

Oracle Solaris 11 Cheat Sheet DTrace

What is DTrace?

Oracle Solaris DTrace is a comprehensive, advanced tracing tool for troubleshooting systematic problems in real time. Administrators, integrators and developers can use DTrace to dynamically and safely observe live production systems for performance issues, including both applications and the operating system itself. DTrace allows you to explore your system to understand how it works, track down problems across many layers of software, and locate the cause of any aberrant behavior. Whether it's at a high level global overview like memory consumption or CPU time, to much finer grained information like what specific function calls are being made, DTrace gives the operational insights that have long been missing in the data center.

Understanding DTrace providers and probes

Oracle Solaris 11 is littered with many different points of instrumentation – places of interest to which you can bind actions so you can understand what is going on in your system at any point in time. These 'probes', or programmatical sensors, are at the heart of DTrace and as they fire, data can be gathered and reported back to the user. DTrace probes are delivered through a series of kernel modules called 'providers'.

Common DTrace Providers	Description
dtrace	Start, end and error probes
syscall	Entry and return probes for all system calls
fbt	Entry and return probes for all kernel calls
profile	Timer driven probes
proc	Process creation and lifecycle probes
pid	Entry and return probes for all user-level processes
io	Probes for all I/O related events
sdt/usdt	Developer defined probes at arbitrary locations/names within source code for kernel and user-level processes
sched	Probes for scheduling related events
lockstat	Probes for locking behavior within the operating system

DTrace Command Components

A typical DTrace command has several components:

- A 4-tuple identifier `provider:module:function:name`, where `module` is a kernel module or application library, and `function` and `name` are the routines that are to be instrumented. If any of these are left blank, it is equivalent to a wildcard match. For example, to fire all `entry` routines in the `syscall` provider we would use the following:
`syscall:::entry`
- A predicate, or relational expression, that determines whether any action should be taken. For example, to check whether the process name matches `bash` we would use the following:
`/execname == "bash"/`
- An action for what should happen if the probe fires and the predicate is satisfied. For example, we may create an array and count the number of times a function call has been made we would use the following:
`{ @array[probefunc] = count(); }`

D Scripting Language

As DTrace command line examples become more complex, it may be necessary to construct them using the D scripting language – an awk like script that can be run using the `dtrace -s` command. D scripts can consist of multiple clauses that usually specify one or more probe descriptions, and their associate predicates and actions.

```
#!/usr/sbin/dtrace -s

probe-description
/predicate/
{
action;
}
```

Did you know?

You can find out more information about the Oracle Solaris DTrace, including product documentation, how to guides, and other resources on Oracle Technology Network:

<http://www.oracle.com/technetwork/server-storage/solaris11/technologies/dtrace-1930301.html>

Oracle Solaris 11 Administrator's Cheat Sheet for DTrace

DTrace Variables and Associative Arrays

DTrace specifies both scalar variables and associative arrays.

Variable scope can be global, thread local or clause local. Thread local variables allow separate storage for each thread's copy of that variable and can be expressed using the following:

```
self->varname = 123;
```

Clause local variables are only active for the duration of the clause lifecycle and can be expressed using the following:

```
this->varname = 123;
```

Associative arrays are used to represent collections of data elements that can be retrieved by specifying a name called a 'key' and are expressed in the following form:

```
name[key] = expression;
```

Common Built-in Variables	Description
args[]	The typed arguments to the current probe – accessed as args[0], args[1], ... but the type corresponds to probe in question
psinfo_t *curpsinfo	The process state of the process associated with the current thread, as described by <code>proc(4)</code>
string execname	The name that was passed to <code>exec(2)</code> to execute the current process
pid_t pid	The process ID of the current process
string probefunc	The function name portion of the current probe's description
string probemod	The module name portion of the current probe's description
string probename	The name portion of the current probe's description
string probeprov	The provider name portion of the current probe's description
uint64_t timestamp	The current value of a nanosecond timestamp counter
uint64_t vtimestamp	The current value of a nanosecond timestamp counter that is virtualized to the amount of time that the current thread has been running on CPU, minus time spent in predicates and actions

DTrace Aggregations, Actions and Subroutines

DTrace provides several built-in aggregating functions to aggregate data rather than rely on individual data points. Aggregations can be expressed using the following:

```
@name[key] = aggfunc (args);
```

Common Aggregation Functions	Description
count	Number of times that the count function is called
sum	The total value of the specified expressions
avg	The average of the specified expressions
min	The smallest value among the specified expressions
max	The largest value among the specified expressions
lquantize	A linear frequency distribution of values of the specified expression, sized by a specific range
quantize	A power-of-two frequency distribution of values of the specified expression.
clear	Clear values in an aggregation
trunc	Truncate aggregation data to certain values

Actions enable DTrace to interact with the system outside – whether to record data or perform destructive behavior (with required security privileges). Subroutines are used to affect internal state such as string manipulation

Common Actions and Subroutines	Description
trace	Outputs the result of an expression to the buffer
printf	Outputs the arguments to the buffer with a specified format
printa	Outputs the aggregation arguments to the buffer with a specified format
stack	Outputs kernel stack to the buffer
ustack	Outputs the user-level stack to the buffer
stop	Stops the process that fired the probe (destructive)
copyinstr	Copies string from address referenced by pointer to the buffer
strjoin	Concatenates two strings
strlen	Returns length of a string

Oracle Solaris 11 Administrator's Cheat Sheet for DTrace

Useful DTrace One Liners

Trace the creation of new processes and output their arguments:

```
# dtrace -n 'proc:::exec-success { trace(curpsinfo->pr_psargs); }'
```

Trace files opened/created by process name:

```
# dtrace -n 'syscall::openat*:entry { printf("%s %s",execname,copyinstr(arg1)); }'
```

Trace the number of system calls made by process name:

```
# dtrace -n 'syscall:::entry { @num[execname] = count(); }'
```

Trace the process name every time a system call is made:

```
# dtrace -n 'syscall:::entry { trace(execname); }'
```

Trace the number of system calls made for each system call:

```
# dtrace -n 'syscall:::entry { @num[probefunc] = count(); }'
```

Trace the number of system calls made by process id:

```
# dtrace -n 'syscall:::entry { @num[pid,execname] = count(); }'
```

Trace lock times by process name:

```
# dtrace -n 'lockstat:::adaptive-block { @time[execname] = sum(arg1); }'
```

Trace file I/O by process name (measured in blocks):

```
# dtrace -n 'io:::start { printf("%d %s %d",pid,execname,args[0]->b_bcount); }'
```

Trace the writes in bytes by process name:

```
# dtrace -n 'sysinfo:::writech { @bytes[execname] = sum(arg0); }'
```

The DTrace Toolkit

The DTrace Toolkit includes a number of pre-written scripts for common system tasks and has been included in Oracle Solaris 11 by default. These can be found in `/usr/dtrace/DTT` and cover a variety of areas including file and disk I/O, memory, CPU, and network.

Oracle Solaris 11 Administrator's Cheat Sheet for DTrace

Contact Us

For more information about Oracle Solaris 11, visit oracle.com/solaris or call +1.800.ORACLE1 to speak to an Oracle representative. Last updated: April 11, 2013.



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd. 0410

Hardware and Software, Engineered to Work Together