Jenkins Data Mining on the Command Line

Noah Sussman

Etsy

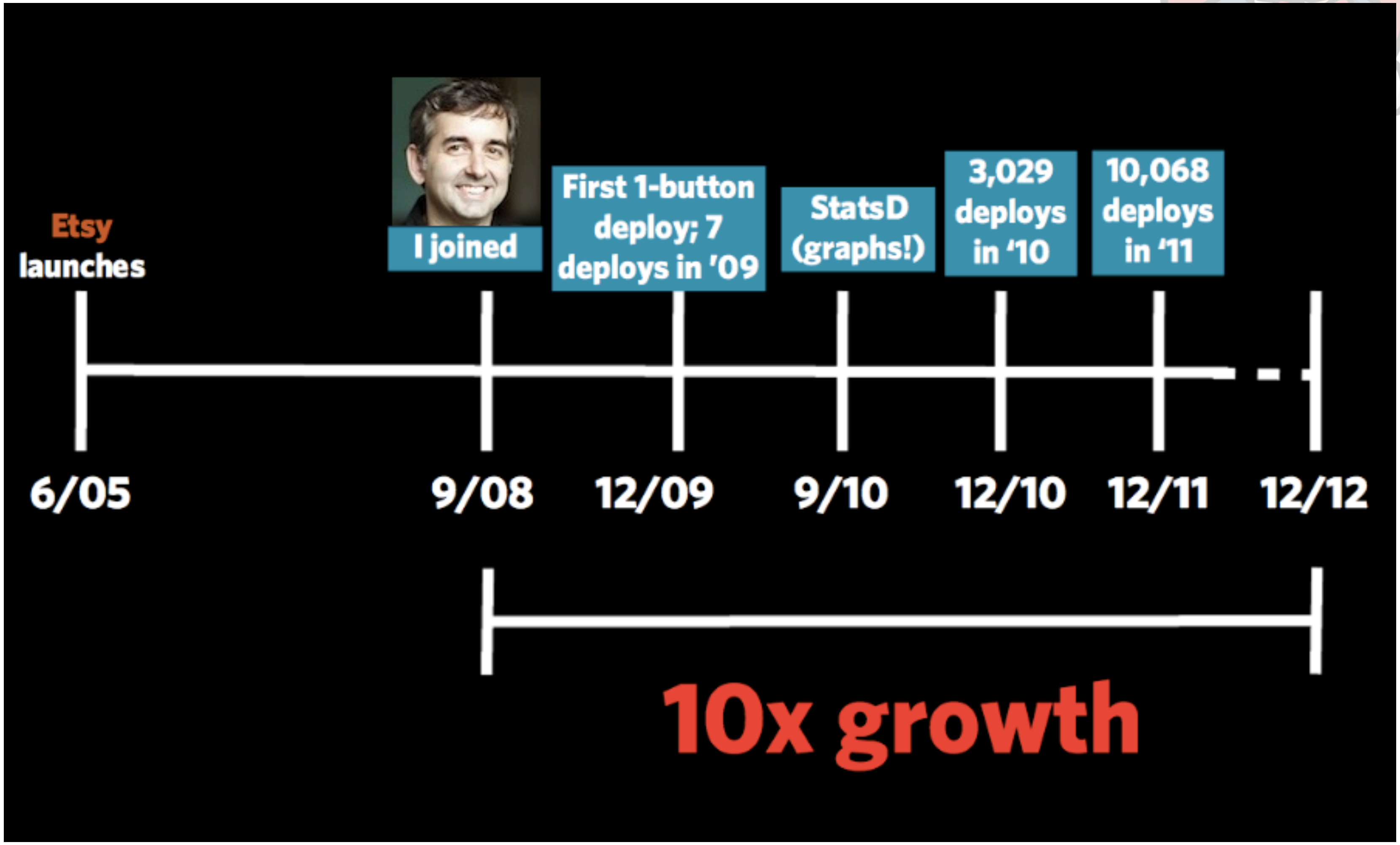
Etsy

Etsy is the marketplace we make together. We enable people anywhere to easily build and directly exchange with independent, creative businesses. Etsy has 15 million members and 875,000 sellers in over 150 countries. In 2011, our sellers grossed more than $525 million in sales.

Etsy

- Total Members: over 15 million
- Total Active Sellers: over 875,000
- Unique Visitors per month: 40 million
- Items Currently Listed: 13 million
- Page Views per month: over 1.4 billion

Chad Dickerson's history of CI at Etsy



CI systems exhibit complex behaviors

- Jenkins plugins provide nice visibility:
- xUnit Plugin
- Warnings Plugin
- Radiator View Plugin
- Extended Email Plugin
- IRC Plugin
- If you're not using these yet – start now!
- But…

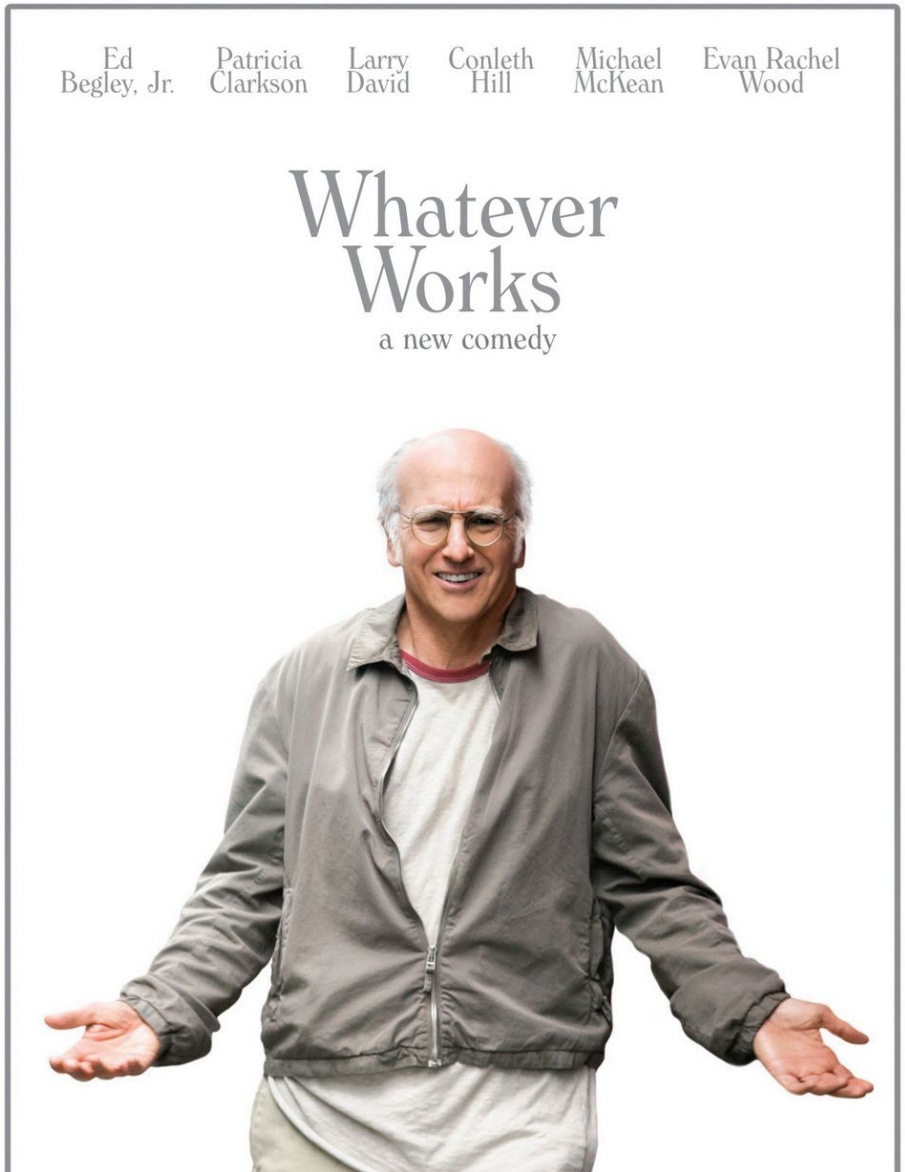
But...

- What is the failure rate of a specific test?
- How many failed builds occur per day?
- What is the status of slave `bob0107`?
- How many executors were busy in the last five minutes?
- How many slaves are currently attached?

