Cgroups, chrony, sshd

Agenda

- Cgroups
- chrony
- sshd
- OS monitoring, debugging and logging
- User management

Control groups

Cgroups

Control groups (or cgroups as they are commonly known) are a feature provided by the Linux kernel to manage, restrict, and audit groups of processes. Compared to other approaches like the nice command or /etc/security/limits.conf, cgroups are more flexible as they can operate on (sub)sets of processes (possibly with different system users).

Control groups can be accessed with various tools:

- using directives in systemd unit files to specify limits for services and slices
- by accessing the cgroup filesystem directly
- via tools like cgcreate, cgexec and cgclassify (part of the libcgroupAUR package)
- using the "rules engine daemon" to automatically move certain users/groups/commands to groups (/etc/cgrules.conf and /usr/lib/systemd/system/cgconfig.service) (part of the libcgroupAUR package)
- and through other software such as Linux Containers (LXC) virtualization
- cgmanager is deprecated and unsupported as it does not work with systemd versions 232 and above.

With systemd: **Installing**

systemd is the preferred and easiest method of invoking and configuring cgroups as it is a part of default installation.

Make sure you have one of these packages installed for automated cgroup handling:

- systemd - for controlling resources of a systemd service.
- libcgroupAUR - set of standalone tools (cgcreate, cgclassify, persistence via cgconfig.conf).

With systemd: **Hierarchy**

Current cgroup hierarchy can be seen with **systemctl status** or **systemd-cgls** command.

```
$ systemctl status
```

```
● myarchlinux
    State: running
     Jobs: 0 queued
   Failed: 0 units
    Since: Wed 2019-12-04 22:16:28 UTC; 1 day 4h ago
   CGroup: /
           ├─user.slice
           │ └─user-1000.slice
           │   ├─user@1000.service
           │   │ ├─gnome-shell-wayland.service
           │   │ │ ├─ 1129 /usr/bin/gnome-shell
           │   │ ├─gnome-terminal-server.service
           │   │ │ ├─33519 /usr/lib/gnome-terminal-server
           │   │ │ ├─37298 fish
           │   │ │ └─39239 systemctl status
           │   │ ├─init.scope
           │   │ │ ├─1066 /usr/lib/systemd/systemd --user
           │   │ │ └─1067 (sd-pam)
           │   └─session-2.scope
           │     ├─1053 gdm-session-worker [pam/gdm-password]
           │     ├─1078 /usr/bin/gnome-keyring-daemon --daemonize --login
           │     ├─1082 /usr/lib/gdm-wayland-session /usr/bin/gnome-session
           │     ├─1086 /usr/lib/gnome-session-binary
           │     └─3514 /usr/bin/ssh-agent -D -a /run/user/1000/keyring/.ssh
           ├─init.scope
           │ └─1 /sbin/init
           └─system.slice
             ├─systemd-udevd.service
             │ └─285 /usr/lib/systemd/systemd-udevd
             ├─systemd-journald.service
             │ └─272 /usr/lib/systemd/systemd-journald
             ├─NetworkManager.service
             │ └─656 /usr/bin/NetworkManager --no-daemon
             ├─gdm.service
             │ └─668 /usr/bin/gdm
             └─systemd-logind.service
               └─654 /usr/lib/systemd/systemd-logind
```

With systemd: **Find cgroup of a process**

The cgroup name of a process can be found in /proc/PID/cgroup.

For example, the cgroup of the shell:

```
$ cat /proc/self/cgroup
0::/user.slice/user-1000.slice/session-3.scope
```

With systemd: **cgroup resource usage**

The **systemd-cgtop** command can be used to see the resource usage:

```
$ systemd-cgtop

Control Group                           Tasks   %CPU   Memory  Input/s Output/s
user.slice                                540  152,8     3.3G        -        -
user.slice/user-1000.slice                540  152,8     3.3G        -        -
user.slice/u…000.slice/session-1.scope    425  149,5     3.1G        -        -
system.slice                               37      -   215.6M        -        -
```

With systemd: **custom cgroups**

systemd.slice systemd unit files can be used to define a custom cgroup configuration. They must be placed in a systemd directory, such as /etc/systemd/system/. The resource control options that can be assigned are documented in systemd.resource-control.

This is an example slice unit that only allows 30% of one CPU to be used:

```
/etc/systemd/system/my.slice
```

```
[Slice]
CPUQuota=30%
```

Remember to run systemctl daemon-reload to pick up any new or changed .slice files.

