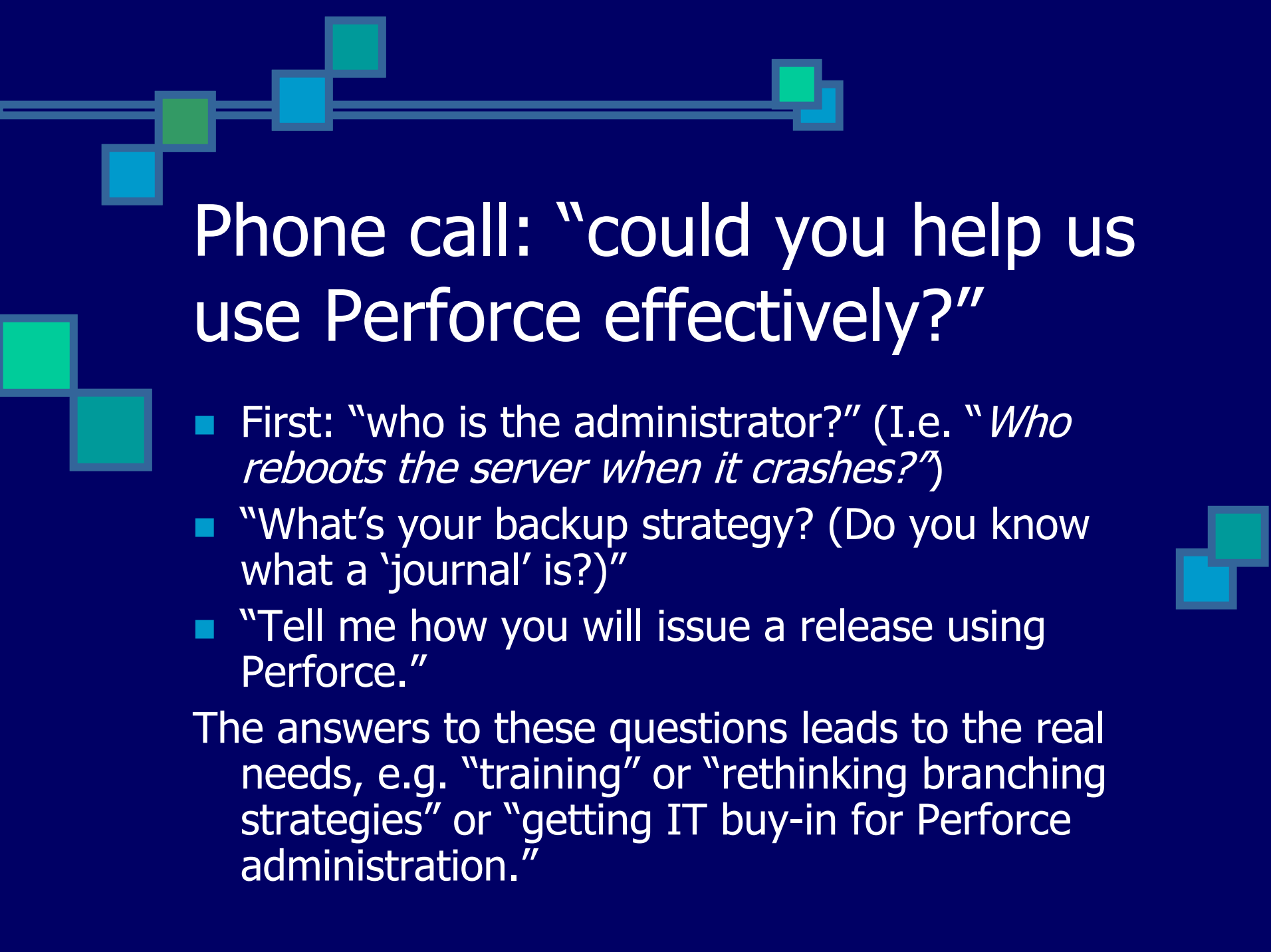


*An annotated list of surprises, good and bad,  
by Jeff Bowles  
Piccolo Engineering, Inc.*



