



LFS201

Essentials of System Administration

Version 2018-12-25



© Copyright the Linux Foundation 2019. All rights reserved.

The training materials provided or developed by The Linux Foundation in connection with the training services are protected by copyright and other intellectual property rights.

Open source code incorporated herein may have other copyright holders and is used pursuant to the applicable open source license.

The training materials are provided for individual use by participants in the form in which they are provided. They may not be copied, modified, distributed to non-participants or used to provide training to others without the prior written consent of The Linux Foundation.

No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without express prior written consent.

Published by:

the **Linux Foundation**

<https://www.linuxfoundation.org>

No representations or warranties are made with respect to the contents or use of this material, and any express or implied warranties of merchantability or fitness for any particular purpose or specifically disclaimed.

Although third-party application software packages may be referenced herein, this is for demonstration purposes only and shall not constitute an endorsement of any of these software applications.

Linux is a registered trademark of Linus Torvalds. Other trademarks within this course material are the property of their respective owners.

If there are any questions about proper and fair use of the material herein, please contact:

training@linuxfoundation.org

Contents

- 1 Introduction 1**
 - 1.1 Labs 1
- 2 Linux Filesystem Tree Layout 3**
 - 2.1 Labs 3
- 3 Processes 7**
 - 3.1 Labs 7
- 4 Signals 11**
 - 4.1 Labs 11
- 5 Package Management Systems 15**
 - 5.1 Labs 15
- 6 RPM 17**
 - 6.1 Labs 17
- 7 dpkg 21**
 - 7.1 Labs 21
- 8 yum 23**
 - 8.1 Labs 23
- 9 zypper 27**
 - 9.1 Labs 27
- 10 APT 29**
 - 10.1 Labs 29
- 11 System Monitoring 33**
 - 11.1 Labs 33
- 12 Process Monitoring 35**
 - 12.1 Labs 35
- 13 Memory Monitoring and Usage 37**
 - 13.1 Labs 37
- 14 I/O Monitoring and Tuning 39**

| | |
|---|-----------|
| 14.1 Labs | 39 |
| 15 I/O Scheduling ** | 43 |
| 15.1 Labs | 43 |
| 16 Linux Filesystems and the VFS | 47 |
| 16.1 Labs | 47 |
| 17 Disk Partitioning | 49 |
| 17.1 Labs | 49 |
| 18 Filesystem Features: Attributes, Creating, Checking, Mounting | 55 |
| 18.1 Labs | 55 |
| 19 Filesystem Features: Swap, Quotas, Usage | 61 |
| 19.1 Labs | 61 |
| 20 The Ext2/Ext3/Ext4 Filesystems | 65 |
| 20.1 Labs | 65 |
| 21 The XFS and BTRFS Filesystems ** | 69 |
| 21.1 Labs | 69 |
| 22 Encrypting Disks | 71 |
| 22.1 Labs | 71 |
| 23 Logical Volume Management (LVM) | 75 |
| 23.1 Labs | 75 |
| 24 RAID ** | 77 |
| 24.1 Labs | 77 |
| 25 Kernel Services and Configuration | 79 |
| 25.1 Labs | 79 |
| 26 Kernel Modules | 81 |
| 26.1 Labs | 81 |
| 27 Devices and udev | 83 |
| 27.1 Labs | 83 |
| 28 Virtualization Overview | 85 |
| 28.1 Labs | 85 |
| 29 Containers Overview | 95 |
| 29.1 Labs | 95 |
| 30 User Account Management | 99 |
| 30.1 Labs | 99 |