With systemd: **As service**

Service unit file

Resources can be directly specified in service definition or as a Systemd#Drop-in files

```
[Service]
MemoryMax=1G # Limit service to 1 gigabyte
```

With systemd: **As service**

systemd-run can be used to run a command in a specific slice.

```
$ systemd-run --slice=my.slice command
```

--uid=username option can be used to spawn the command as specific user.

```
$ systemd-run --uid=username --slice=my.slice command
```

The --shell option can be used to spawn a command shell inside the slice.

With systemd: **As unprivileged user**

Unprivileged users can divide the resources provided to them into new cgroups, if some conditions are met. Cgroups v2 must be utilized for a non-root user to be allowed managing cgroup resources.

Switching to cgroups v2

Linux enables both v1 and v2 cgroups. However by default systemd mounts cgroup v1.

You could run the following command:

```
grep cgroup /proc/filesystems
```

If your system supports cgroup v2, you would see:

```
nodev cgroup
nodev cgroup2
```

On a system with only cgroup v1, you would only see:

```
nodev cgroup
```

Switching to cgroups v2 requires one of the following Kernel parameters at boot time:

- systemd.unified_cgroup_hierarchy=1 - Systemd will mount /sys/fs/cgroup as cgroup v2
- cgroup_no_v1="all" - The kernel will disable all v1 cgroup controllers

With systemd: **As unprivileged user**

Alternatively you can mount cgroups v2 manually:

```
# mount -t cgroup2 none /sys/fs/cgroup
```

Verify that v2 cgroups have been mounted:

```
$ ls /sys/fs/cgroup
```

```
cgroup.controllers      cgroup.subtree_control  init.scope/      system.slice/
cgroup.max.depth        cgroup.threads          io.cost.model    user.slice/
cgroup.max.descendants  cpu.pressure            io.cost.qos
cgroup.procs            cpuset.cpus.effective   io.pressure
cgroup.stat             cpuset.mems.effective   memory.pressure
```

If you see something like this, you are still making use of v1 cgroups:

```
$ ls /sys/fs/cgroup
```

```
blkio/    cpu,cpuacct/  freezer/  net_cls@          perf_event/  systemd/
cpu@      cpuset/       hugetlb/  net_cls,net_prio/ pids/        unified/
cpuacct@  devices/      memory/   net_prio@         rdma/
```

With systemd: **As unprivileged user**

Controller types

Controller	Can be controlled by user	Options
cpu	Requires delegation	CPUAccounting, CPUWeight, CPUQuota, AllowedCPUs, AllowedMemoryNodes
io	Requires delegation	IOWeight, IOReadBandwidthMax, IOWriteBandwidthMax, IODeviceLatencyTargetSec
memory	Yes	MemoryLow, MemoryHigh, MemoryMax, MemorySwapMax
pids	Yes	TasksMax
rdma	No	?
eBPF	No	IPAddressDeny, DeviceAllow, DevicePolicy

Note: eBPF is technically not a controller but those systemd options implemented using it and only root is allowed to set them.

With systemd: **As unprivileged user**

User delegation

For user to control cpu and io resources, the resources need to be delegated. This can be done by creating a unit overload.

For example if your user id is 1000:

```
# systemctl edit user@1000.service
```

```
[Service]
Delegate=yes
```

Reboot and verify that the slice your user session is under has cpu and io controller:

```
$ cat /sys/fs/cgroup/user.slice/user-1000.slice/cgroup.controllers
```

```
cpuset cpu io memory pids
```

With systemd: **As unprivileged user**

User-defined slices

The user slice files can be placed in ~/.config/systemd/user/.

To run the command under certain slice:

```
$ systemd-run --user --slice=my.slice command
```

You can also run your login shell inside the slice:

```
$ systemd-run --user --slice=my.slice --shell
```

With libcgroup: **Ad-hoc groups**

One of the powers of cgroups is that you can create "ad-hoc" groups on the fly. You can even grant the privileges to create custom groups to regular users. groupname is the cgroup name:

```
# cgcreate -a user -t user -g memory,cpu:groupname
```

Now all the tunables in the group groupname are writable by your user:

```
$ ls -l /sys/fs/cgroup/memory/groupname
```

```
total 0
-rwxrwxr-x 1 user root 0 Sep 25 00:39 cgroup.event_control
-rwxrwxr-x 1 user root 0 Sep 25 00:39 cgroup.procs
-rwxrwxr-x 1 user root 0 Sep 25 00:39 cpu.rt_period_us
-rwxrwxr-x 1 user root 0 Sep 25 00:39 cpu.rt_runtime_us
-rwxrwxr-x 1 user root 0 Sep 25 00:39 cpu.shares
-rwxrwxr-x 1 user root 0 Sep 25 00:39 notify_on_release
-rwxrwxr-x 1 user root 0 Sep 25 00:39 tasks
```