Tools

- `find, grep, cut, spark, perl, irb`
- My tools work on Mac OS and Linux.
- If you're on Windows you can use Cygwin.
- Or you could do it all with Perl or Ruby.
- Or with something else.
- *There's more than one way to do it.*

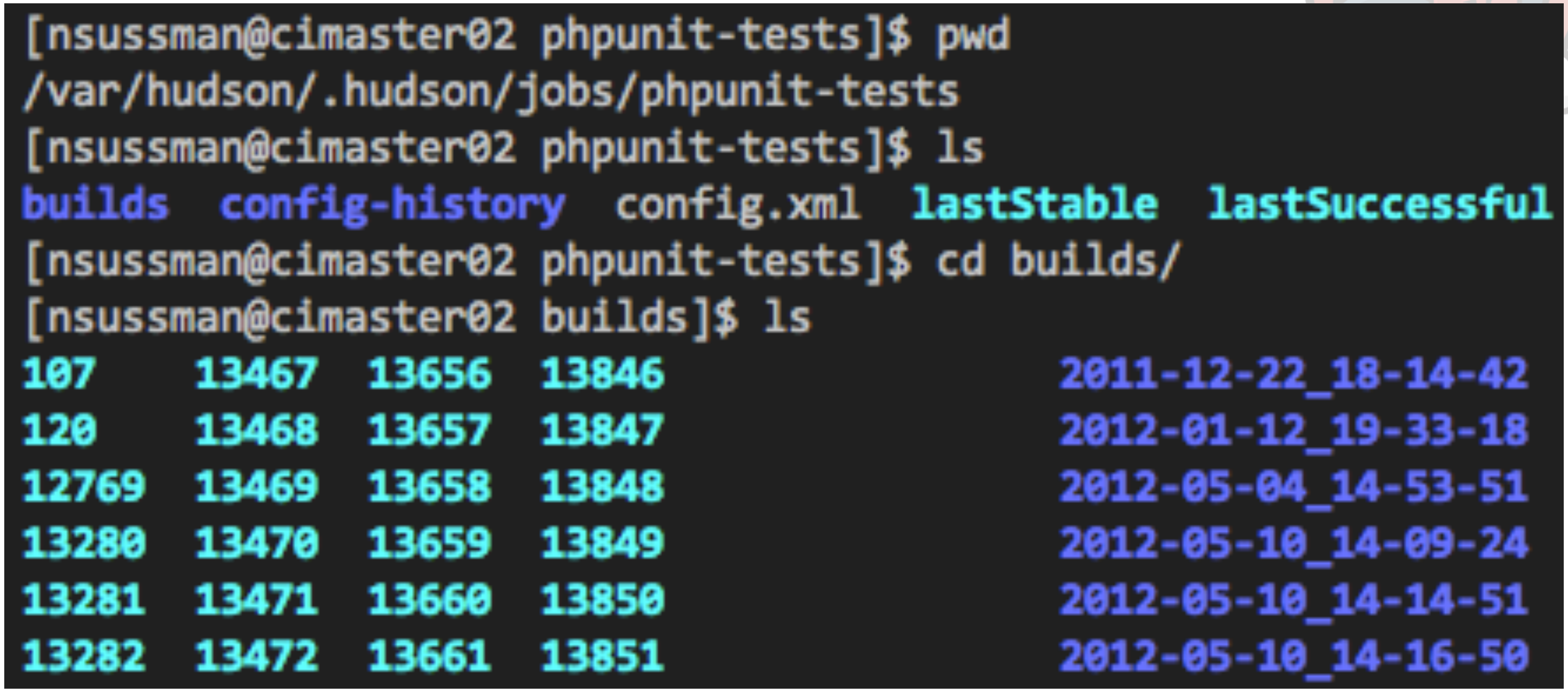
A note about methodology



Whatever works

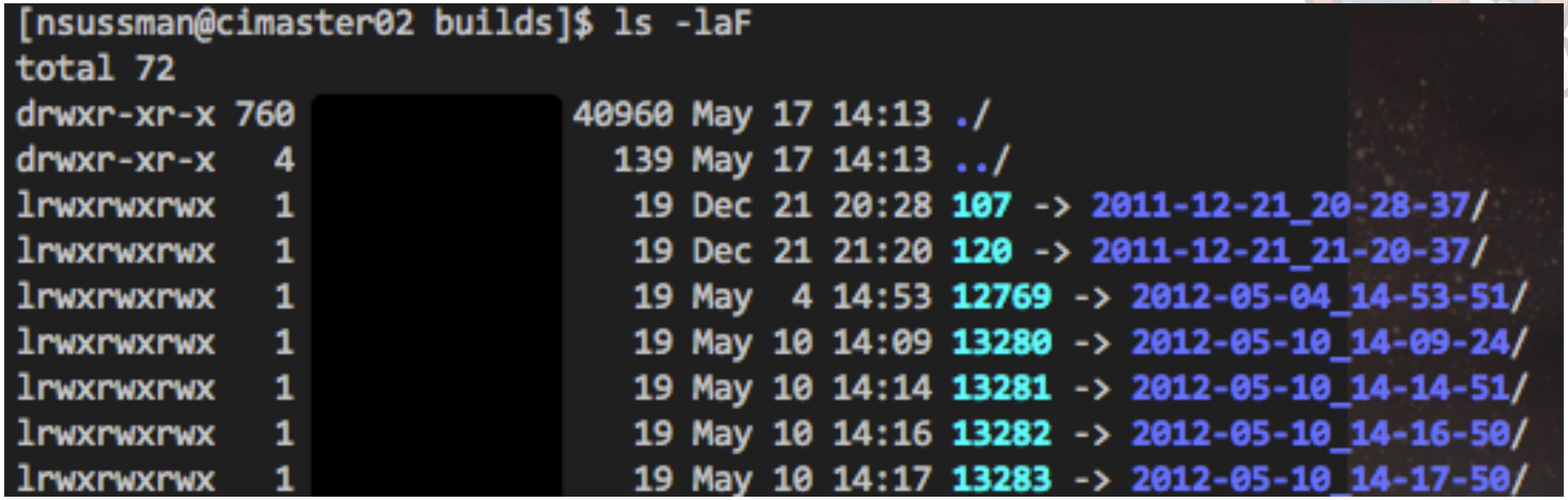
- Simple > Robust
- Simple > Reusable
- Simple > Performant
- Simple > Elegant
- Do the *minimum amount of work* necessary to get the information that you need in order to make a decision.

Directory structure of a Jenkins project



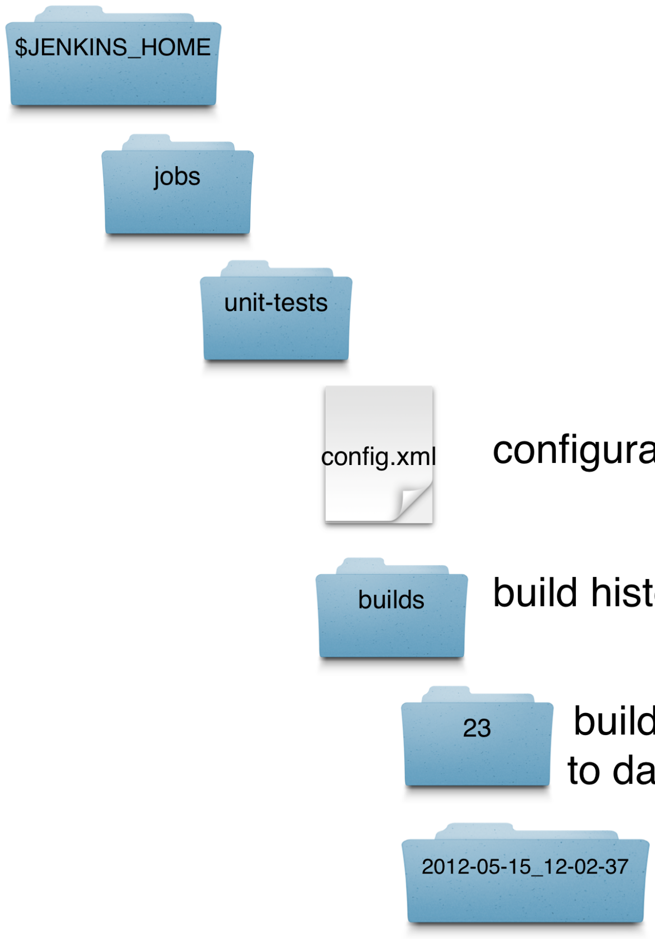
```
[nsussman@cimaster02 phpunit-tests]$ pwd
/var/hudson/.hudson/jobs/phpunit-tests
[nsussman@cimaster02 phpunit-tests]$ ls
builds  config-history  config.xml  lastStable  lastSuccessful
[nsussman@cimaster02 phpunit-tests]$ cd builds/
[nsussman@cimaster02 builds]$ ls
107    13467  13656  13846                  2011-12-22_18-14-42
120    13468  13657  13847                  2012-01-12_19-33-18
12769  13469  13658  13848                  2012-05-04_14-53-51
13280  13470  13659  13849                  2012-05-10_14-09-24
13281  13471  13660  13850                  2012-05-10_14-14-51
13282  13472  13661  13851                  2012-05-10_14-16-50
```