| | |
|--|------------|
| 31 Group Management | 103 |
| 31.1 Labs | 103 |
| 32 File Permissions and Ownership | 105 |
| 32.1 Labs | 105 |
| 33 Pluggable Authentication Modules (PAM) | 109 |
| 33.1 Labs | 109 |
| 34 Network Addresses | 111 |
| 34.1 Labs | 111 |
| 35 Network Devices and Configuration | 113 |
| 35.1 Labs | 113 |
| 36 Firewalls | 119 |
| 36.1 Labs | 119 |
| 37 System Startup and Shutdown | 123 |
| 37.1 Labs | 123 |
| 38 GRUB | 125 |
| 38.1 Labs | 125 |
| 39 Init, SystemV, Upstart, systemd | 127 |
| 39.1 Labs | 127 |
| 40 Backup and Recovery Methods | 131 |
| 40.1 Labs | 131 |
| 41 Linux Security Modules | 135 |
| 41.1 Labs | 135 |
| 42 Local System Security | 141 |
| 42.1 Labs | 141 |
| 43 Basic Troubleshooting | 145 |
| 43.1 Labs | 145 |
| 44 System Rescue | 147 |
| 44.1 Labs | 147 |

Chapter 1

Introduction



1.1 Labs

Exercise 1.1: Configuring the System for **sudo**

It is very dangerous to run a **root shell** unless absolutely necessary: a single typo or other mistake can cause serious (even fatal) damage.

Thus, the sensible procedure is to configure things such that single commands may be run with superuser privilege, by using the **sudo** mechanism. With **sudo** the user only needs to know their own password and never needs to know the root password.

If you are using a distribution such as **Ubuntu**, you may not need to do this lab to get **sudo** configured properly for the course. However, you should still make sure you understand the procedure.

To check if your system is already configured to let the user account you are using run **sudo**, just do a simple command like:

```
$ sudo ls
```

You should be prompted for your user password and then the command should execute. If instead, you get an error message you need to execute the following procedure.

Launch a root shell by typing **su** and then giving the **root** password, not your user password.

On all recent **Linux** distributions you should navigate to the `/etc/sudoers.d` subdirectory and create a file, usually with the name of the user to whom root wishes to grant **sudo** access. However, this convention is not actually necessary as **sudo** will scan all files in this directory as needed. The file can simply contain:

```
student ALL=(ALL) ALL
```

if the user is `student`.

An older practice (which certainly still works) is to add such a line at the end of the file `/etc/sudoers`. It is best to do so using the **visudo** program, which is careful about making sure you use the right syntax in your edit.

You probably also need to set proper permissions on the file by typing:

```
$ chmod 440 /etc/sudoers.d/student
```

(Note some **Linux** distributions may require 400 instead of 440 for the permissions.)

After you have done these steps, exit the root shell by typing `exit` and then try to do `sudo ls` again.

There are many other ways an administrator can configure **sudo**, including specifying only certain permissions for certain users, limiting searched paths etc. The `/etc/sudoers` file is very well self-documented.

However, there is one more setting we highly recommend you do, even if your system already has **sudo** configured. Most distributions establish a different path for finding executables for normal users as compared to root users. In particular the directories `/sbin` and `/usr/sbin` are not searched, since **sudo** inherits the `PATH` of the user, not the full root user.

Thus, in this course we would have to be constantly reminding you of the full path to many system administration utilities; any enhancement to security is probably not worth the extra typing and figuring out which directories these programs are in. Consequently, we suggest you add the following line to the `.bashrc` file in your home directory:

```
PATH=$PATH:/usr/sbin:/sbin
```

If you log out and then log in again (you don't have to reboot) this will be fully effective.

Chapter 2

Linux Filesystem Tree Layout



2.1 Labs

Exercise 2.1: Sizes of the Default Linux Directories

Use the **du** utility to calculate the overall size of each of your system's top-level directories.

Type the command:

```
$ du --help
```

for hints on how to obtain and display this result efficiently.

Solution 2.1

To obtain a full list of directories under / along with their size:

```
$ sudo du --max-depth=1 -hx /
```

```
4.3M    /home
16K      /lost+found
39M      /etc
4.0K     /srv
3.6M     /root
178M     /opt
138M     /boot
6.1G     /usr
1.1G     /var
16K      /mnt
4.0K     /media
869M     /tmp
8.4G     /
```

Where we have used the options:

- `--maxdepth=1`: Just go down one level from / and sum up everything recursively underneath in the tree.
- `-h`: Give human-readable numbers (KB, MB, GB).
- `-x` Stay on one filesystem; don't look at directories that are not on the / partition. In this case that means ignore:

`/dev /proc /run /sys`

because these are pseudo-file systems which exist in memory only; they are just empty mount points when the system is not running. Because this is a **RHEL 7** system, the following mount points are also not followed:

`/bin /sbin /lib /lib64`

since they are just symbolically linked to their counterparts under `/usr`.