With libcgroup: **Ad-hoc groups**

Cgroups are hierarchical, so you can create as many subgroups as you like. If a normal user wants to run a bash shell under a new subgroup called foo:

```
$ cgcreate -g memory,cpu:groupname/foo
$ cgexec    -g memory,cpu:groupname/foo bash
```

To make sure (only meaningful for legacy (v1) cgroups):

```
$ cat /proc/self/cgroup
```

```
11:memory:/groupname/foo
6:cpu:/groupname/foo
```

With libcgroup: **Ad-hoc groups**

A new subdirectory was created for this group. To limit the memory usage of all processes in this group to 10 MB, run the following:

```
$ echo 10000000 > /sys/fs/cgroup/memory/groupname/foo/memory.limit_in_bytes
```

Note that the memory limit applies to RAM use only -- once tasks hit this limit, they will begin to swap. But it won't affect the performance of other processes significantly.

Similarly you can change the CPU priority ("shares") of this group. By default all groups have 1024 shares. A group with 100 shares will get a ~10% portion of the CPU time:

```
$ echo 100 > /sys/fs/cgroup/cpu/groupname/foo/cpu.shares
```

With libcgroup: **Persistent group configuration**

Note: when using Systemd > = 205 to manage cgroups, you can ignore this file entirely.

If you want your cgroups to be created at boot, you can define them in /etc/cgconfig.conf instead. For example, the "groupname" has a permission for $USER and users of group $GROUP to manage limits and add tasks. A subgroup "groupname/foo" group definitions would look like this:

/etc/cgconfig.conf

```
group groupname {
  perm {
# who can manage limits
    admin {
      uid = $USER;
      gid = $GROUP;
    }
# who can add tasks to this group
    task {
      uid = $USER;
      gid = $GROUP;
    }
  }
# create this group in cpu and memory controllers
  cpu { }
  memory { }
}

group groupname/foo {
  cpu {
    cpu.shares = 100;
  }
  memory {
    memory.limit_in_bytes = 10000000;
  }
}
```

Chrony

How to Install and Use Chrony in Linux

Chrony is a flexible implementation of the *Network Time Protocol* (NTP). It is used to synchronize the system clock from different NTP servers, reference clocks or via manual input.

It can also be used NTPv4 server to provide time service to other servers in the same network. It is meant to operate flawlessly under different conditions such as intermittent network connection, heavily loaded networks, changing temperatures which may affect the clock of ordinary computers.

Chrony comes with two programs:

- chronyc – command line interface for chrony
- chronyd – daemon that can be started at boot time

Install Chrony in Linux

On some systems, chrony may be installed by default. Still if the package is missing, you can easily install it. using your default package manager tool on your respective Linux distributions using following command.

```
# yum -y install chrony    [On CentOS/RHEL]
# apt install chrony       [On Debian/Ubuntu]
# dnf -y install chrony    [On Fedora 22+]
```

To check the status of chronyd use the following command.

```
# systemctl status chronyd      [On SystemD]
# /etc/init.d/chronyd status    [On Init]
```

If you want to enable chrony daemon upon boot, you can use the following command.

```
# systemctl enable chronyd       [On SystemD]
# chkconfig --add chronyd        [On Init]
```

Check Chrony Synchronization in Linux

To check if chrony is actually synchronized, we will use it's command line program chronyc, which has the tracking option which will provide relevant information.

```
# chronyc tracking
```

```
root@tecmint:~# chronyc tracking
Reference ID    : C355D708 (office.ipacct.com)
Stratum         : 2
Ref time (UTC)  : Thu Oct 25 07:15:41 2018
System time     : 0.000003145 seconds slow of NTP time
Last offset     : -0.000003148 seconds
RMS offset      : 0.000236665 seconds
Frequency       : 2.191 ppm fast
Residual freq   : +0.000 ppm
Skew            : 0.162 ppm
Root delay      : 0.004256285 seconds
Root dispersion : 0.001030352 seconds
Update interval : 130.5 seconds
Leap status     : Normal
root@tecmint:~# _
```

Check Chrony Synchronization in Linux

The listed files provide the following information:

- *Reference ID* – the reference ID and name to which the computer is currently synced.
- *Stratum* – number of hops to a computer with an attached reference clock.
- *Ref time* – this is the UTC time at which the last measurement from the reference source was made.
- *System time* – delay of system clock from synchronized server.
- *Last offset* – estimated offset of the last clock update.
- *RMS offset* – long term average of the offset value.
- *Frequency* – this is the rate by which the system's clock would be wrong if chronyd is not correcting it. It is provided in ppm (parts per million).
- *Residual freq* – residual frequency indicated the difference between the measurements from reference source and the frequency currently being used.
- *Skew* – estimated error bound of the frequency.
- *Root delay* – total of the network path delays to the stratum computer, from which the computer is being synced.
- *Leap status* – this is the leap status which can have one of the following values – normal, insert second, delete second or not synchronized.

Check Chrony Synchronization in Linux

To check information about chrony's sources, you can issue the following command.

```
# chronyc sources
```

```
root@tecmint:/home/user# chronyc sources
210 Number of sources = 8
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^- chilipepper.canonical.com     2   8   377   216    -29us[  -18us] +/-   48ms
^- pugot.canonical.com           2   7   377    87  -2030us[-2030us] +/-   43ms
^- alphyn.canonical.com          2   8   377   155  -3809us[-3809us] +/-  109ms
^- golem.canonical.com           2   8   377   544   -295us[ -256us] +/-   66ms
^- bigfoot.ieti.eu               2   8   377    89  +2543us[+2543us] +/-   66ms
^- home.mnet.bg                  3   8   377   154   +111us[ +111us] +/-  119ms
^- tryler.ludost.net             2   8   377   155   +213us[ +213us] +/-   52ms
^* office.ipacct.com             1   8   377   157    +67us[  +79us] +/- 3673us
```

Configure Chrony in Linux

The configuration file of chrony is located at /etc/chrony.conf or /etc/chrony/chrony.conf and sample configuration file may look something like this:

```
server 0.rhel.pool.ntp.org iburst
server 1.rhel.pool.ntp.org iburst
server 2.rhel.pool.ntp.org iburst
server 3.rhel.pool.ntp.org iburst

stratumweight 0
driftfile /var/lib/chrony/drift
makestep 10 3
logdir /var/log/chrony
```

Configure Chrony in Linux

The above configuration provide the following information:

- *server* – this directive used to describe a NTP server to sync from.
- *stratumweight* – how much distance should be added per stratum to the sync source. The default value is 0.0001.
- *driftfile* – location and name of the file containing drift data.
- *Makestep* – this directive causes chrony to gradually correct any time offset by speeding or slowing down the clock as required.
- *logdir* – path to chrony's log file.

If you want to step the system clock immediately and ignoring any adjustments currently being in progress, you can use the following command:

```
# systemctl stop chrony          [On SystemD]
# /etc/init.d/chronyd stop       [On Init]
```

sshd

How to Install SSH Server on Linux

sshd is the OpenSSH server process. It listens to incoming connections using the SSH protocol and acts as the server for the protocol. It handles user authentication, encryption, terminal connections, file transfers, and tunneling.

The SSH server usually comes up as a readily installable package on most linux distributions. However, it is not always installed by default. You can try ssh localhost to test if it is running; if it responds with something like Connection refused, then it is not running.

On Debian-derived distributions, the command to install an SSH server is usually:

```
aptitude install openssh-server
```

How to Install SSH Server on Linux

On Red Hat derived distributions, the command would usually be:

```
yum install openssh-server
```

These commands must be run as **root.**

If the server does not start automatically, try using the service sshd start command, or just reboot the computer.

Startup and Roles of Different sshd processes

The sshd process is started when the system boots. The program is usually located at /usr/sbin/sshd. It runs as root. The initial process acts as the master server that listens to incoming connections. Generally this process is the one with the lowest process id or the one that has been running the longest. It is also the parent process of all the other sshd processes. The following command can be used to display the process tree on Linux, and it is easy to see which one is the parent process.

```
ps axjf
```

Startup and Roles of Different sshd processes

For example, it is easy to see in the following output that process 2183 is the master server.

```
  PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
  ...
     1  2183  2183  2183 ?           -1 Ss       0   8:51 /usr/sbin/sshd -D
  2183 12496 12496 12496 ?           -1 Ss       0   0:00  \_ sshd: cessu [priv]
 12496 12567 12496 12496 ?           -1 S    15125  24:07  |   \_ sshd: cessu
  2183 12568 12568 12568 ?           -1 Ss       0   0:00  \_ sshd: cessu [priv]
 12568 12636 12568 12568 ?           -1 S    15125   0:00  |   \_ sshd: cessu@pts/2
 12636 12637 12637 12637 pts/2    12637 Ss+  15125   0:00  |       \_ -zsh
  ...
```