Directory structure of a Jenkins project



```
[nsussman@cimaster02 builds]$ ls -laF
total 72
drwxr-xr-x 760 [redacted] 40960 May 17 14:13 ./
drwxr-xr-x   4              139 May 17 14:13 ../
lrwxrwxrwx   1               19 Dec 21 20:28 107 -> 2011-12-21_20-28-37/
lrwxrwxrwx   1               19 Dec 21 21:20 120 -> 2011-12-21_21-20-37/
lrwxrwxrwx   1               19 May  4 14:53 12769 -> 2012-05-04_14-53-51/
lrwxrwxrwx   1               19 May 10 14:09 13280 -> 2012-05-10_14-09-24/
lrwxrwxrwx   1               19 May 10 14:14 13281 -> 2012-05-10_14-14-51/
lrwxrwxrwx   1               19 May 10 14:16 13282 -> 2012-05-10_14-16-50/
lrwxrwxrwx   1               19 May 10 14:17 13283 -> 2012-05-10_14-17-50/
```

Directory structure of a Jenkins project

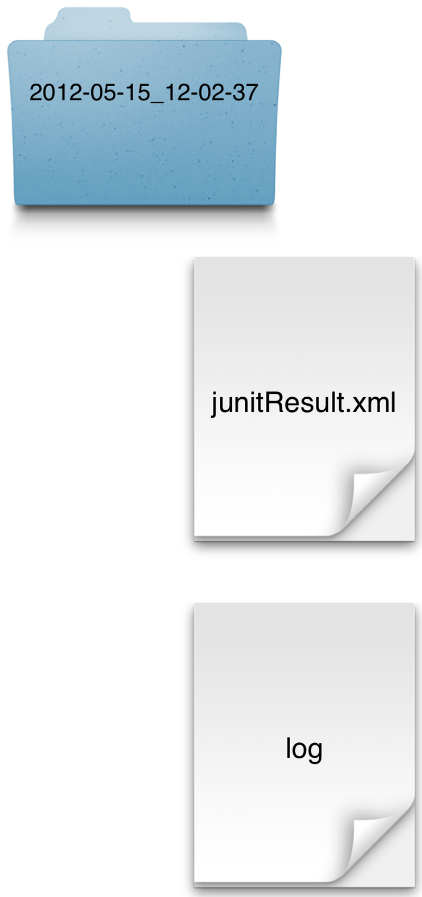


configuration file for this job

build history saved as flat files

build numbers are symlinks to dated directories

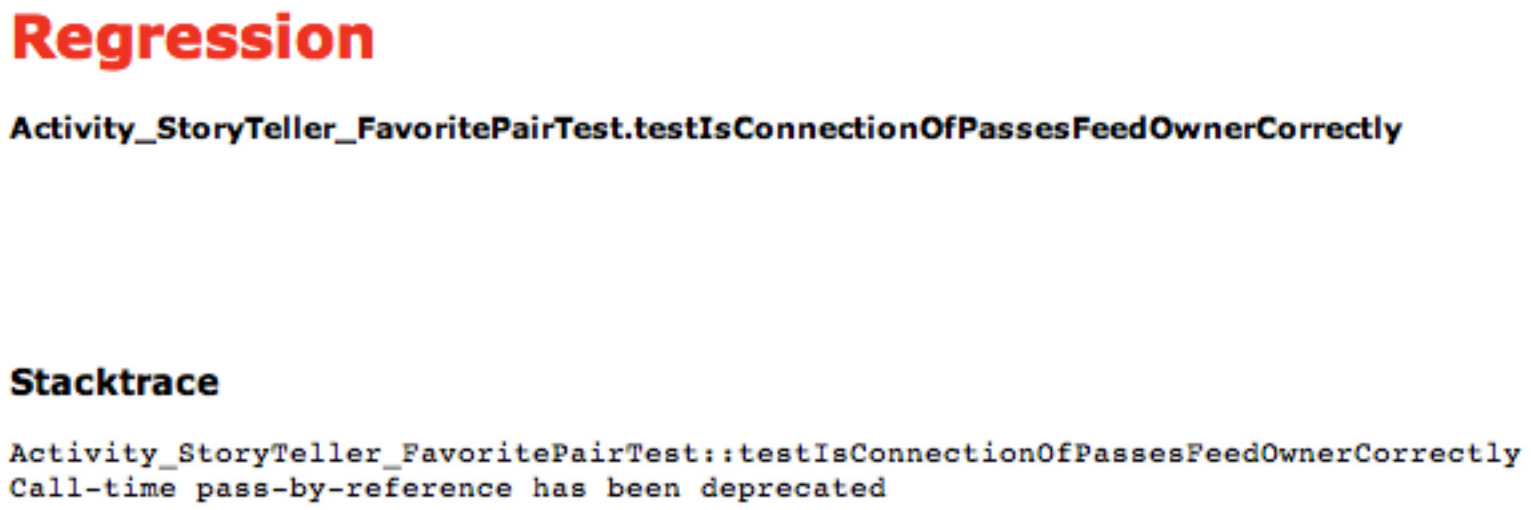
Directory structure of a build



Determining how often a test fails

- You *can* get this information through the xUnit plugin.
- But you have to click through a lot of screens.
- Doing it on the command line is faster.

Determining how often a test fails



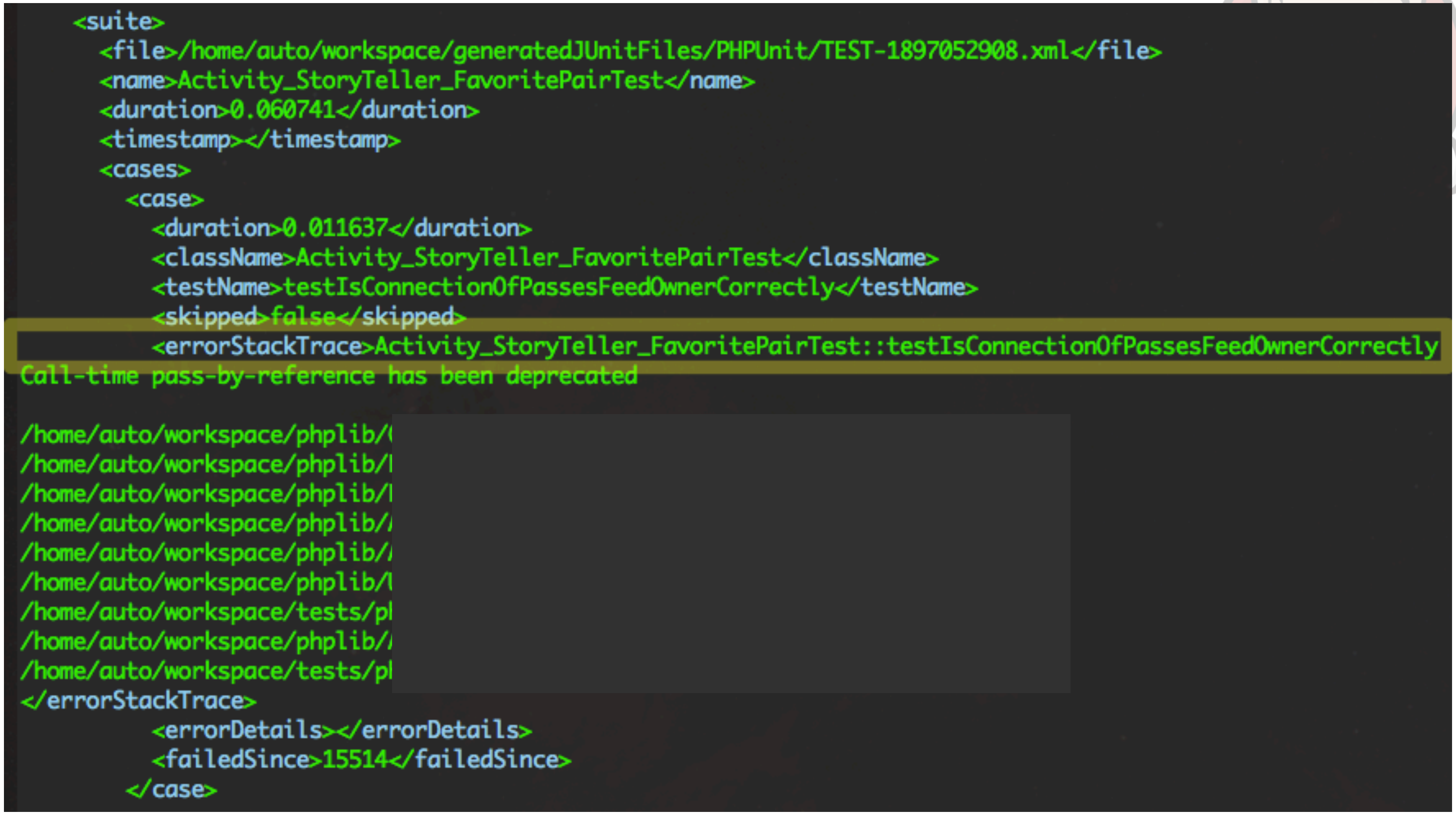
Regression

Activity_StoryTeller_FavoritePairTest.testIsConnectionOfPassesFeedOwnerCorrectly

Stacktrace

```
Activity_StoryTeller_FavoritePairTest::testIsConnectionOfPassesFeedOwnerCorrectly
Call-time pass-by-reference has been deprecated
```