Exercise 2.2: Touring the /proc Filesystem

Exactly what you see in this exercise will depend on your kernel version, so you may not match the output shown precisely.

1. As root, **cd** into `/proc` and do a directory listing. This should display a number of files and directories:

```
$ cd /proc
$ ls -F

c7:/proc>ls -F
1/      128/    1510/    20/      2411/    30895/   53/      6925/    802/    951/      kmsg
10/     129/    1511/    2015/    2425/    31/      54/      7/       81/     952/      kpagecgroup
1002/   13/     1512/    2022/    2436/    31449/   55/      70/      813/    957/      kpagecount
1007/   130/    1513/    2023/    2444/    32/      56/      702/     814/    97/       kpageflags
10540/  131/    1514/    20300/   2451/    33/      58/      709/     816/    9742/     loadavg
10590/  13172/  152/     20354/   2457/    34/      585/     71/      817/    98/       locks
10798/  132/    15552/   20380/   2489/    35/      59/      718/     82/     99/       meminfo
10805/  133/    15663/   20388/   25/      36/      60/      719/     83/     9923/     misc
10806/  134/    15737/   20392/   2503/    37/      61/      72/      834/    acpi/     modules
10809/  135/    159/     20396/   2504/    374/     6193/    721/     835/    asound/   mounts@
10810/  136/    15981/   2086/    2531/    379/     62/      723/     84/     buddyinfo mtrr
10813/  137/    16/      2090/    2546/    38/      63/      725/     841/    bus/      net@
10894/  138/    162/     211/    2549/    380/     634/     727/     842/    cgroups   pagetypeinfo
10925/  1384/   1632/    22/      2562/    40/      64/      73/      85/     cmdline   partitions
10932/  1385/   1636/    2205/    25794/   41/      65/      7300/    857/    config.gz sched_debug
10934/  1387/   166/     2209/    26/      42/      662/     74/      86/     consoles  scsi/
10935/  139/    1670/    2212/    2610/    43/      663/     757/     864/    cpuinfo   self@
10941/  1390/   17/      2232/    26108/   44/      665/     758/     867/    crypto    slabinfo
10983/  1393/   17271/   2238/    2619/    4435/   666/     76/      87/     devices   softirqs
10998/  14/     17361/   2296/    2624/    45/      67/      761/     88/     diskstats stat
11/     140/    1793/    2298/    2627/    46/      670/     762/     881/    dma        swaps
11047/  1410/   18/      23/     2644/    468/     671/     765/     886/    driver/    sys/
1105/   1415/  1831/    23042/   2645/    47/      673/     766/     887/    execdomains sysrq-trigger
1121/   1429/  18880/   2344/    2679/    470/     674/     768/     888/    fb         sysvipc/
1123/   1437/  18903/   2348/    27/     484/     678/     769/     889/    filesystems thread-self@
1135/   1445/   19/     2353/    2706/    49/      679/     77/      89/     fs/        timer_list
11420/  146/    19392/   2354/    2762/    492/     68/      771/     9/      interrupts timer_stats
11499/  1463/   19488/   2365/    28/     493/     682/     78/      90/     iomem      tty/
11515/  147/    1954/   23683/   2858/    5/      683/     79/      92/     ioports    uptime
11530/  1476/   1963/    2370/    28730/   50/      686/     793/     921/    irq/       version
1163/   148/   19727/   2372/    28734/   51/      687/     794/     928/    kallsyms   vmallocinfo
1164/   1485/  19734/   2374/    29/     510/     69/      8/      930/    kcore      vmstat
12/     149/   19984/   24/     2973/    514/     690/     80/      931/    keys       zoneinfo
127/    15/     2/      2406/    3/      52/      691/     801/     944/    key-users
```

Notice many of the directory names are numbers; each corresponds to a running process and the name is the **process ID**. An important subdirectory we will discuss later is `/proc/sys`, under which many system parameters can be examined or modified.