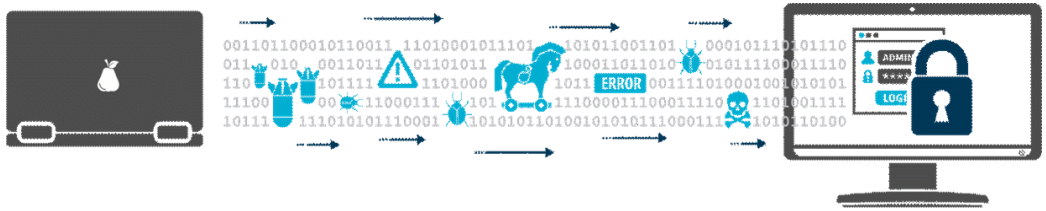
The other sshd processes are child processes that serve a single connection. A new process is created for each new SSH session.

Startup and Roles of Different sshd processes

If the SSH server is upgraded or restarted, only the master server is generally restarted. The server has been designed so that the server processes serving existing connections continue to operate. This minimizes the disruption to users when, for example, server configuration is changed. The easiest way to restart the SSH server is usually to use service sshd restart. However, care should be taken when upgrading configurations remotely, as errors could prevent connecting to the server again (see below).

It is also possible to kill individual processes by killing the server process for a particular user, terminal, or command. This could be done, e.g., using the kill -9 <processid> command.

It is also possible to run multiple master sshd processes on the same system. This is very unusual, but we have seen enterprise customers with more than ten servers running simultaneously with different configurations. Each server would need to listen to a different port and usually would have a different configuration file.



Configuration File

The SSH server has a configuration file, usually /etc/sshd/sshd_config. The configuration file specifies encryption options, authentication options, file locations, logging, and various other parameters. For a detailed description, please the sshd_config documentation.

Logging

The SSH server uses the syslog subsystem for logging. There are many ways to configure syslog and several syslog servers. Many enterprises also collect syslog data into their centralized SIEM (Security Incident and Event Management) system.

On most systems, syslog is configured to log SSH-related messages by default into files under /var/log/. On Debian-derived systems, the default log file is usually /var/log/auth.log. On Red Hat derived systems, the default log file is usually /var/log/secure.

Both the syslog facility and logging level can be configured in the server configuration file. It is **strongly advised to set the logging level to VERBOSE**, so that fingerprints for SSH key access get properly logged. Newest OpenSSH versions may log them automatically, but many Linux distributions still come with versions that don't log fingerprints without this setting. See SSH key management for why this is important.

Debugging SSH Connection

Sometimes logging into an SSH server just doesn't seem to work, and it can be difficult to figure out what the problem is. There are basically three tools that help diagnose connection and authentication problems:

Debugging SSH Connection: **SSH Client - v Option**

The first approach is to add the -v option when calling the client on the command line. For example:

```
ssh -v [user@]host
```

This will print verbose debugging output that can usually identify what the problem is. Things to check include:

- Does it successfully establish the TCP connection to server? If not, it could be DNS or routing problem or the server could be down. If the output includes Connection established, then the connection was successful.
- Check the user name that it is trying to authenticate as. Look for a line containing Authenticating to <hostname> as '<username>'.

Debugging SSH Connection: **SSH Client -v Option**

Things to check include:

- Check that it successfully negotiates encryption. If you see a line containing SSH2_MSG_SERVICE_ACCEPT received, then encryption negotiation was successful. If not, then the server or client must be reconfigured. An outdated host key on the client could also cause this (use ssh-keygen -R <hostname> on the client to remove old host key if necessary; see ssh-keygen).
- Look at the authentication methods the server is willing to accept. Look for lines containing Authentications that can continue: <list of methods>. If the method you are trying to use is not included, you need to change the configuration of the server and restart the server. This is a fairly common cause of problems if using anything other than password or public key authentication.
- If you see a line containing Authentication succeeded, then it is not an authentication problem. If login fails after this, then it could be a problem with the user's login shell or, e.g., .bashrc.
- It is fairly common for X11 forwarding to fail. It is disabled by default in the OpenSSH server. You need to edit the sshd_config file on the server to have the line X11Forwarding yes to enable it. It often does not to be enabled on enterprise application servers, but in universities, home environments, and development servers it is usually needed. Again, remember to restart the server.