junitResult.xml

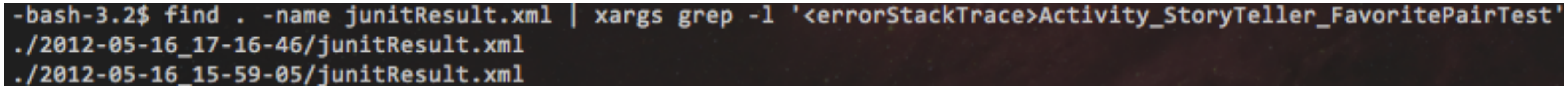


```
    <suite>
      <file>/home/auto/workspace/generatedJUnitFiles/PHPUnit/TEST-1897052908.xml</file>
      <name>Activity_StoryTeller_FavoritePairTest</name>
      <duration>0.060741</duration>
      <timestamp></timestamp>
      <cases>
        <case>
          <duration>0.011637</duration>
          <className>Activity_StoryTeller_FavoritePairTest</className>
          <testName>testIsConnectionOfPassesFeedOwnerCorrectly</testName>
          <skipped>false</skipped>
          <errorStackTrace>Activity_StoryTeller_FavoritePairTest::testIsConnectionOfPassesFeedOwnerCorrectly
Call-time pass-by-reference has been deprecated

/home/auto/workspace/phplib/[redacted]
/home/auto/workspace/phplib/
/home/auto/workspace/phplib/
/home/auto/workspace/phplib/
/home/auto/workspace/phplib/
/home/auto/workspace/phplib/
/home/auto/workspace/tests/pl
/home/auto/workspace/phplib/
/home/auto/workspace/tests/pl
</errorStackTrace>
          <errorDetails></errorDetails>
          <failedSince>15514</failedSince>
        </case>
```

Find all occurrences of that stack trace

```
$ cd $HUDSON_HOME/jobs/unit-tests/builds
$ find . -name junitResult.xml | \
   xargs grep ‘<errorStackTrace>Activity_StoryTeller_FavoritePairTest’
```



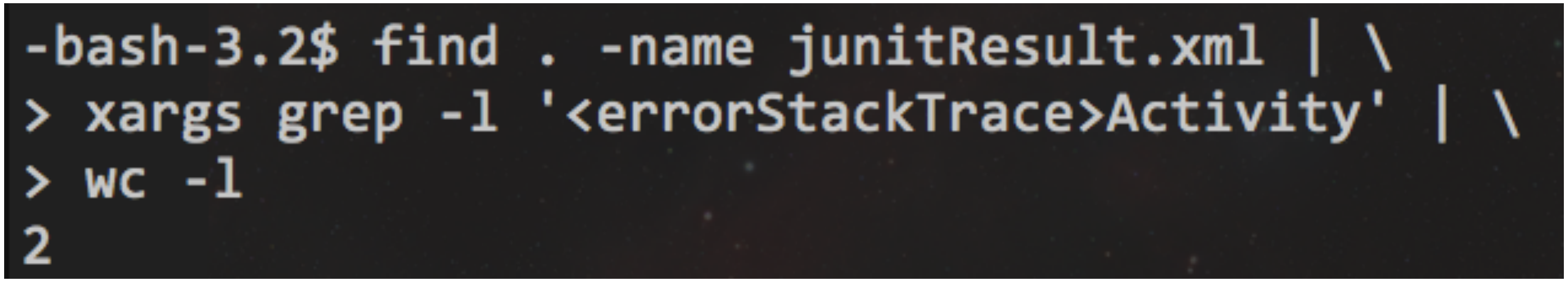
```
-bash-3.2$ find . -name junitResult.xml | xargs grep -l '<errorStackTrace>Activity_StoryTeller_FavoritePairTest'
./2012-05-16_17-16-46/junitResult.xml
./2012-05-16_15-59-05/junitResult.xml
```

Breakdown

- `find . -name junitResult.xml` *recursively finds all the JUnit result files in the current directory and its subdirectories.*
- `xargs grep -l <string>` *filters for just the files that contain the string we are looking for – in this case the first line of the stack trace.*

Counting the number of failures

```
find . -name junitResult.xml | \
  xargs grep '<errorStackTrace>' | \
  wc -l
```



```
-bash-3.2$ find . -name junitResult.xml | \
> xargs grep -l '<errorStackTrace>Activity' | \
> wc -l
2
```

Sorta mappy, sorta reduce-y

- Use **find** and **grep** to find interesting builds.
- Use **wc** to return a count.
- It's not fancy, but hey *whatever works.*

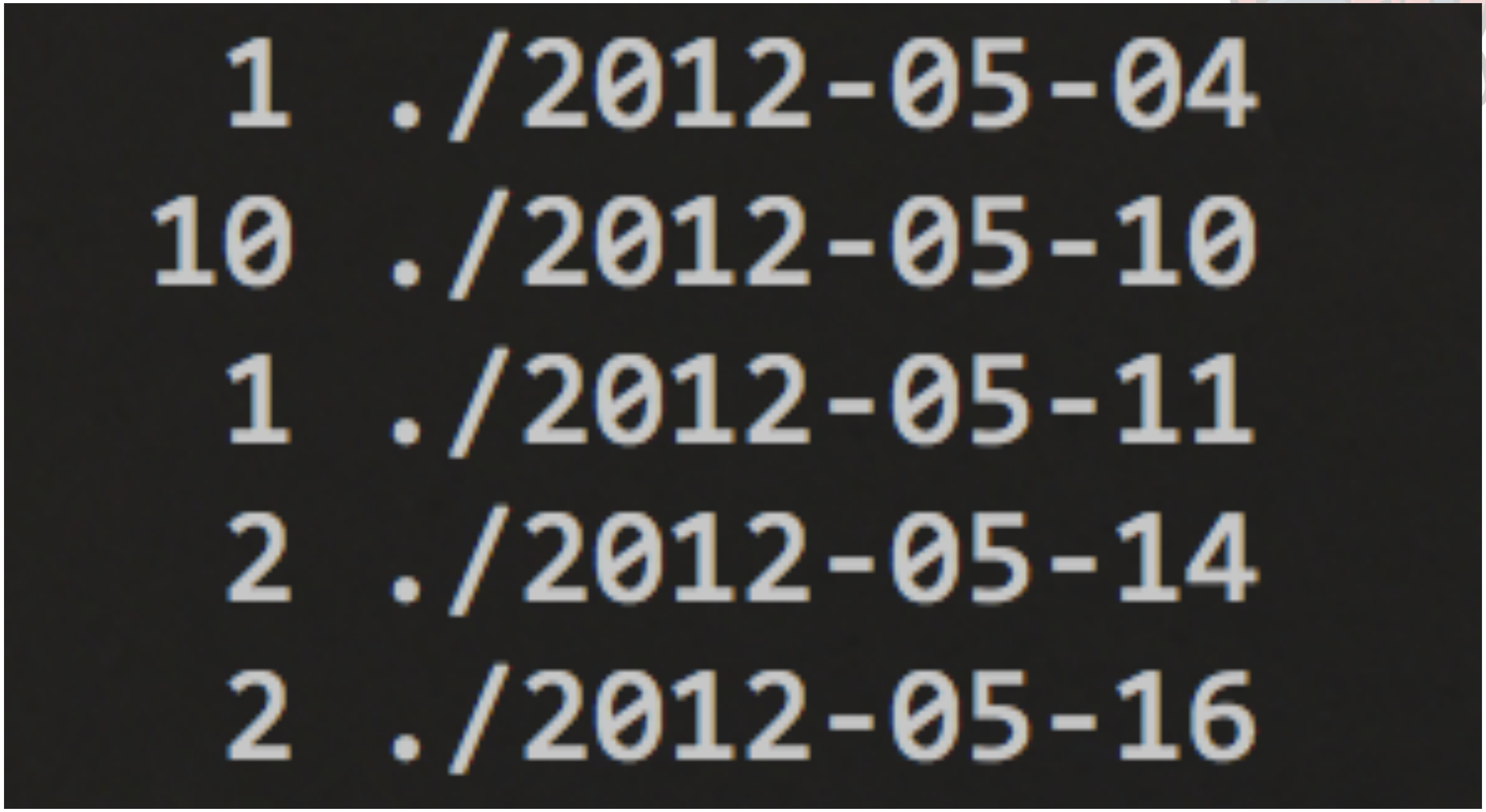
Bucketing failed builds by date

- How many builds fail each day?
- You can figure this out from the build trend graph, but again it's *much* faster to do it on the command line.