2. View the following files:

- `/proc/cpuinfo`:
- `/proc/meminfo`:
- `/proc/mounts`:
- `/proc/swaps`:

- `/proc/version:`
- `/proc/partitions:`
- `/proc/interrupts:`

The names give a pretty good idea about what information they reveal.

Note that this information is not being constantly updated; it is obtained only when one wants to look at it.

3. Take a peek at any random process directory (if it is not a process you own some of the information might be limited unless you use **sudo**):

```
$ ls -F 4435
```

```
attr/      coredump_filter  gid_map      mountinfo    oom_score_adj  sessionid     syscall
autogroup  cpuset          io           mounts       pagemap        setgroups     task/
auxv       cwd@            limits       mountstats   personality     smaps         timerslack_ns
cgroup     environ         loginuid     net/         projid_map     stack         uid_map
clear_refs exe@            map_files/   ns/          root@          stat          wchan
cmdline    fd/            maps         oom_adj      sched          statm
comm       fdinfo/        mem          oom_score    schedstat     status
```

Take a look at some of the fields in here such as: `cmdline`, `cwd`, `environ`, `mem`, and `status`

Chapter 3

Processes



3.1 Labs

Exercise 3.1: Controlling Processes with ulimit

Please do:

```
$ help ulimit
```

and read `/etc/security/limits.conf` before doing the following steps.

1. Start a new shell by typing **bash** (or opening a new terminal) so that your changes are only effective in the new shell. View the current limit on the number of open files and explicitly view the hard and soft limits.
2. Set the limit to the hard limit value and verify if it worked.
3. Set the hard limit to 2048 and verify it worked.
4. Try to set the limit back to the previous value. Did it work?

Solution 3.1

1.

```
$ bash
$ ulimit -n
1024
$ ulimit -S -n
1024
$ ulimit -H -n
4096
```
2.

```
$ ulimit -n hard
$ ulimit -n
4096
```
3.

```
$ ulimit -n 2048
$ ulimit -n
```

```
2048
```

```
4. $ ulimit -n 4096
```

```
bash: ulimit: open files: cannot modify limit: Operation not permitted
```

```
$ ulimit -n
```

```
2048
```

You can't do this anymore!

Note that if we had chosen a different limit, such as stack size (`-s`) we could raise back up again as the hard limit is unlimited.

Exercise 3.2: Examining System V IPC Activity

System V IPC is a rather old method of Inter Process Communication that dates back to the early days of **UNIX**. It involves three mechanisms:

1. **Shared Memory Segments**
2. **Semaphores**
3. **Message Queues**

More modern programs tend to use **POSIX IPC** methods for all three of these mechanisms, but there are still plenty of **System V IPC** applications found in the wild.

To get an overall summary of **System V IPC** activity on your system, do:

```
$ ipcs
```

```
----- Message Queues -----
key          msqid          owner          perms          used-bytes   messages

----- Shared Memory Segments -----
key          shmid          owner          perms          bytes         nattch       status
0x01114703  0              root          600            1000          6            dest
0x00000000  98305          coop          600            4194304       2            dest
0x00000000  196610         coop          600            4194304       2            dest
0x00000000  23068675       coop          700            1138176       2            dest
0x00000000  23101444       coop          600            393216        2            dest
0x00000000  23134213       coop          600            524288        2            dest
0x00000000  24051718       coop          600            393216        2            dest
0x00000000  23756807       coop          600            524288        2            dest
0x00000000  24018952       coop          600            67108864      2            dest
0x00000000  23363593       coop          700            95408         2            dest
0x00000000  1441811        coop          600            2097152       2            dest

----- Semaphore Arrays -----
key          semid          owner          perms          nsems
0x00000000  98304          apache         600            1
0x00000000  131073         apache         600            1
0x00000000  163842         apache         600            1
0x00000000  196611         apache         600            1
0x00000000  229380         apache         600            1
```

Note almost all of the currently running shared memory segments have a key of 0 (also known as `IPC_PRIVATE`) which means they are only shared between processes in a parent/child relationship. Furthermore, all but one are marked for destruction when there are no further attachments.