# Types of questions

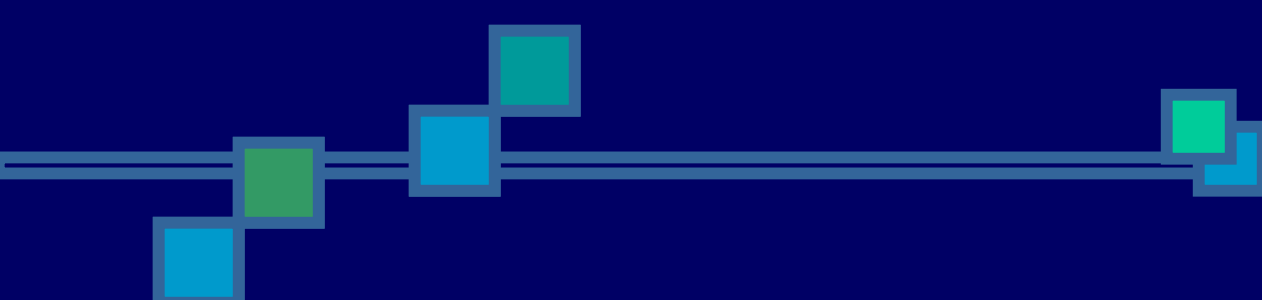
- 
- Use Perforce “more effectively”
  - Get a “tune-up”
  - Anticipating future needs/problems before they occur
- 




# Phone call: “could you help us use Perforce effectively?”

- First: “who is the administrator?” (I.e. “*Who reboots the server when it crashes?*”)
- “What’s your backup strategy? (Do you know what a ‘journal’ is?)”
- “Tell me how you will issue a release using Perforce.”

The answers to these questions leads to the real needs, e.g. “training” or “rethinking branching strategies” or “getting IT buy-in for Perforce administration.”




## Followup: “Where to put a new project?”

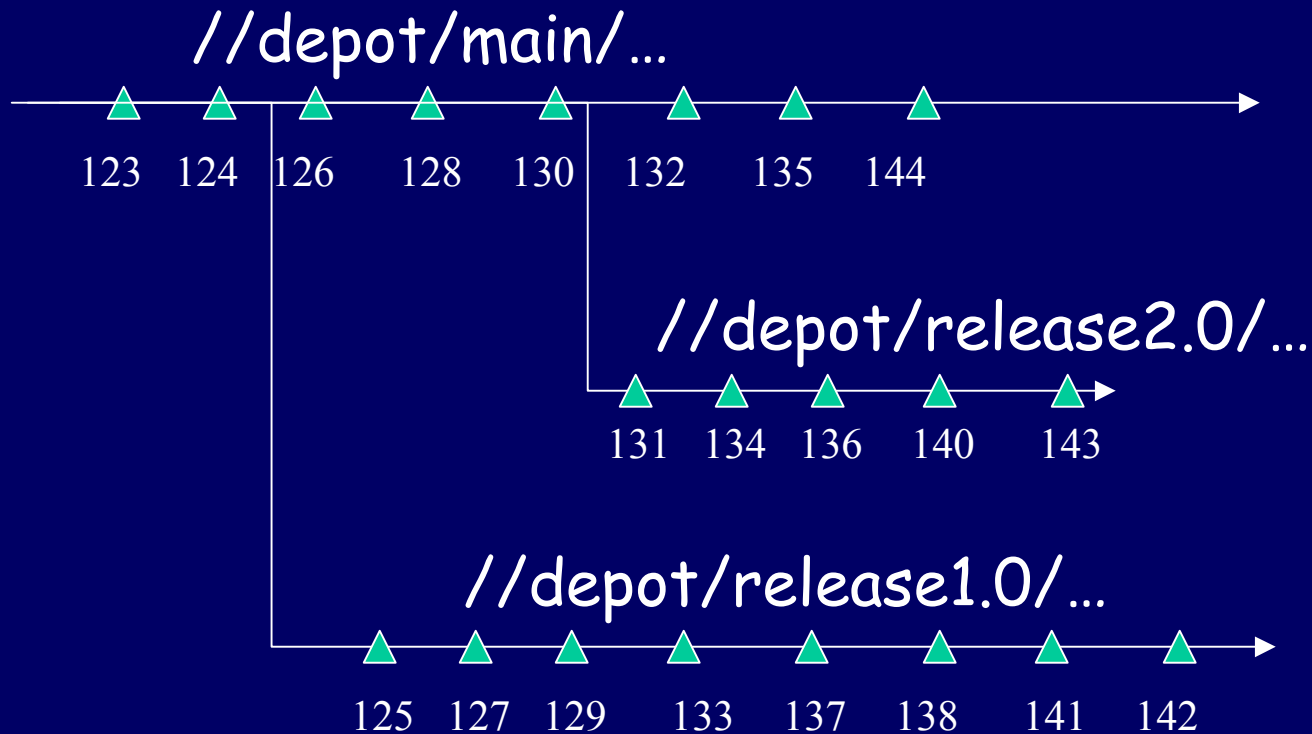
- “p4 add” is good enough.
  - But put the files into the right directory structure first: no reason to add first, then rename, except perhaps to preserve a history.
  - “ant” or other incremental-compile Java environments can help spot source files that live in the wrong place in the tree.
- 