How many failed builds per day?

```
$ cd unit-tests/builds
$ find . -name log | \
   xargs grep -l FAILURE | \
   cut -d'_' -f1 | \
   sort | \
   uniq -c
```

Here's the output

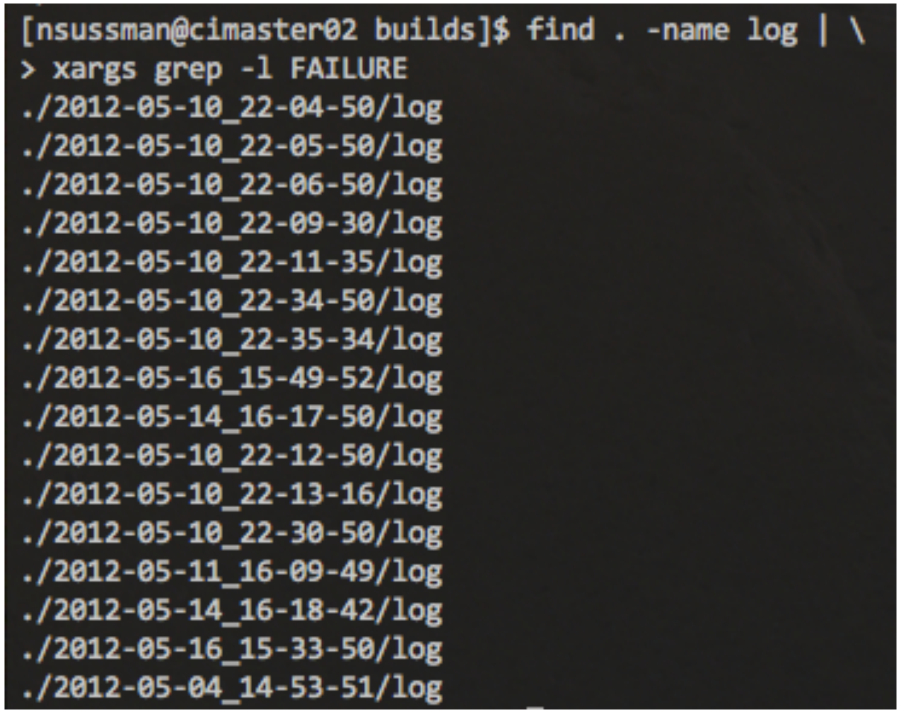


```
 1 ./2012-05-04
10 ./2012-05-10
 1 ./2012-05-11
 2 ./2012-05-14
 2 ./2012-05-16
```

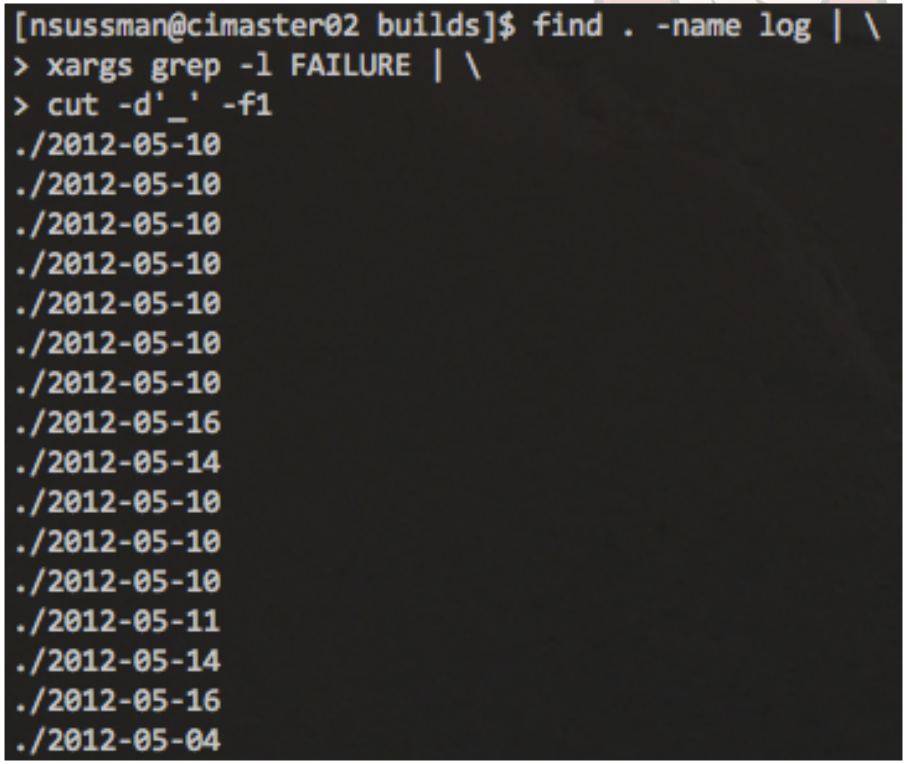
Breakdown

- `find . -name log` *finds console logs*
- `xargs grep -l FAILURE` *filters for console logs from builds that failed*
- `cut -d'_' -f1` *returns just the* ***day*** *the build was performed, ignoring the rest of the time stamp.*
- `sort|uniq –c` *returns a table showing how many failed builds occurred on each day.*

Digression: cut -d'_' -f1

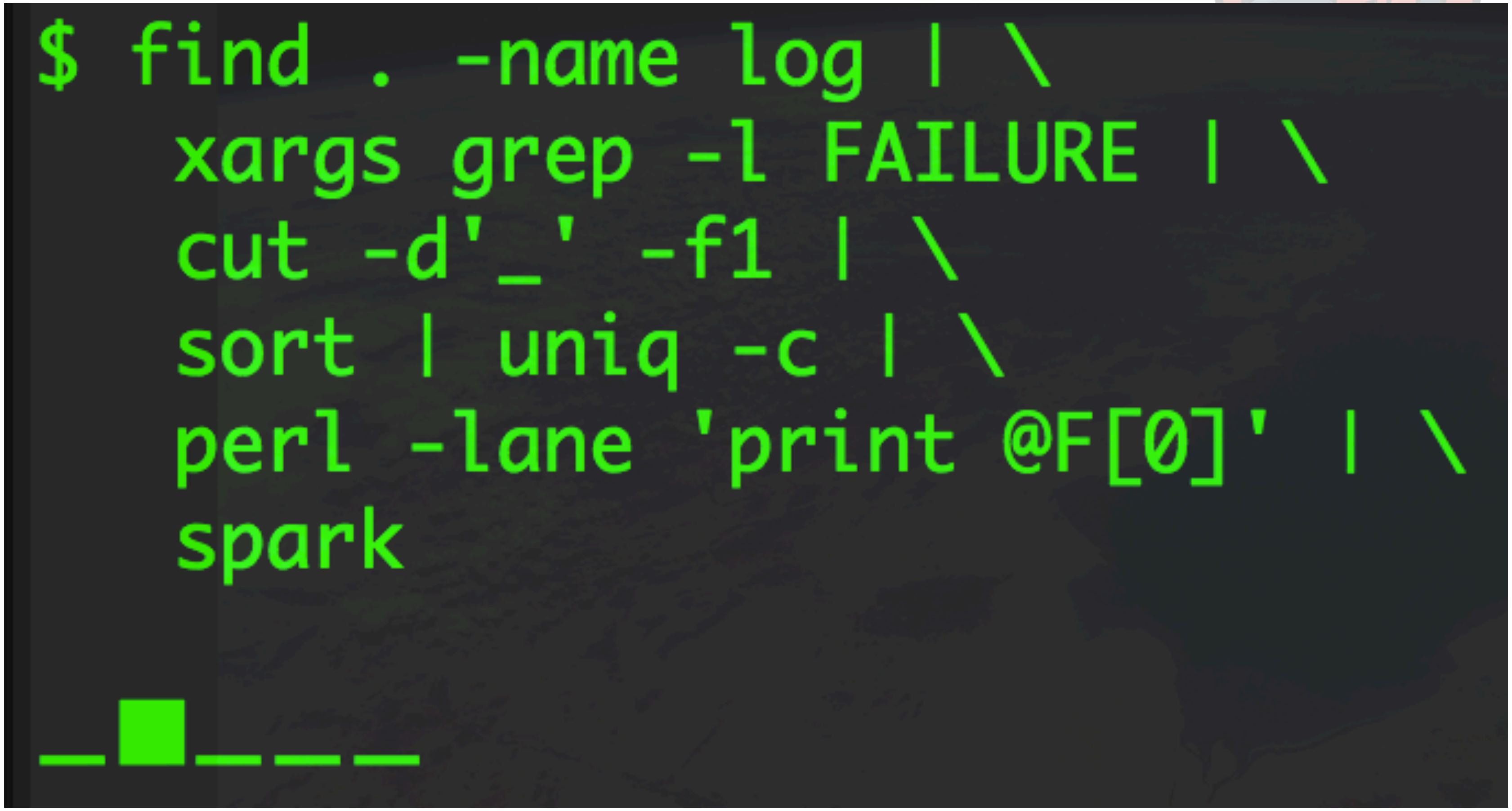


```
[nsussman@cimaster02 builds]$ find . -name log | \
> xargs grep -l FAILURE
./2012-05-10_22-04-50/log
./2012-05-10_22-05-50/log
./2012-05-10_22-06-50/log
./2012-05-10_22-09-30/log
./2012-05-10_22-11-35/log
./2012-05-10_22-34-50/log
./2012-05-10_22-35-34/log
./2012-05-16_15-49-52/log
./2012-05-14_16-17-50/log
./2012-05-10_22-12-50/log
./2012-05-10_22-13-16/log
./2012-05-10_22-30-50/log
./2012-05-11_16-09-49/log
./2012-05-14_16-18-42/log
./2012-05-16_15-33-50/log
./2012-05-04_14-53-51/log
```



```
[nsussman@cimaster02 builds]$ find . -name log | \
> xargs grep -l FAILURE | \
> cut -d'_' -f1
./2012-05-10
./2012-05-10
./2012-05-10
./2012-05-10
./2012-05-10
./2012-05-10
./2012-05-10
./2012-05-16
./2012-05-14
./2012-05-10
./2012-05-10
./2012-05-10
./2012-05-11
./2012-05-14
./2012-05-16
./2012-05-04
```