One can gain further information about the processes that have created the segments and last attached to them with:

```
$ ipcs -p
```

```

----- Message Queues PIDs -----
msqid      owner      lspid      lrpids

----- Shared Memory Creator/Last-op PIDs -----
shmids     owner      cpids      lpids
0          root      1023      1023
98305      coop      2265      18780
196610     coop      2138      18775
23068675   coop      989       1663
23101444   coop      989       1663
23134213   coop      989       1663
24051718   coop      20573     1663
23756807   coop      10735     1663
24018952   coop      17875     1663
23363593   coop      989       1663
1441811    coop      2048      20573

```

Thus, by doing:

```
$ ps aux |grep -e 20573 -e 2048
```

```

coop      2048  5.3  3.7 1922996 305660 ?        Rl   Oct27  77:07 /usr/bin/gnome-shell
coop      20573 1.9  1.7 807944 141688 ?        Sl   09:56   0:01 /usr/lib64/thunderbird/thunderbird
coop      20710 0.0  0.0 112652  2312 pts/0    S+   09:57   0:00 grep --color=auto -e 20573 -e 2048

```

we see **thunderbird** is using a shared memory segment created by **gnome-shell**.

Perform these steps on your system and identify the various resources being used and by who. Are there any potential **leaks** (shared resources no longer being used by any active processes) on the system? For example, doing:

```
$ ipcs
```

```

.....
----- Shared Memory Segments -----
key      shmids  owner      perms      bytes      nattch      status
.....
0x00000000 622601   coop      600      2097152    2          dest
0x0000001a 13303818 coop      666      8196      0
.....

```

shows a shared memory segment with no attachments and not marked for destruction. Thus it might persist forever, leaking memory if no subsequent process attaches to it.

Chapter 4

Signals



4.1 Labs

Exercise 4.1: Examining Signal Priorities and Execution

We give you a **C** program that includes a signal handler that can handle any signal. The handler avoids making any system calls (such as those that might occur while doing I/O). This file can be extracted from your downloaded SOLUTIONS file as [signals.c](#)



signals.c

```
1  /*
2   * Examining Signal Priorities and Execution.
3   *
4   * The code herein is: Copyright the Linux Foundation, 2014
5   * Author: J. Cooperstein
6   *
7   * This Copyright is retained for the purpose of protecting free
8   * redistribution of source.
9   *
10  * This code is distributed under Version 2 of the GNU General Public
11  * License, which you should have received with the source.
12  *
13  @*/
14
15  #include <stdio.h>
16  #include <unistd.h>
17  #include <signal.h>
18  #include <stdlib.h>
19  #include <string.h>
20  #include <pthread.h>
21
22  #define NUMSIGS 64
23
24  /* prototypes of locally-defined signal handlers */
```



```

25
26 void (sig_handler) (int);
27
28 int sig_count[NUMSIGS + 1];          /* counter for signals received */
29 volatile static int line = 0;
30 volatile int signumbuf[6400], sigcountbuf[6400];
31
32 int main(int argc, char *argv[])
33 {
34     sigset_t sigmask_new, sigmask_old;
35     struct sigaction sigact, oldact;
36     int signum, rc, i;
37     pid_t pid;
38
39     pid = getpid();
40
41     /* block all possible signals */
42     rc = sigfillset(&sigmask_new);
43     rc = sigprocmask(SIG_SETMASK, &sigmask_new, &sigmask_old);
44
45     /* Assign values to members of sigaction structures */
46     memset(&sigact, 0, sizeof(struct sigaction));
47     sigact.sa_handler = sig_handler;      /* we use a pointer to a handler */
48     sigact.sa_flags = 0;                  /* no flags */
49     /* VERY IMPORTANT */
50     sigact.sa_mask = sigmask_new;         /* block signals in the handler itself */
51
52     /*
53      * Now, use sigaction to create references to local signal
54      * handlers * and raise the signal to myself
55      */
56
57     printf
58         ( " \n Installing signal handler and Raising signal for signal number: \n\n " );
59     for (signum = 1; signum <= NUMSIGS; signum++) {
60         if (signum == SIGKILL || signum == SIGSTOP || signum == 32
61             || signum == 33) {
62             printf( " --" );
63             continue;
64         }
65         sigaction(signum, &sigact, &oldact);
66         /* send the signal 3 times! */
67         rc = raise(signum);
68         rc = raise(signum);
69         rc = raise(signum);
70         if (rc) {
71             printf( "Failed on Signal %d \n ", signum);
72         } else {
73             printf( "%4d" , signum);
74             if (signum % 16 == 0)
75                 printf( " \n " );
76         }
77     }
78     fflush(stdout);
79
80     /* restore original mask */
81     rc = sigprocmask(SIG_SETMASK, &sigmask_old, NULL);
82