# The tune-up request

- Trying to anticipate future problems:
    - ❖ Making a release (branching)
    - ❖ Restoring from catastrophe
    - ❖ Storing a web site in Perforce
  - Buying insurance, so to speak.
  - Spreading around the work since the Perforce admin is busy, overworked, or gone
- 


# Review of 'mainline' strategy



So `//depot/release1.0/...@138`  
is an immutable designation of a source tree!



# The tune-up: branching


- Reworking the “mainline”:
    - ❖ Sometimes it’s easiest to make a “//depot/newmain/...” (branched from the older “//depot/main/...”) that is in the new structure.
  - Changing an unworkable branch strategy:
    - ❖ “1.0 is parent of 2.0 is parent of 2.5 is parent of...” has caused a number of headaches.
  - Implementing “component-ware” strategy.
- 



# The tune-up: branching - 2



Reworking the "mainline":

- Sometimes, you just map renamed files in parent to the original name in a child, using "p4 branch" specifications.
  - For major work, it's sometimes necessary to make a new "main codeline" called "//depot/newmain/..." to have a clean place to start.
- 


*Of course, you obsolete //depot/main/... at the same time, using "p4 protect" to take away 'write' permission for most folks.*



# The tune-up: branching - 3




“1.0 is parent of 2.0 is parent of 2.5 is parent of...”  
has caused a number of headaches.

- 
- A good way out of this is to take the most current codeline and declare it to be the new “main” (or branch it to create the new “main”) and then integrate from other codelines.
  - Beware of “p4 resolve -at” (lazy copy) in this case. You really want the merge-into operation for any new integrates into the new main.



# The tune-up: branching - 4

- Implementing “component-ware” strategy. For example, App 4.0 consists of:
    - ❖ CoreLib version 1.0
    - ❖ GUILib version 1.3
    - ❖ GraphicsLib version 2.56
    - ❖ AppSrc version 4.0
  - Question: should we use labels or branches to do this?
- 




# The tune-up: branching - 5



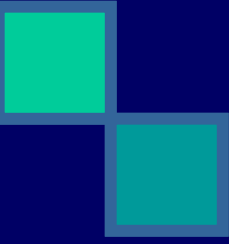

Commands to create codeline might be:

```
p4 integ //depot/main/core/...@core_v1 //depot/app4.0/core/...
p4 integ //depot/main/gui/...@gui_v1.3 //depot/app4.0/gui/...
(and so on)
p4 submit
```

- 
- Of course, you'd use a branch specification to do this ("p4 branch"), for integrations both directions.
  - Advantages: tracks history, creates a consistent source tree for app 4.0.




