

# OpenSolaris Cheatsheet

## From Genunix

The OpenSolaris Cheatsheet is a very quick reference to some of the new features of [OpenSolaris](http://opensolaris.org).

## Service Management Facility (SMF)

<http://opensolaris.org/os/community/smf>

### Informative Commands

```
svcs -a           # State of all the services on the system
svcs -l <FMRI>   # Detailed info about a service instance
svcs -d <FMRI>   # Show the dependencies for a given service instance
svcs -D <FMRI>   # Show the dependents for a given service instance
svcs -p <FMRI>   # Show the process ID for a given service instance
svcs -x [FMRI]   # Explain why a service instance has failed
svcs -v [FMRI]   # Show verbose information for a given service instance
```

```
inetadm -l <FMRI> # Show detailed information about an inetd service
```

\* FMRI: Fault Management Resource Identifier

### Administrative Commands

```
svcadm enable <FMRI>           # Attempts to enable the service
svcadm disable [-t] <FMRI>     # Disable a given service (temporarily until the next reboot when -t specified)
svcadm restart <FMRI>         # Restart a given service
svcadm refresh <FMRI>         # Re-read the config info
svcadm mark <state> <FMRI>     # Manually put a service in maintenance or degraded state
svcadm clear <FMRI>           # To clear the maintenance or degraded state
                               # after rectifying the fault
svcadm milestone <mstone>|all  # Enable/disable the services for the milestone

inetadm -e <FMRI>             # Attempt to enable a given inetd-based service
inetadm -d <FMRI>             # Disable a given inetd-based service
```

### Configuration Commands

```
svccfg # To manipulate the data in the repository
svcprop # To display info about a service
inetadm # To display/modify inet related services
```

The svccfg sub-commands are:

```
svccfg select <FMRI>
  list
  listprop
  setprop <property> = <value>
  delete <FMRI>
  validate <manifest.xml>
  import <manifest.xml>
  unselect
  listsnap
  selectsnap
  revert [<snapshot>]
```

### Files

- Logs: /var/svc/log and /etc/svc/volatile
- Manifests: /var/svc/manifest/
- Methods: /lib/svc/method/

To see the log file location :

```
svcs -l <FMRI> | grep logfile
```

# Zones

<http://opensolaris.org/os/community/zones>

## Commands

```
zoneadm
zonecfg
zonename
```

## Creating a zone

```
# zonecfg -z myzone1
myzone1: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:myzone1> create
zonecfg:myzone1> add fs
zonecfg:myzone1:fs> set dir=/mnt/local
zonecfg:myzone1:fs> set special=/opt/sfw
zonecfg:myzone1:fs> set type=lofs
zonecfg:myzone1:fs> set options=[ro,nodevices]
zonecfg:myzone1:fs> end
zonecfg:myzone1> add rctl
zonecfg:myzone1:rctl> set name=zone.cpu-shares
zonecfg:myzone1:rctl> add value (priv=privileged, limit=1,action=none)
zonecfg:myzone1:rctl> end
zonecfg:myzone1> add attr
zonecfg:myzone1:attr> set name=comment
zonecfg:myzone1:attr> set type=string
zonecfg:myzone1:attr> set value="firstzone"
zonecfg:myzone1:attr> end
zonecfg:myzone1> set autoboot=true
zonecfg:myzone1> set zonepath=/export/home/zone/myzone1
zonecfg:myzone1> add net
zonecfg:myzone1:net> set address=10.6.49.141
zonecfg:myzone1:net> set physical=hme0
zonecfg:myzone1:net> end
zonecfg:myzone1> verify
zonecfg:myzone1> commit
zonecfg:myzone1> exit
```

Now let us install and boot the zone:

```
# zoneadm -z myzone1 install
# zoneadm -z myzone1 boot
```

## DTrace

<http://opensolaris.org/os/community/dtrace/>

DTrace Probe: *provider:module:function:name*

Eg: **proc:genunix:exec-common:exec**

## DTrace One-liners

Files opened by process:

```
# dtrace -n 'syscall::open*:entry { printf("%s %s",execname,copyinstr(arg0)); }'
dtrace: description 'syscall::open*:entry ' matched 2 probes
CPU ID FUNCTION:NAME
  0   15  open:entry ls /var/ld/ld.config
  0   15  open:entry ls /lib/libc.so.1
```