Quick 'n' dirty graphs with spark



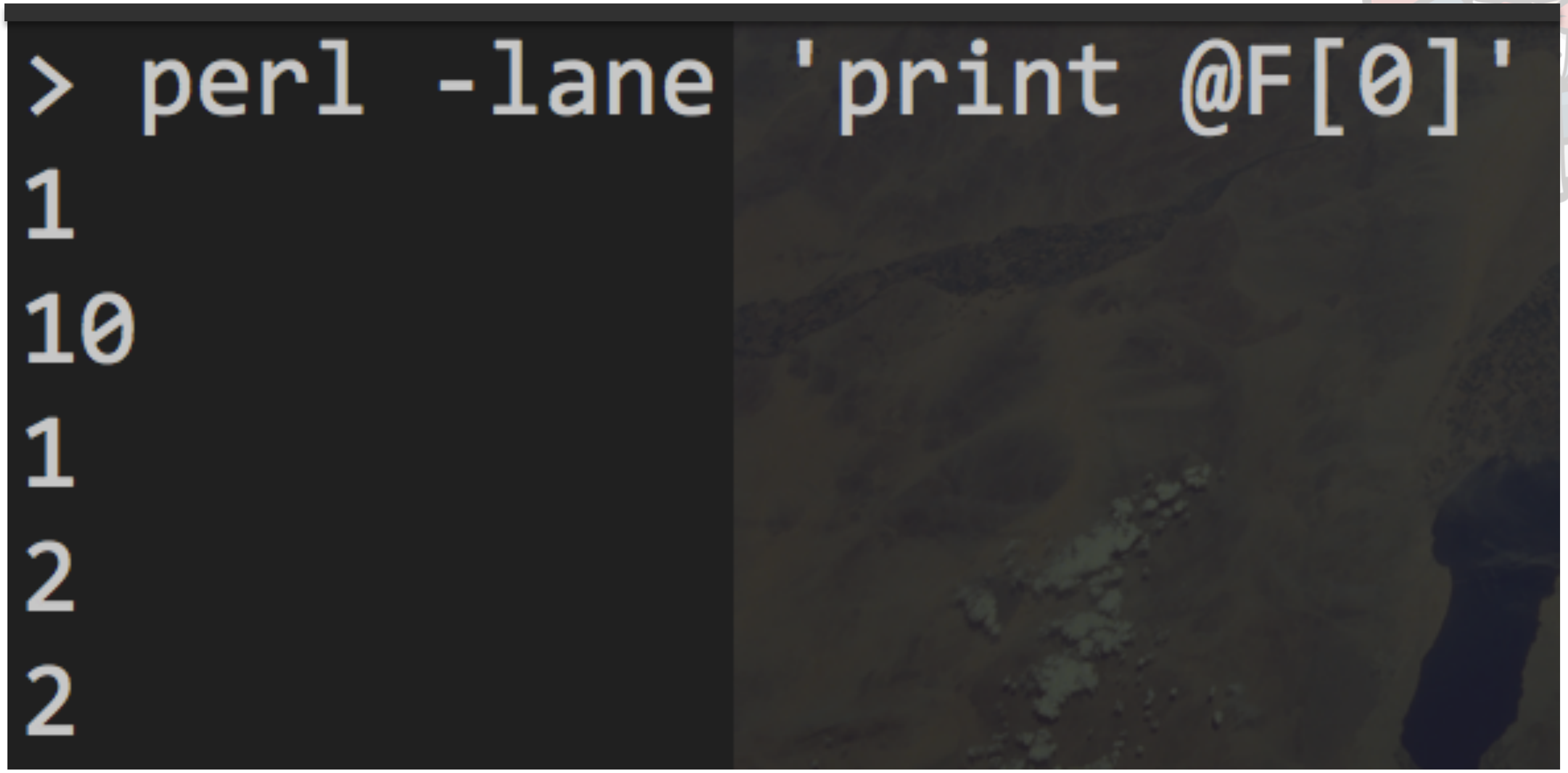
```
$ find . -name log | \
   xargs grep -l FAILURE | \
   cut -d'_' -f1 | \
   sort | uniq -c | \
   perl -lane 'print @F[0]' | \
   spark

_█___
```

Quick 'n' dirty graphs with spark

- `spark` *takes a whitespace-delimited list of numbers as its input and produces a graph using Unicode block characters.*
- `perl -lane 'print @F[0]'` *filters out everything but the counts of builds that failed per day, resulting in list of integers that can be consumed by* `spark`

Output from `perl -lane`



```
> perl -lane 'print @F[0]'
1
10
1
2
2
```

spark is simple, fun and useful

https://github.com/holman/spark

The Jenkins JSON API

- To read the documentation, go to http://ci.example.com/api
- You can append /api/json to the end of nearly any Jenkins URL to get JSON data.
- http://ci.example.com/api/json *for latest builds*
- http://ci.example.com/job/unit-tests/api/json *for history of a specific build.*
- http://ci.example.com/computer/api/json *for slave information.*

