System administration

Packages and probes

Douglas Temple

duggles@netsoc.tcd.ie

For
DU Internet Society [NETSOC]

5th December, 2016

e

Tonight's outline

- Package managers for RHEL/Debian
- What to do with multiple versions/types of software
- Building stuff yourself
- Local query and monitoring tools

Package management

- **Problem**: Software compilation/installation/configuration is hard
- **Solution**: Get precompiled binaries and stock configs from distro!
- Packages are compressed archives with some extra scripting
- Signed by package maintainers for security
- Different distros have different behaviours
- We focus on RPM- and DEB-based packages (RIP Arch)
- There are high-level package managers, and low-level package manipulators (my words)
- High-level:
  - `aptitude`, `apt-get`, `apt` (new!) - DEB
  - `yum`, `dnf` (new!) - RPM
- Low-level:
  - `dpkg` - DEB
  - `rpm` - RPM

High-level package management

Update package list apt-get update	yum check-update
Update system apt-get upgrade	yum update
List repositories cat /etc/apt/sources.list	yum repolist
Add repository edit /etc/apt/sources.list	add to /etc/yum.repos.d/
Remove repository edit /etc/apt/sources.list	remove from /etc/yum.repos.d/
Search by package name apt-cache search *pkg-name*	yum list *pkg-name*
Search by *pattern* apt-cache search *pattern*	yum search *pattern*
Search by file name apt-file search path	yum provides file

High-level package management

Show package information apt-cache show *pkg-name*	yum info *pkg-name*
Install from repository apt-get install *pkg-name*	yum install *pkg-name*
Update package apt-get install *pkg-name*	yum update *pkg-name*
Remove package apt-get remove *pkg-name*	yum erase *pkg-name*
Install from package file dpkg -i *pkg-name*	yum localinstall *pkg-name*
Purge package apt-get purge *pkg-name*	yum erase *pkg-name*

Low-level package manipulation

Install package dpkg -i *pkg-name*	rpm -i *pkg-name*
Uninstall package dpkg -r *pkg-name*	rpm -e *pkg-name*
Show package description dpkg -i *pkg-name*	rpm -qi *pkg-name*
List installed packages dpkg -l -a	rpm -qa
Find package containing file dpkg -S *file*	rpm -qf *file*
List files in package dpkg -L *pkg-name*	rpm -ql *pkg-name*

Misc. package stuff

Cracking open packages

- rpm2cpio *rpmfile* | cpio -idmv
- dpkg-deb -x *debfile*
- Alternative: ar -vx *debfile*; tar -xzvf data.tar.gz
  - Also get control.tar.gz and debian-binary

Reconfiguring packages

- dpkg-reconfigure *package-name*
- Not in rpm :(

Some fiddling with package configuration

Diversions

- Sometimes package updates break things
- Commonly caused by replacing config files
- Can avoid this by *diverting* package
- `dpkg-divert --add /some/file --rename /some/where/else`

Multiple architectures

- Sometimes you want multiple architectures (e.g. i386 on x86_64)
- `dpkg --add-architecture=i386` does just that
- `apt install libfoo:i386` will install the i386 version
- Might require special packages for certain things
  - `lib32stdc++6` `libc6-i386`
  - `gcc-6-multilib`

Multiple versions of same software

Some solutions

- Name things differently (`python3`, `python2.7`, etc.)
- Choose one for your users
  - `update-alternatives --config java`
- Use environment modules

Environement modules

- Simple environment manipulation (modifies PATH, INCLUDE_PATH, etc.)
- Couples into your shell and uses simple TCL for description of dependencies