New processes with arguments:

```
# dtrace -n 'proc:::exec-success { trace(curpsinfo->pr_psargs); }'
dtrace: description 'proc:::exec-success ' matched 1 probe
CPU ID FUNCTION:NAME
  0   3995  exec_common:exec-success /usr/bin/sh clear
  0   3995  exec_common:exec-success /usr/bin/tput clear
```

Write size distribution by process:

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

<... wait for a few seconds ...>

^C

```
in.telnetd
value  ----- Distribution ----- count
  1 |
  2 | @@@@@@@@@@@@@@@@@@@@@@
  4 | @@
  8 | @@@@@@@@@@@@@@@@@@
 16 |
 32 |
 64 |
128 |
256 | @@@@@@@@@@
512 |
```

```
svc.configd
value  ----- Distribution ----- count
  2 |
  4 | @@@@@@
  8 | @@
 16 |
 32 |
 64 |
128 |
256 |
512 |
1024 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
2048 |
```

Disk size by process:

```
# dtrace -n 'io:::start { printf("%d %s %d", pid,execname,args[0]->b_bcount); }'
CPU    ID  FUNCTION:NAME
  0    3940  bdev_strategy:start 21108 find 1024
  0    3940  bdev_strategy:start 21108 find 8192
  0    3940  bdev_strategy:start 5752 screen-4.0.2 8192
```

## References

- Numerous scripts in `/usr/demo/dtrace` directory
- DTrace Toolkit: <http://www.opensolaris.org/os/community/dtrace/dtracetoolkit/>

## ZFS

<http://opensolaris.org/os/community/zfs/>

To create your first pool:

```
# zpool create tank c1t2d0
```

You now have a single-disk storage pool named `tank`, with a single filesystem mounted at `/tank`.

If you want a mirrored storage for mail and home dirs:

```
# zpool create tank mirror c1t2d0 c2t2d0
# zfs create tank/mail
# zfs set mountpoint=/var/mail tank/mail
```

Create home dirs and mount them all in `/export/home/<username>`:

```
# zfs create tank/home
# zfs set mountpoint=/export/home tank/home
# zfs create tank/home/ahrens
# zfs create tank/home/bonwick
```

You can also create a RAID-Z pool:

```
# zfs create tank raidz c0t0d0 c0t0d1
```

Or a stripe:

```
# zfs create tank c0t0d0 c0t0d1
```

Filesystems in ZFS are hierarchical; each one inherits properties from above. In this example, the mountpoint property is inherited as a pathname prefix. That is, `tank/home/ahrens` is automatically mounted at `/export/home/ahrens`.

ZFS provides built-in compression:

```
# zfs set compression=on tank/home
```

To give *ahrensa* 10G quota:

```
# zfs set quota=10g tank/home/ahrens
```

To give *bonwick* a 100G reservation:

```
# zfs set reservation=100g tank/home/bonwick
```

To automatically NFS-export all home directories:

```
# zfs set sharenfs=rw tank/home
```

To scrub all disks & verify the integrity:

```
# zpool scrub tank
```

To replace a flaky disk:

```
# zpool replace tank c2t2d0 c4t1d0
```

To add more space:

```
# zpool add tank mirror c5t1d0 c6t1d0
```

To move your pool from SPARC machine *sparky* to AMD machine *amdy*:

On *sparky*:

```
# zpool export tank
```

Physically move your disks from *sparky* to *amdy*.

Then, on *amdy*:

```
# zpool import tank
```

Everything will just work -- ZFS has ***adaptive endianness*** to cope with different byte order on different platforms.

To remove a disk from a stripe or RAID-Z: ack! you can't do this yet. It's listed as something to be implemented at some point. In the mean time, you can replace disks with other disks.

You can remove one half of a mirror though, as long there are other copies:

```
# zpool detach tank c5t1d0
```