Using depth= to get more granular data

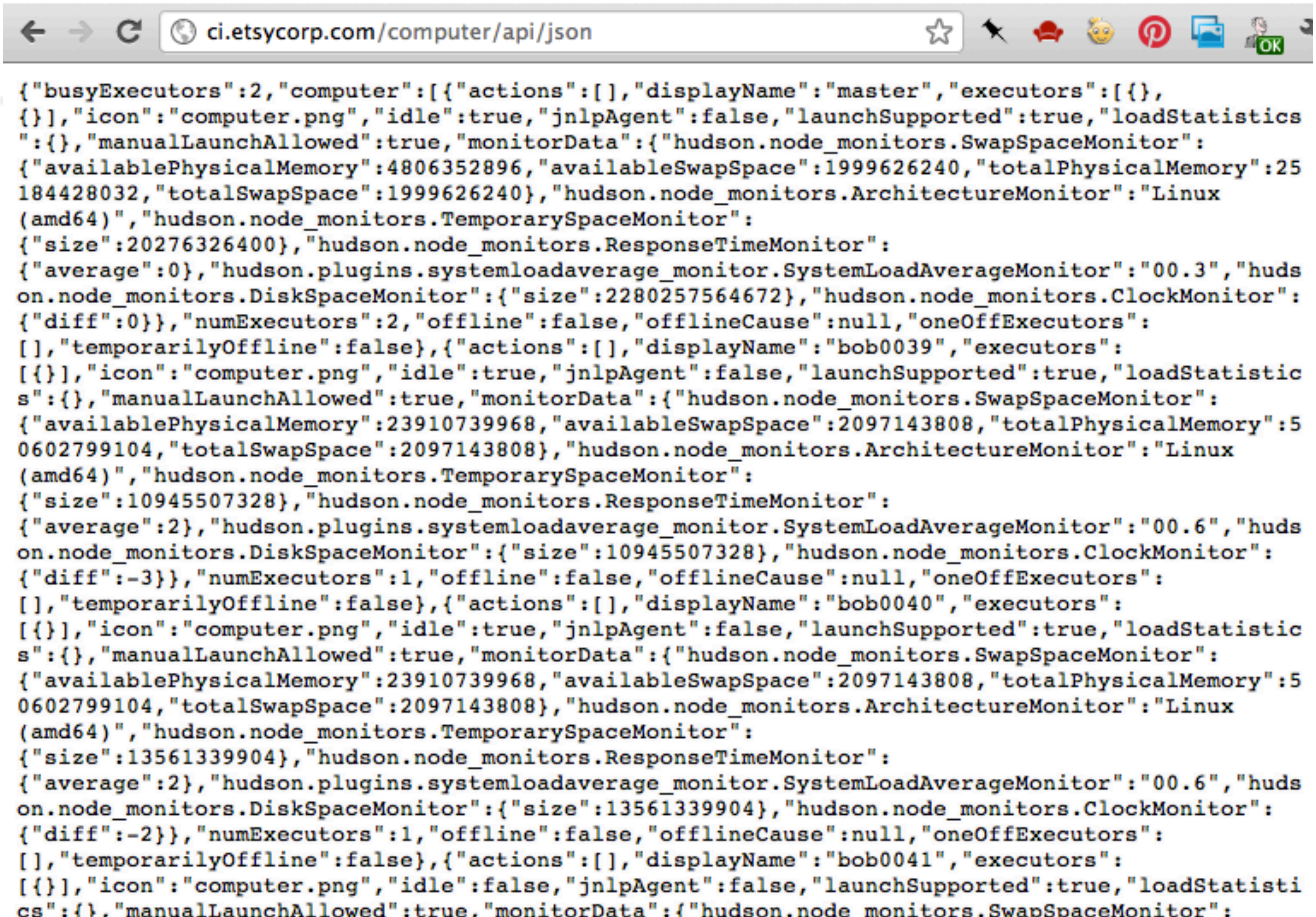
- If the API response doesn't contain some data that you expected, try appending ?depth=1 to the URL.
- If you still don't get what you want, increase the integer value.
- Usually you'll keep getting more data up until around ?depth=5
- Exactly what and how much data you'll get is dependent on the configuration of your Jenkins instance.

Drawbacks of using depth=

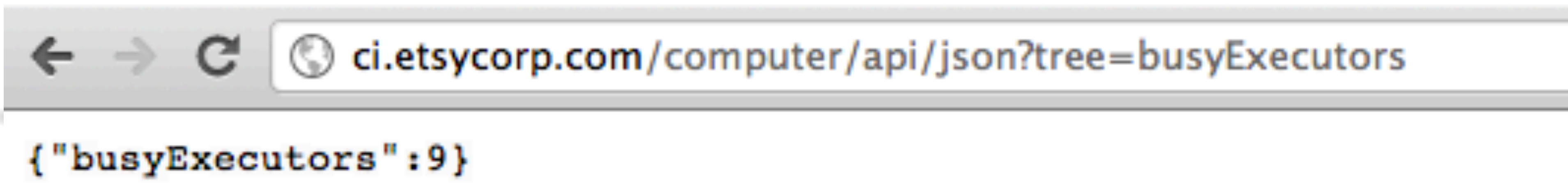
- Depending on how deep you go into the API response, you can wind up with a **lot** of data.
- Such large responses can be expensive to download.
- In some cases you can request a response so large that you will wind up DDoSing Jenkins!

Using `tree=` to filter the API response

- The `tree=` URL parameter is like a SQL query.
- Use `depth=` to look at the wealth of information available.
- Then use `tree=` to select only the information you actually need.
- This can dramatically reduce the size of your API responses.

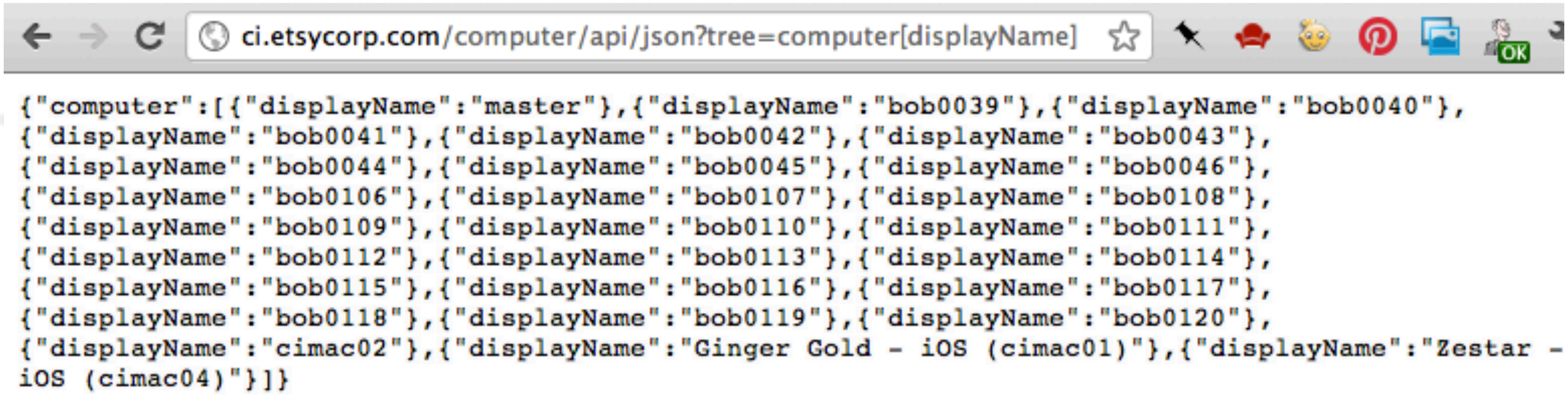

ci.etsycorp.com/computer/api/json

```
{"busyExecutors":2,"computer":[{"actions":[],"displayName":"master","executors":[{},
{}],"icon":"computer.png","idle":true,"jnlpAgent":false,"launchSupported":true,"loadStatistics
":{},"manualLaunchAllowed":true,"monitorData":{"hudson.node_monitors.SwapSpaceMonitor":
{"availablePhysicalMemory":4806352896,"availableSwapSpace":1999626240,"totalPhysicalMemory":25
184428032,"totalSwapSpace":1999626240},"hudson.node_monitors.ArchitectureMonitor":"Linux
(amd64)","hudson.node_monitors.TemporarySpaceMonitor":
{"size":20276326400},"hudson.node_monitors.ResponseTimeMonitor":
{"average":0},"hudson.plugins.systemloadaverage_monitor.SystemLoadAverageMonitor":"00.3","huds
on.node_monitors.DiskSpaceMonitor":{"size":2280257564672},"hudson.node_monitors.ClockMonitor":
{"diff":0}},"numExecutors":2,"offline":false,"offlineCause":null,"oneOffExecutors":
[],"temporarilyOffline":false},{"actions":[],"displayName":"bob0039","executors":
[{}],"icon":"computer.png","idle":true,"jnlpAgent":false,"launchSupported":true,"loadStatistic
s":{},"manualLaunchAllowed":true,"monitorData":{"hudson.node_monitors.SwapSpaceMonitor":
{"availablePhysicalMemory":23910739968,"availableSwapSpace":2097143808,"totalPhysicalMemory":5
0602799104,"totalSwapSpace":2097143808},"hudson.node_monitors.ArchitectureMonitor":"Linux
(amd64)","hudson.node_monitors.TemporarySpaceMonitor":
{"size":10945507328},"hudson.node_monitors.ResponseTimeMonitor":
{"average":2},"hudson.plugins.systemloadaverage_monitor.SystemLoadAverageMonitor":"00.6","huds
on.node_monitors.DiskSpaceMonitor":{"size":10945507328},"hudson.node_monitors.ClockMonitor":
{"diff":-3}},"numExecutors":1,"offline":false,"offlineCause":null,"oneOffExecutors":
[],"temporarilyOffline":false},{"actions":[],"displayName":"bob0040","executors":
[{}],"icon":"computer.png","idle":true,"jnlpAgent":false,"launchSupported":true,"loadStatistic
s":{},"manualLaunchAllowed":true,"monitorData":{"hudson.node_monitors.SwapSpaceMonitor":
{"availablePhysicalMemory":23910739968,"availableSwapSpace":2097143808,"totalPhysicalMemory":5
0602799104,"totalSwapSpace":2097143808},"hudson.node_monitors.ArchitectureMonitor":"Linux
(amd64)","hudson.node_monitors.TemporarySpaceMonitor":
{"size":13561339904},"hudson.node_monitors.ResponseTimeMonitor":
{"average":2},"hudson.plugins.systemloadaverage_monitor.SystemLoadAverageMonitor":"00.6","huds
on.node_monitors.DiskSpaceMonitor":{"size":13561339904},"hudson.node_monitors.ClockMonitor":
{"diff":-2}},"numExecutors":1,"offline":false,"offlineCause":null,"oneOffExecutors":
[],"temporarilyOffline":false},{"actions":[],"displayName":"bob0041","executors":
[{}],"icon":"computer.png","idle":false,"jnlpAgent":false,"launchSupported":true,"loadStatisti
cs":{},"manualLaunchAllowed":true,"monitorData":{"hudson.node_monitors.SwapSpaceMonitor":
```