# The tune-up request

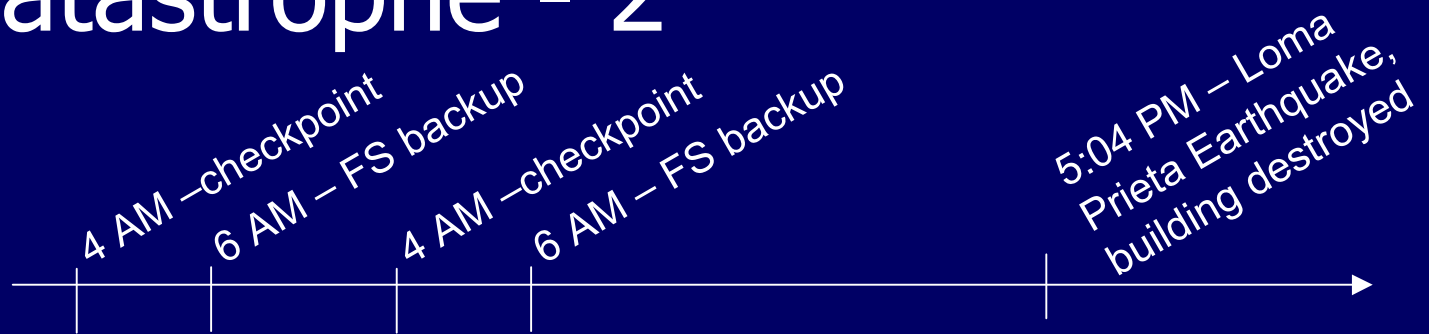
- 
- Trying to anticipate future problems:
    - ❖ Making a release (branching)
    - ❖ **Restoring from catastrophe**
    - ❖ Storing a web site in Perforce
  - Buying insurance, so to speak.
  - Spreading around the work since the Perforce admin is busy, overworked, or gone
- 



# The tune-up: restoring from catastrophe

- First question: “is this an intellectual exercise or is something broken? (How badly?)”
  - Journal should be turned on for every production environment. Always.
  - The key question is “when the earthquake hits and destroys your computers and backups, how do you restore?”
- 

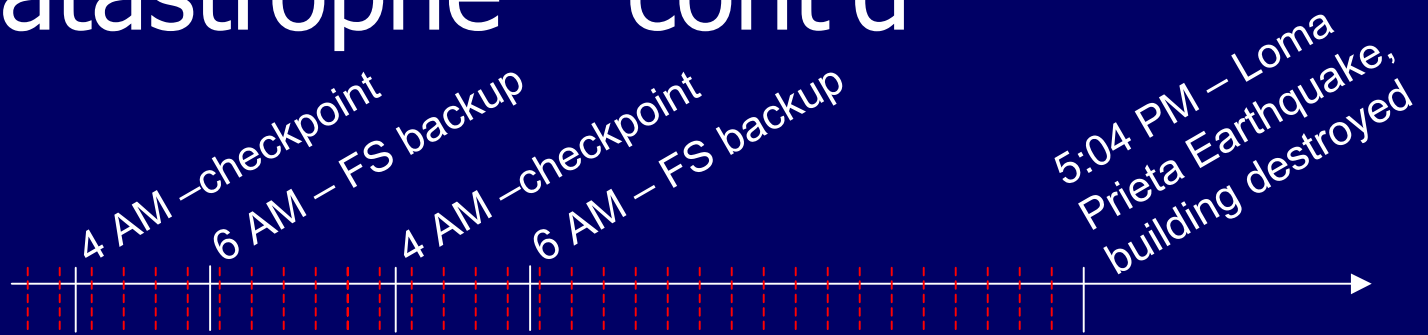
# The tune-up: restoring from catastrophe - 2



- Restore to this morning at 4 AM: trivial. Restore the checkpoint and depot/\* tree.
- Restore to this morning at 6 AM: trivial. Restore checkpoint, journal, and depot/\* tree.