Sample environement module file

```
%Module1.0
module-whatis "GCC environment for 5.2"
prepend-path PATH /usr/support/modules/gnu-5.2/bin
prepend-path MANPATH /usr/support/modules/gnu-5.2/share/man
prepend-path INFOPATH /usr/support/modules/gnu-5.2/share/info
proc ModulesHelp { } {
puts stderr "gcc 5.2 environment"
puts stderr "\n"
puts stderr "configured with: ACML, FFTW3, ATLAS, ScaLAPACK, libint, libunwind, OpenMPI"
}
prepend-path LD_LIBRARY_PATH /usr/support/modules/gnu-5.2/lib
prepend-path LD_LIBRARY_PATH /usr/support/modules/gnu-5.2/lib64
prepend-path LD_RUN_PATH /usr/support/modules/gnu-5.2/lib
prepend-path LIBRARY_PATH /usr/support/modules/gnu-5.2/lib
prepend-path LIBRARY_PATH /usr/support/modules/gnu-5.2/lib64
prepend-path LD_RUN_PATH /usr/support/modules/gnu-5.2/lib
prepend-path LD_RUN_PATH /usr/support/modules/gnu-5.2/lib64
prepend-path INCLUDE_PATH /usr/support/modules/gnu-5.2/include
prepend-path INCLUDE /usr/support/modules/gnu-5.2/include
prepend-path CPLUS_INCLUDE_PATH /usr/support/modules/gnu-5.2/include
prepend-path C_INCLUDE_PATH /usr/support/modules/gnu-5.2/include
```

Building it yourself

- Huge amount of established software is written using autotools
- A convention has been to have an `INSTALL` text file as a readme
- Most autotools programs can be built with the command: `./configure && make && make install`
- `configure` is the autotools setup which queries your system and generates the final makefile
- Compilation can be sped up (some times) by doing `make -j N`, N being related to the number of cores
- The standard `make install` will install the compiled binaries (and other things) into the standard hierarchy below some prefix
  - Default is usually something like `/usr` so you need root privileges
  - Alternatively you can do something like `./configure --prefix=/home/myuser/.localinstall`

More

- Autotools is a pain to develop with
- CMake is somewhat more pleasant
- It essentially replaces `./configure`

Monitoring

- Comes in two flavours:
  - Active
  - Passive
- When developing code it can be useful to actively monitor machine state
- `strace` - system call trace
- `ltrace` - library call trace
- `systemtap` - Hugh-Mungus system profiler
- `perf` - Very low-level performance metrics

perf example

```
sudo perf stat -B dd if=/dev/zero of=/dev/null count=1000000
1000000+0 records in
1000000+0 records out
512000000 bytes (512 MB, 488 MiB) copied, 0.227182 s, 2.3 GB/s

 Performance counter stats for 'dd if=/dev/zero of=/dev/null count=1000000':

        228.205614      task-clock (msec)         #    0.988 CPUs utilized
                 1      context-switches          #    0.004 K/sec
                 0      cpu-migrations            #    0.000 K/sec
                69      page-faults               #    0.302 K/sec
       716,225,259      cycles                    #    3.139 GHz
     1,448,054,958      instructions              #    2.02  insn per cycle
       302,209,028      branches                  # 1324.284 M/sec
         1,009,682      branch-misses             #    0.33% of all branches

       0.231090674 seconds time elapsed
```

Computer montoring tools

- Processes: `htop`, `top`, `ps`, `uptime`
- RAM: `free`, `vmstat`, `slabtop`
- I/O: `iostat`, `vmstat`, `iotop`
- Network: `netstat`, `ss`, `nicstat`, `iftop`, `bmon`, `ethtool`
- Disk: `du`, `df`

Process states

- R Run state: The process is currently running on the CPU
- S Sleep state: The process is not doing anything
- D Uninterruptible sleep: The process is waiting for I/O to continue running
- Z Zombie state: The processes's parent has not reaped its child (typically bad)
- T Stopped state: The process has been paused by a signal or trace

