

SCRIPTING TECHNIQUES USING PERFORCE

JEFF A. BOWLES

PICCOLO ENGINEERING, INC.

STEVE VANCE

STELLAR ADVANCES

Terminology

- Context
 - “where” something happens.
 - E.g., server-side trigger, client-side build script
- Techniques
 - “how it’s done, tricks to do get it done.”
 - E.g., choice of scripting languages, how to retrieve data, how to optimize loops.
- Purpose
 - “the big picture.”

Terminology (example)

- Context (“where”)
 - “Downstairs at breakfast table, with Sunday Times spread out in front of me.”
- Techniques (“how”)
 - “Pour the milk into the cup first, then the sugar, then add the tea. Stir 32.4 seconds, clockwise, then stop. Drink at leisure.”
- Purpose (“what” or “why”)
 - “Enjoying the Times with a nice bit of tea.”

Terminology

(example: build script)

- Context (“where”)
 - Client workspace
- Techniques (“how”)
 - Self-contained script that...
 - ... retrieves files (as of specific date or label or changelist)...
 - ... into empty workspace.
 - Then builds and regresses, and reports results.
- Purpose (“what” or “why”)
 - Provide nightly “heart-beat” to project via an automated build of current source.

Terminology

(example: trigger scripts)

- Context (“where”)
 - Script invoked by Perforce server...
 - ... to be run whenever “p4 protect” is run.
 - Installed using “p4 triggers”.
- Techniques (“how”)
 - Self-contained...
 - ... Python script... (in this case)
 - ... information / arguments passed to script on command-line, including pathname to “p4 protect” data that is to be validated.
 - ... extreme caution to avoid performance bottlenecks. (In this case, zero calls to ‘p4’ from script.
- Purpose (“what” or “why”)
 - Implement a trigger that refuses any “p4 protect” configuration that grants access to “user *”.

Terminology

(example: post-submit script)

- Context (“where”)
 - Client workspace
- Techniques (“how”)
 - Self-contained script that...
 - ... runs ‘p4 review’ to figure out what’s been submitted since last time script was run...
 - ... then runs ‘p4 reviews’ for each change, to see who to send email to....
 - ... generates email for each submission.
- Purpose (“what” or “why”)
 - Send email every time someone submits a change, to the users who indicate (via ‘p4 user’ form) they want that email.

Terminology

(example: server maintenance)

- Context (“where”)
 - On server machine, invoking ‘p4d’ directly
- Techniques (“how”)
 - Self-contained script that...
 - ... invokes ‘p4d’ to create checkpoint or create journal entry...
 - ... and does not need ‘p4 protect’ access to do so. (Just needs access to ‘p4 root’ area.)
- Purpose (“what” or “why”)
 - Provide regular checkpoint even if server is not running or does not grant ‘super’ permissions to the IT group that maintains the machine.

Terminology

(example: p4win ‘tools’ item)

- Context (“where”)
 - On client machine, invoked from ‘p4win tools’ menu.
- Techniques (“how”)
 - Runs in command-line window, output might be discarded if window’s closed too quickly.
 - Args passed by ‘p4win’ to script, including substitutions (“selected file/directory”, for example.)
- Purpose (“what” or “why”)
 - Extend “p4win” functionality to run ‘outside’ commands such as various administrative commands.

Techniques

- Language
 - “p4” command-line is usable from most languages (Unix shell, Perl, Python, Ruby).
 - “p4” marshal output provides preparsed data for Python and Ruby users.
 - P4Perl and P4Ruby provide strong module/language support.
 - Know more than one language, and exploit language features (with reason).