*Of course, your filesystem backups are off-site, right?  
How long to retrieve them? Hours? Weeks?*

# The tune-up: restoring from catastrophe – cont'd




- *Restore to 5:04 PM failure. Trivial? Hard?*

*What if we run an incremental copy of the journal and depot/\* trees to an off-site server every 20 minutes during the day? Then we could recover to within 20 minutes of the crash.*

*You decide what exposure you're comfortable with.*

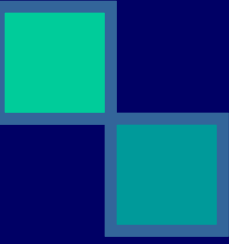



# The tune-up request

- Trying to anticipate future problems:
    - ❖ Making a release (branching)
    - ❖ Restoring from catastrophe
    - ❖ **Storing a web site in Perforce**
  - Buying insurance, so to speak.
  - Spreading around the work since the Perforce admin is busy, overworked, or gone
- 

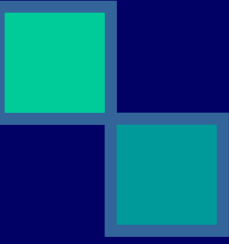



# The tune-up: store a web site

- 
- Strategy #1: map DocumentRoot area into Perforce as a workspace, run “p4 sync” frequently (or when files updated in Perforce).
  - Strategy #2: Use Apache plug-in (WebKeeper) on Unix.
  - Strategy #3: Use ‘p4ftp’ to update files in Perforce from your HTML editor, use one of the other [two] strategies to then put them into web site.
- 

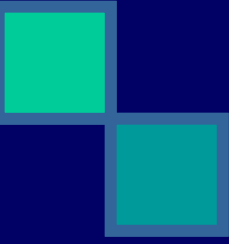



# The tune-up request

- 
- Trying to anticipate future problems:
    - ❖ Making a release (branching)
    - ❖ Restoring from catastrophe
    - ❖ Storing a web site in Perforce
  - **Buying insurance, so to speak.**
  - Spreading around the work since the Perforce admin is busy, overworked, or gone
- 




# The tune-up: buying insurance

- 
- Often, an engineering manager wants an “audit” to make sure that the assumptions made are good ones:
    - ❖ Backups, release strategies make sense;
    - ❖ To hear alternative approaches to problems everyone seems to have.
    - ❖ To have an outside source say certain things, usually “we need to have an owner for builds or release engineering.”
- 

Ideal: make sure that all procedures are checked in and also stored in hard-copy somewhere.

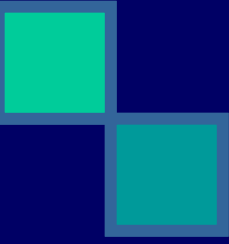



# The tune-up request

- Trying to anticipate future problems:
    - ❖ Making a release (branching)
    - ❖ Restoring from catastrophe
    - ❖ Storing a web site in Perforce
  - Buying insurance, so to speak.
  - Spreading around the work since the Perforce admin is busy, overworked, or gone
- 




# The tune-up: staff-time needs

- 
- Sometimes, there's just too much work.
    - ❖ Scripting "p4 review" daemons (see last year's talk)
    - ❖ Making bug database integrations work
    - ❖ Making an initial stab at a build script
- 



# Other items in the files...

- “p4 password” (for security)
    - ❖ Passwords could go into P4CONFIG file?
  - Sharing workspaces (CR-LF issues)
    - ❖ See 2001.1 release notes. Most efficient approach is to use “share” in workspace options as necessary.
  - “p4 depots” for remote depots
    - ❖ Comparison to ClearCase MultiSite product.
- 