Debugging SSH Connection: **Log Files**

Looking at the log files can often reveal insights into the cause of the problem. The messages sent to the client are intentionally designed to reveal quite little about the user being logged in as. This is for security reasons. For example, we don't want attacker to be able to test which user accounts exist on a target system. More information about, e.g., authentication failures can often be found in the log file.

Debugging SSH Connection: **Run the Server in Debug Mode**

A system administrator can manually run the server with the **-d** option to get extra verbose output from the server. This is often the last resort when diagnosing connection problems. Usually the cause of authentication failures is quite clearly visible in its output.

It may be desirable to run the new server in a different port then the normal server, so as to not prevent new connections to the server (especially if it remote!). In this case, the server would be run (as root) with something like **sshd -d -p 2222** and then the client would connect with **ssh -p 2222 [user@]host**.

Command-Line Options

It is rare to have to manually provide command options for the SSH server. Generally only people repackaging SSH or creating new linux distributions or new embedded platforms (e.g., IoT devices) would do this.

The following options are available in OpenSSH:

- **-4** Only use IPv4 addresses. This might be used in environments where DNS gives IPv6 addresses but routing does not work for them.
- **-6** Only use IPv6 addresses. This might be used for testing to make sure IPv6 connectivity works.
- **-C connection_spec** Used for testing particular Match blocks in the configuration file, in combination with the -T option. The *connection_spec* is a comma-separated list of *<keyword>=<value>* pairs, where <keyword> can be one of: *user, host, laddr, lport, addr*. Multiple -C options are permitted and combined.
- **-c host_certificate_file** Specifies the path of a file containing the host certificate for the host. The certificate is in OpenSSH's proprietary format.

Command-Line Options

- **-D** Do not detach and become daemon. This is often used when sshd is run using systemd. This allows easier monitoring of the process in such environments. Without this option, the SSH server forks and detaches from terminal, making itself a background daemon process. The latter has been the traditional way to run the SSH server until recently. Many embedded systems would still use the latter.
- **-d** Enables debug mode. The server does not fork, and will exit after processing a single connection. This can be used for diagnosing user authentication and other problems, and usually gives more information about the problem than is set to the client.
- **-E log_file** Appends logs to the specified file, instead of sending them to *syslog*.
- **-e** Write debug logs to standard error. This could be used for debugging.
- **-f config_file** Specifies the path of the server configuration file. By default, */etc/ssh/sshd_config* is used.
- **-g login_grace** Specifies how quickly users must authenticate themselves after opening a connection to the SSH server. The default is 120 seconds, but this can be changed in the server configuration file. The timeout prevents permanently reserving resources on the server by opening an unauthenticated connection to it.

Command-Line Options

- **-h host_key_file** Specifies a file from which to read a host key. The default is to use */etc/ssh/ssh_host_<algorithm>_key* files. Only one host key can be specified for each algorithm.
- **-i** This would be used if the server was run through inetd. However, nobody does it these days.
- **-k timeout** This option is obsolete. It was used with SSH version 1. Its use is strongly discouraged.
- **-o option** Overrides any configuration option specified in the configuration file. This could be useful for testing and running multiple servers on different ports.
- **-p port** Specifies the port that the server listens on. The default is 22. The port can also be specified in the server configuration file.
- **-q** Doesn't send anything to the system log. This is not recommended; the only real use of this option would be for an attacker to hide logins using a backdoor. This option really shouldn't be there.
- **-T** Reads the server configuration file, checks its syntax, and exits. This is useful for checking that the configuration file is ok before restarting the server. Checking the configuration file is especially important if updating the configuration remotely. In fact, in such cases it is best to first test the new configuration by running a second server on a new port, and only restart the primary server after a successful login using the test server. This can be combined with the -C option to test individual Match blocks in the configuration file.

Command-Line Options

- **-t** Checks the validity of the configuration file and referenced keys. See -T for advice on additional testing before restarting a server remotely.
- **-u len** This obscure option has only one useful purpose: specifying -u0 causes dotted IP addresses to be stored in the utmp file (which contains information about logins to the server). This disables DNS lookups by the SSH server, if the authentication mechanism or from= patterns on authorized keys do not require then. Otherwise it would specify the size of an utmp structure on the host, the cases where it needs to be manually specified are very rare.

Q&A