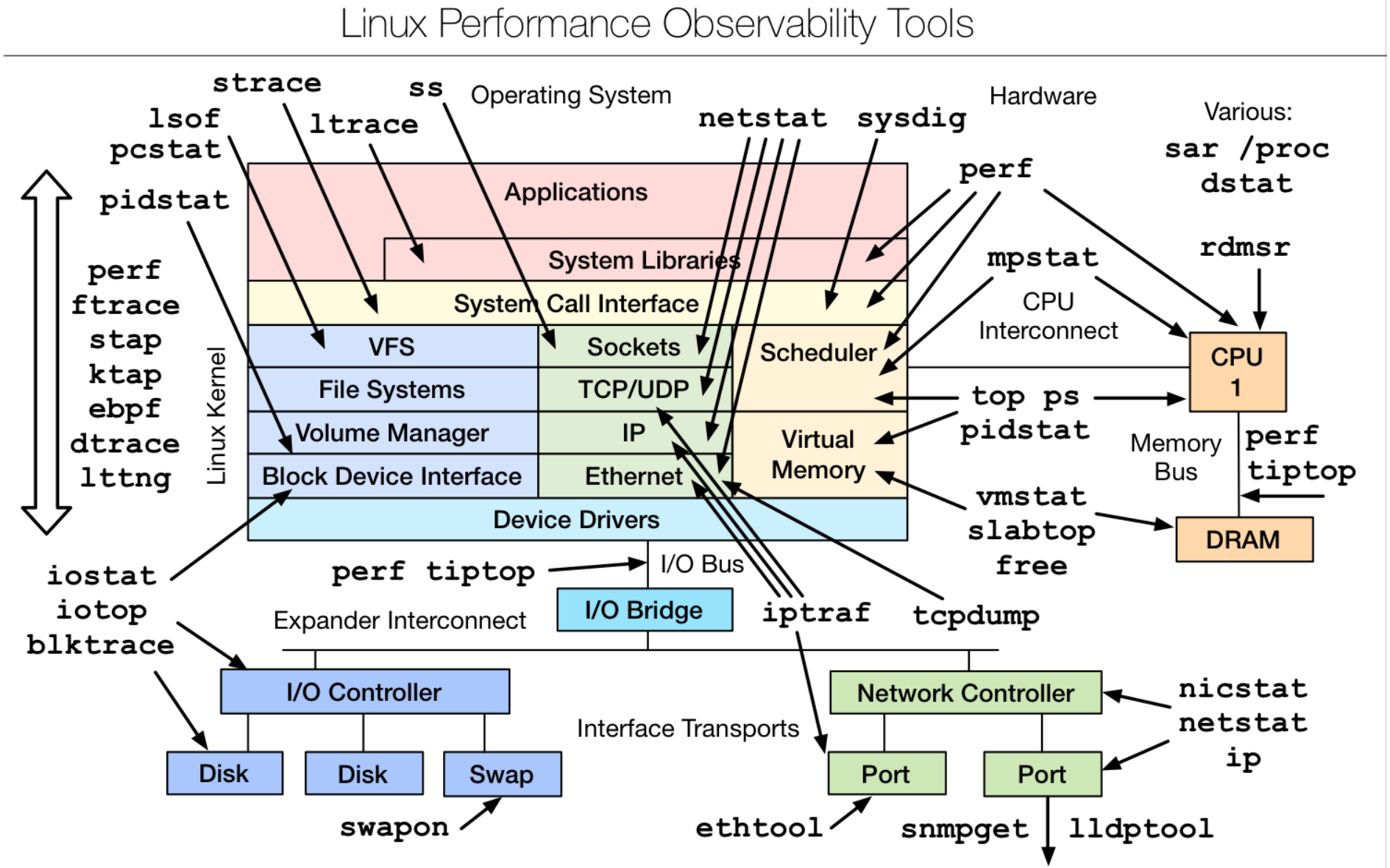
Process commands

Command	Result
ps	Shows processes running in that shell
ps aux	Show processes of all users, and those not on a terminal
ps -eF	As above, but using GNU's syntax
ps axuf	As above, but show the process tree
pstree	Different way to view the process tree
pgrep *name*	Search for process containing *name*, return PIDs
top	Process manager ('q' to exit)
free [-m]	View free memory (-m in MB)
uptime	Show uptime and load

- Load is average CPU utilization in 1, 5, and 15 minute intervals
- There is a load of 1 for each process waiting on I/O or in the run state

```
~:$ ps aux
USER PID %CPU %MEM    VSZ   RSS TTY STAT START TIME COMMAND
root   1  0.0  0.0 21444  1212 ?   Ss   Jul11 0:01 /sbin/init
```

Totally stolen



Passive monitoring tools

- Munin - 5 minute averages of many different metrics (doesn't scale fantastically)
- Nagios + NRPE - Easy polling of certain system metrics (+alerts)
- Ganglia - Highly distributed monitoring solution