```



```

83     printf( " \n Signal   Number(Times Processed) \n " );
84     printf( "----- \n " );
85     for (i = 1; i <= NUMSIGS; i++) {
86         printf( "%4d:%3d " , i, sig_count[i]);
87         if (i % 8 == 0)
88             printf( " \n " );
89     }
90     printf( " \n " );
91
92     printf( " \n History: Signal   Number(Count Processed) \n " );
93     printf( "----- \n " );
94     for (i = 0; i < line; i++) {
95         if (i % 8 == 0)
96             printf( " \n " );
97         printf( "%4d(%1d)" , signumbuf[i], sigcountbuf[i]);
98     }
99     printf( " \n " );
100    exit(EXIT_SUCCESS);
101 }
102
103 void sig_handler(int sig)
104 {
105     sig_count[sig]++;
106     signumbuf[line] = sig;
107     sigcountbuf[line] = sig_count[sig];
108     line++;
109 }

```

You will need to compile it and run it as in:

```

$ gcc -o signals signals.c
$ ./signals

```

When run, the program:

- Does not send the signals SIGKILL or SIGSTOP, which can not be handled and will always terminate a program.
- Stores the sequence of signals as they come in, and updates a counter array for each signal that indicates how many times the signal has been handled.
- Begins by suspending processing of all signals and then installs a new set of signal handlers for all signals.
- Sends every possible signal to itself multiple times and then unblocks signal handling and the queued up signal handlers will be called.
- Prints out statistics including:
 - The total number of times each signal was received.
 - The order in which the signals were received, noting each time the total number of times that signal had been received up to that point.

Note the following:

- If more than one of a given signal is **raised** while the process has blocked it, does the process **receive** it multiple times? Does the behavior of **real time** signals differ from normal signals?
- Are all signals received by the process, or are some handled before they reach it?

- What order are the signals received in?

One signal, SIGCONT (18 on **x86**) may not get through; can you figure out why?



Please Note

On some **Linux** distributions signals 32 and 33 can not be blocked and will cause the program to fail. Even though system header files indicate SIGRTMIN=32, the command `kill -1` indicates SIGRTMIN=34.

Note that **POSIX** says one should use signal names, not numbers, which are allowed to be completely implementation dependent.

You should generally avoid sending these signals.

Chapter 5

Package Management Systems



5.1 Labs

Exercise 5.1: Version Control with git

Your system may already have **git** installed. Doing `which git` should show you if it is already present. If not, while you may obtain the source and compile and install it, it is usually easier to install the appropriate pre-compiled binary packages; your instructor can help you identify the needed packages if they are not already installed or cannot be installed with one of the following commands:

```
$ sudo yum install git*
$ sudo zypper install git*
$ sudo apt install git*
```

according to your particular distribution.

Let's get a feel for how **git works** and how easy it is to use. For now we will just make our own local project.