ci.etsycorp.com/computer/api/json?tree=busyExecutors

```
{"busyExecutors":9}
```

```
?tree=busyExecutors
```


ci.etsycorp.com/computer/api/json?tree=computer[displayName]

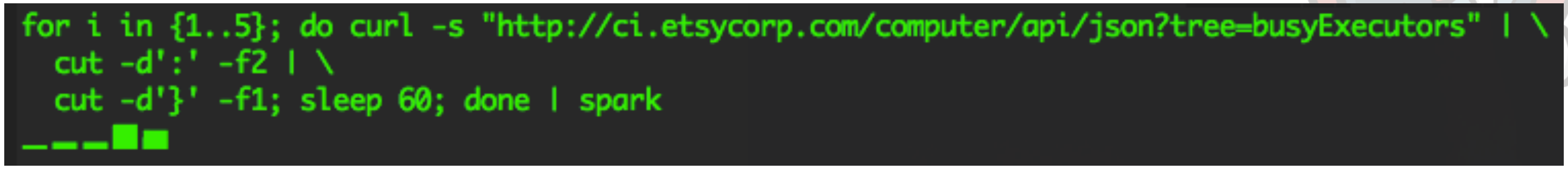
{"computer":[{"displayName":"master"},{"displayName":"bob0039"},{"displayName":"bob0040"},
{"displayName":"bob0041"},{"displayName":"bob0042"},{"displayName":"bob0043"},
{"displayName":"bob0044"},{"displayName":"bob0045"},{"displayName":"bob0046"},
{"displayName":"bob0106"},{"displayName":"bob0107"},{"displayName":"bob0108"},
{"displayName":"bob0109"},{"displayName":"bob0110"},{"displayName":"bob0111"},
{"displayName":"bob0112"},{"displayName":"bob0113"},{"displayName":"bob0114"},
{"displayName":"bob0115"},{"displayName":"bob0116"},{"displayName":"bob0117"},
{"displayName":"bob0118"},{"displayName":"bob0119"},{"displayName":"bob0120"},
{"displayName":"cimac02"},{"displayName":"Ginger Gold - iOS (cimac01)"},{"displayName":"Zestar -
iOS (cimac04)"}]}

```
?tree=computer[displayName]
```

Is the slave named bob0107 attached?

```
curl -s "http://ci.etsycorp.com/computer/api/json?tree=computer\[displayName\]" | \
  grep bob0107
```

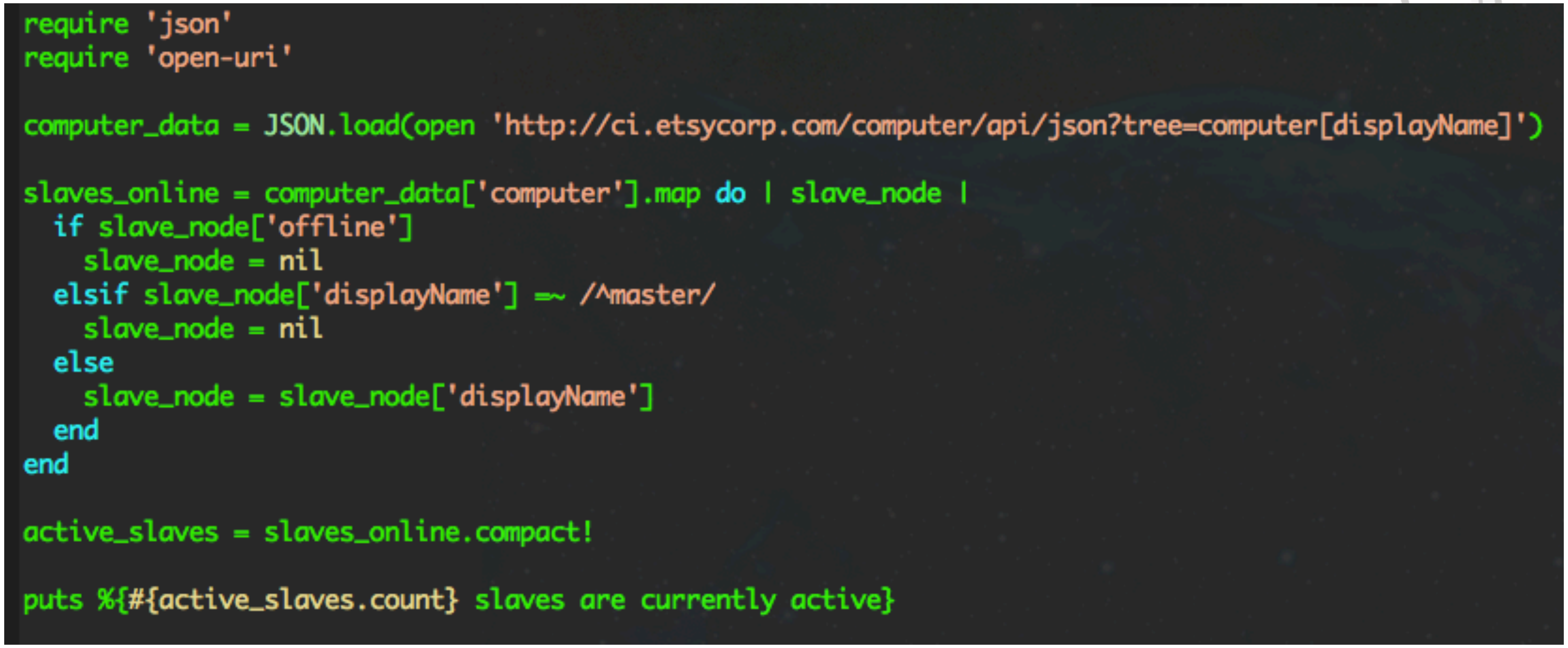
How many executors were busy in the last five minutes?



```
for i in {1..5}; do curl -s "http://ci.etsycorp.com/computer/api/json?tree=busyExecutors" | \
  cut -d':' -f2 | \
  cut -d'}' -f1; sleep 60; done | spark
▁▂▂█▅
```

Use the IRB for deeper analysis

- How many slaves are currently active?



```ruby
require 'json'
require 'open-uri'

computer_data = JSON.load(open 'http://ci.etsycorp.com/computer/api/json?tree=computer[displayName]')

slaves_online = computer_data['computer'].map do | slave_node |
  if slave_node['offline']
    slave_node = nil
  elsif slave_node['displayName'] =~ /^master/
    slave_node = nil
  else
    slave_node = slave_node['displayName']
  end
end

active_slaves = slaves_online.compact!

puts %{#{active_slaves.count} slaves are currently active}
```

My god, it's full of data.

- Use **find** & **grep** to explore Jenkins' flat files.
- If you can't find it in a flat file, then you can almost certainly find it in the JSON API.
- Use the **depth=** and **tree=** URL parameters to explore and filter the JSON API.
- Use the **IRB** and **Nokogiri** when you need to do a deep dive into the JSON API.
- Keep it simple.

Going Deeper

- Use the Post-Build Task plugin to run a shell script and use Netcat to send the resulting data to Graphite or another RRD tool.
- Drop your shell script into a new Jenkins job and use the Plot Plugin to build ad hoc graphs of the results over time.
- These examples all produced TSV output. Redirect that to a file and you can graph it with GNUPlot or Google SpreadSheets.

Questions?

@NoahSussman

Thank You To Our Sponsors

Platinum Sponsor	CloudBees
Gold Sponsors	JFrog, CLOUDANT
Silver Sponsors	LIFERAY, SendGrid Email Delivery. Simplified., CLOUDSMITH™
Bronze Sponsors	WAN DISCO