Cheat Sheet: Advanced Linux Commands

Lets Get Started...

This cheat sheet should help you get started with developing a (web) application on Red Hat Enterprise Linux (RHEL). We'll assume you have a VM running RHEL, by - for example - having run through the steps in the "Using Vagrant to Get Started with RHEL" blog

As an example scenario, we are going to pretend we are developing a LAMP (Linux, Apache, MariaDB and PHP) application on single machine running Red Hat Enterprise Linux 7. As a first step, we're going to install Apache, PHP and MariaDB (the drop-in replacement for MySQL that's shipped with Red Hat Enterprise Linux 7), and start the appropriate services:

`# yum -y install httpd mariadb-server php-mysql php`	Installs the correct packages to start developing a LAMP application: the Apache webserver, the base packages for PHP, and a MariaDB server, including MySQL bindings for PHP.
`$ systemctl status httpd`	Show information about httpd, including process ID, child processes, time since startup, what man pages are available, the most recent log messages, and more.
`# systemctl start httpd mariadb`	Start the httpd and mariadb services. Instead of 'start', you can also use stop or restart, for obvious use cases.
`# systemctl enable httpd mariadb`	Enable the httpd and mariadb services to start at next boot. You can also use disable, mask or unmask.

So the framework is installed and services should be running; let's check if everything is ok by checking out the logs. (You must either be a member of the 'adm' group on the system, or run these commands with 'sudo' prepended to them to see all log messages.)

`$ journalctl -f -l`	Show and keep open (-f) the system log, allowing you to see new messages scrolling by. The -l flag prevents truncating of long lines.
`$ journalctl -f -l -u httpd -u mariadb`	Same as above, but only for log messages from the httpd and mariadb services.
`$ journalctl -f -l -u httpd -u mariadb --since -300`	Same as above, only for log messages that are less than 300 seconds (5 minutes) old

Now in order to test our app in the VM, we need the IP address of the server. For that we want to see the IP address configured for the first network card, called 'eth0' in most virtual machines:

`$ nmcli d`	Show the status of all network interfaces
`$ nmcli d show eth0`	Show details of network interface eth0; alternatively you can use 'ip a s eth0'
`# nmcli d connect eth0`	Bring up the network interface eth0. You can use 'disconnect' to bring the interface down.

Now let's drop an example PHP file in /var/www/html to see if everything works

`$ cat << EOF > /var/www/html/test.php`
`<?php`
`  phpinfo();`
`?>`
`EOF`
	All text between the first line and EOF will be added to /var/www/html/test.php. Any existing content in that file will be overwritten. This is called a 'heredoc'.

Now we can download the test.php file from either the same machine, or our local workstation:

`$ curl http://www.someapp.org/test.php`
`$ curl http://10.0.0.10/test.php`
	Use the 'curl' command to perform a download of the test.php file at www.someapp.org or 10.0.0.10, respectively

`$ curl http://localhost:80/someapp/api -v`	Fetch sent and received HTTP GET status, API response payload from the local host
`$ curl https://localhost:443/someapp/api -v -F "arg1=foo" -F "arg2=bar"`	Fetch sent and received HTTPS POST status, API response payload from the local host
`$ host www.someapp.org`	Use the 'host' command to test DNS name resolution; you might need to run 'yum -y install bind-utils' for this command to work.

Generally, files in `/var/www/html` are owned by apache.In a dev environment, you might want to make those files owned by apache and a developer group. Here are some commands that are useful to make that a reality.

`# chown apache:developers test.php`	Change ownership of test.php to "apache" and the "developers" group. (You can only change ownership of a file to another user if you are the superuser, "root".)
`# chmod u+rw,g+rw,o+r test.php`	Change the mode of test.php to allow owner (u) and users in the group (g) to read and write (+rw) it, and the rest of the world (o) to just read (+r) it.
`# chmod g+rw test.php`	Allow users in the group of test.php to read and write it

`# chown -R :developers /var/www/html`	Change ownership of /var/www/html and all files in that directory to the developers group.
`# chmod g+s /var/www/html`	Special command to make sure that all files created in /var/www/html are owned by the group that own /var/www/html; it sets to so-called sticky bit.

Maybe you have a script that you want to use on that server, too. You'll need to make it executable first:

`$ chmod 755 somescript`	Allow the owner of somescript to read, write and execute it, and the rest of the world to just read and execute it.
`$ chmod +x somefile`	Allow execution of somefile

Red Hat Enterprise Linux 7 ships with a security feature called SELinux. SELinux basically labels all files, and then whitelists what labels a program (e.g. Apache) is allowed to read.

`$ ls -lZ test.php`	Show the SELinux label of test.php. Files in /var/www/html need to be labeled httpd_sys_content_t (content readable by Apache) or httpd_sys_rw_content_t (content readable and writable by Apache).
`# ausearch -sv no --comm httpd`	Search the audit log for recently denied events triggered by Apache ('httpd'). Useful for debugging an application that might be running into SELinux related problems.
`# restorecon -FvR /var/www/html`	Use this command to restore the default labels on all files under /var/www/html if different from those mentioned above.
`$ getenforce`	Show what mode SELinux is in: Disabled, Permissive or Enforcing. Switch SELinux to enforcing mode with 'setenforce 1'.
`# semanage fcontext -l | grep '/var/www'`	View all SELinux rules that potentially apply to /var/www in the extensive SELinux docs. Install the policycoreutils-python package with yum to get the 'semanage' command.

If you have a database on a separate server, you need to allow Apache to initiate network connections, which SELinux denies by default. This is done by setting an SELinux boolean.

`$ getsebool  -a`	Show all available SELinux boolean settings
`# setsebool httpd_can_network_connect_db 1`	Tell SELinux to allow httpd to make connections to databases on other servers. Use the -P flag to make permanent.

The above should hopefully get you started with developing on RHEL, but you can do so much more! For example, here are some commands to run a program in the background in your shell.

`$ ./someprogram &`	Start someprogram in the background. You can also just start someprogram and hit CTRL-Z to suspend it and send it to the background.
`$ jobs`	Show all background jobs in current shell; add -l for more information on the jobs.
`$ bg [number]`	Continue suspended job (i.e. a job suspended with CTRL-Z) in the background.
`$ fg [number]`	Bring a background job to the foreground again.

And if you need to get an idea on how your application or system is performing, you might like these commands

`$ free`	Show the amount of free memory. Please note it's not necessarily a problem if Linux seems to use a lot of memory!
`$ vmstat 3`	Every three seconds, show statistics about the system, like utilization, memory in use, etc.
`$ iotop`	Show 'top' like output for disk i/o. Must be root to run this. First install the iotop package with yum.
`$ ps xauww`	Show the system process list

Finally, maybe you want to use Java instead of PHP. These two commands install some programs you might want to use in that case

`# subscription-manager repos --enable rhel-server-rhscl-7-rpms`	Enable the Software Collections repositories to install packages from (required for Maven)
`# yum -y install java-1.8.0-openjdk-devel tomcat maven30 git`	Single command to install your Java compiler, Tomcat webserver, maven and git.

About the Author

Maxim Burgerhout is a solution architect in the Red Hat Benelux team. He is often spotted talking about systems management and infrastructure, including infrastructure automation, implementing self-service deployments and configuration management.

In the past, he's been involved in various migrations from legacy Unix to Red Hat Enterprise Linux. Those migrations always involved making developers feel at home on the new platform by providing the right tools and documentation and getting them up to speed quickly.

Maxim has done some development in Ruby, PHP and Python in the past and is currently learning Java, because, well, just because.

@MaximBurgerhout   Linkedin