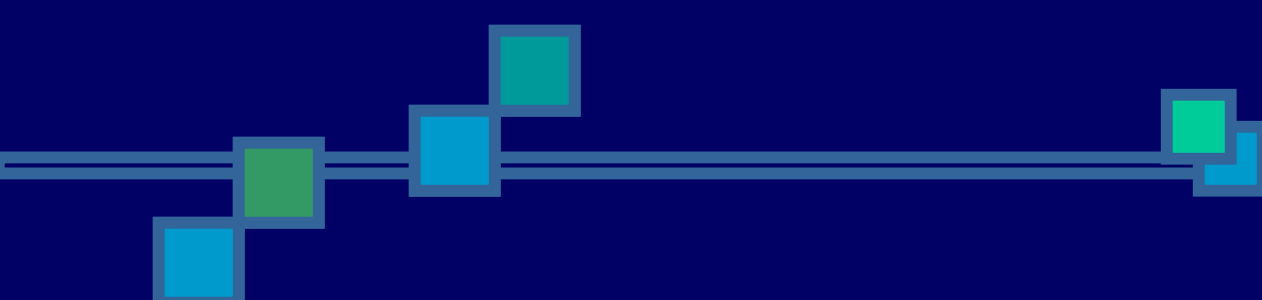


# Perforce depots for the ClearCase “multisite” user...

- A ClearCase Multisite installation gives you, the local user, a snapshot of a remote site’s work in a readonly, local branch.
  - ❖ *Including labels, including metadata.*
- A Perforce remote depot gives you live readonly access, real-time, of a remote server’s data.
  - *But not labels, usernames, metadata.*

*Neither is a seamless development environment: both assume that development on a given file happens at only one site.*





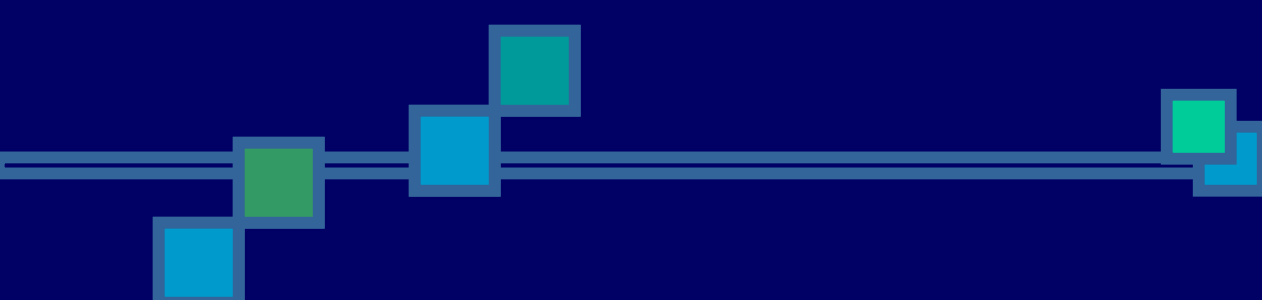
# Perforce remote depots – classic presentation

Atreides:1666

- `//depot/...` - local work
- `//spice/...` - maps to `//depot/outgoing/...` on arrakis:1666

Arrakis:1666

- `//depot/...` - local work
- 



# Perforce remote depots – imitating ClearCase a bit

Atreides:1666

- //depot/... - local work
- //spice/... - maps to //depot/outgoing/... on arrakis:1666, generates calls to arrakis
- //copy-of-spice/... - created every morning at 4 AM, from //spice/...


This is what developers will normally view, and is a local cache of arrakis work marked readonly.

*Perhaps //spice/... is restricted (“p4 protect”) to generate fewer packets to arrakis, also.*





“From the files of a Perforce  
Consultant...”



*An annotated list of surprises, good and bad,  
by Jeff Bowles  
jab@piccoloeng.com  
Piccolo Engineering, Inc.*