Techniques

(“Pick a friendly language.”)

```
p4 = P4Marshal.new("p4port" =>
  defaultPort, "p4port" => defaultPort)

#-----
# first call to P4: 'p4 users'
#-----
userHash = {}
p4.run("users").each do |u|
  userName = u['User']
  userHash[userName] = u
end
```

```
#-----
# second call to P4: 'p4 clients'
#-----
puts "List of clients:"
p4.run("clients").each do |c|
  clientName,clientOwner =
    c['client'],c['Owner']
  if userHash[clientOwner] != nil
    puts "#{clientName} OK"
  else
    puts "#{clientName} ERR"
  end
end
```

//guest/jeff_bowles/scripts/oldclients.rb

Techniques (Tagged output)

- Output is pre-parsed.
- “p4 -Ztag” is works well for Perl users.

```
User Administrator  
Update 1098801486  
Access 1111005788  
.FullName Administrator  
Email Administrator@SIQDev1
```

```
User AlexB  
Update 1110447078  
Access 1110471261  
FullName AlexB  
Email AlexB@AlexB
```

- “Marshal” output is straightforward interface for Python/Ruby users.

Techniques (capturing output)

- Simplest example:
 `p4 files //depot/main/...@label1 > contents1.txt`
 `p4 files //depot/main/...@label2 > contents2.txt`
 `diff contents1.txt contents2.txt`
- Commands that bring up editor (“submit”, “label”, “user”)
 `p4 label -o newlabel > xxx.txt`
 # futz with xxx.txt using perl/python/ruby/shell
 `p4 label -i < xxx.txt`
- It’s worth writing a subroutine/method to wrap ‘p4’ commands and capture the output.
- “p4 print” is used, often, in this situation, to retrieve file contents (stdout) without updating workspace files.

Techniques

(loop through results)

- The expensive example:
foreach f in filelist:
Process "f" somehow, invoking 'p4' for each file.
- The less expensive example:
use label name, or changelist number, or temporary client
definition to refer to list of files.
p4 cmd @labelname or p4 cmd @=chgnum
- The rule:
"Loop through results. Don't put a call to 'p4' inside a loop, and write Perforce library routines/methods to avoid such constructs."

Techniques

(loop through results,
use ‘//clientname/...’ when appropriate)

```
#-----
```

```
# first call to P4: 'p4 client -o'
```

```
[cl_spec] = runp4cmd("p4 -G client -o")
```

```
cl_name = cl_spec['Client']
```

```
cl_root = cl_spec['Root']
```

```
#-----
```

```
# second call to P4: 'p4 fstat //myclient/...'
```

```
ret = runp4cmd("p4 -G fstat //%s/..." % cl_name)
```

```
#-----
```

```
# now, we have a list of all Perforce-managed files in this workspace.
```

```
# we can use Python methods for making a list of local files,
```

```
# and compare the lists....
```

//guest/jeff_bowles/scripts/p4unknown.py

Techniques

(use comment convention that documents performance)

```
# Task: determine which client specs have the
# option 'nocompress' set.
#
# status: tested on Win/NT using perl 5.6
# num of calls to 'p4': 1
# room for optimization/improvement: add getopt call
...
#-----
# only call to P4: 'p4 clients'
#-----
$client_tagged_cmd = "p4 -Ztag -u arthur clients";
@ret = readinZtag($client_tagged_cmd);
# ... (process output that is stored in @ret)
```

//guest/jeff_bowles/scripts/findnocompress.pl

Example: bug database triggers

1. The p4DTI project (bidirectional, uses ‘p4 logger’ and ‘p4 jobs’ to propagate data.)
2. “p4 jobs” enhancements to enforce state transitions, via “in” trigger on “job” form.

Example: access control of forms

1. A trigger might refuse an update unless the invoker is a member of a certain group. (*More in Steve's talk.*)
2. A trigger might refuse an update to “protect” form that violates a security policy. (*E.g., disallow ‘user *’ from having any sort of access.*)

The big questions: what, where, how

1. Define the context and the purpose, informed by the techniques and restrictions available.
2. Some tasks can be addressed in multiple ways. (Checkpoints created using *p4 admin* and using '*p4d -jc*'.)

The following talks...

1. JT's talk on a post-processor for using “p4 -G” as the first half of a Unix-friendly reporting/mining tool.
2. Steve's talk gives many “rules of the road” and ideas for using the new *forms* triggers.
3. Reb's talk on one specific strategy for a trigger, using the new functionality to augment basic Perforce storage strategies/policies.