1. First we create a working directory and then initialize **git** to work with it:

```
$ mkdir git-test
$ cd git-test
$ git init
```

2. Initializing the project creates a `.git` directory which will contain all the version control information; the main directories included in the project remain untouched. The initial contents of this directory look like:

```
$ ls -l .git

total 40
drwxrwxr-x 7 coop coop 4096 Dec 30 13:59 ./
drwxrwxr-x 3 coop coop 4096 Dec 30 13:59 ../
drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 branches/
-rw-rw-r-- 1 coop coop  92 Dec 30 13:59 config
-rw-rw-r-- 1 coop coop  58 Dec 30 13:59 description
-rw-rw-r-- 1 coop coop  23 Dec 30 13:59 HEAD
drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 hooks/
drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 info/
drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 objects/
drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 refs/
```

Later we will describe the contents of this directory and its subdirectories; for the most part they start out empty.

3. Next we create a file and add it to the project:

```
$ echo some junk > somejunkfile
$ git add somejunkfile
```

4. We can see the current status of our project with:

```
$ git status

# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   somejunkfile
#
```

Notice it is telling us that our file is **staged** but not yet **committed**.

5. Let's tell **git** who is responsible for this repository:

```
$ git config user.name "Another Genius"
$ git config user.email "b_genius@linux.com"
```

This must be done for each new project unless you have it predefined in a global configuration file.

6. Now let's modify the file, and then see the history of differences:

```
$ echo another line >> somejunkfile
$ git diff

diff --git a/somejunkfile b/somejunkfile
index 9638122..6023331 100644
--- a/somejunkfile
+++ b/somejunkfile
@@ -1,2 @@
 some junk
+another line
```

7. To actually commit the changes to the repository we do:

```
$ git commit -m "My initial commit"

Created initial commit eafad66: My initial commit
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 somejunkfile
```

If you do not specify an identifying message to accompany the commit with the `-m` option you will jump into an editor to put some content in. You **must** do this or the commit will be rejected. The editor chosen will be what is set in your `EDITOR` environment variable, which can be superseded with setting `GIT_EDITOR`.

8. You can see your history with:

```
$ git log

commit eafad66304ebbcd6acfe69843d246de3d8f6b9cc
Author: A Genius <a_genius@linux.com>
Date:   Wed Dec 30 11:07:19 2009 -0600

    My initial commit
```

and you can see the information got in there. You will note the long hexadecimal string which is the **commit number**; it is a 160-bit, 40-digit unique identifier. **git** cares about these beasts, not file names.

9. You are now free to modify the already existing file and add new files with `git add`. But they are staged until you do another `git commit`

10. Now that was not so bad. But we have only scratched the surface.

Chapter 6

RPM



6.1 Labs

Exercise 6.1: Using RPM



Please Note

To do these labs you need to have access to a system that is **RPM**-based, such as **RHEL**, **CentOS**, **Fedora**, **SUSE**, or **openSUSE**.

Here we will just do a number of simple operations for querying and verifying **rpm** packages.

This lab will work equally well on **Red Hat** and **SUSE**-based systems.

1. Find out what package the file `/etc/logrotate.conf` belongs to.
2. List information about the package including all the files it contains.
3. Verify the package installation.
4. Try to remove the package.

Solution 6.1

1.

```
$ rpm -qf /etc/logrotate.conf
```



```
logrotate-3.8.6-4.el7.x86_64
```
2.

```
$ rpm -qil logrotate
```



```
...
```

Note a fancier form that combines these two steps would be:

```
$ rpm -qil $(rpm -qf /etc/logrotate.conf)
```

3.

```
$ rpm -V logrotate
```

```
..?...... /etc/cron.daily/logrotate
S.5....T. c /etc/logrotate.conf
```



On RedHat

4. On RHEL 7:

```
$ sudo rpm -e logrotate

error: Failed dependencies:
logrotate is needed by (installed) vsftpd-3.0.2-9.el7.x86_64
logrotate >= 3.5.2 is needed by (installed) rsyslog-7.4.7-7.el7_0.x86_64
```



On openSUSE

On OpenSUSE-Leap 42.1:

```
$ sudo rpm -e logrotate

$ sudo rpm -e logrotate
error: Failed dependencies:
logrotate is needed by (installed) wpa_supplicant-2.2-6.2.x86_64
logrotate is needed by (installed) mcelog-1.20-3.2.x86_64
logrotate is needed by (installed) iscsiui-0.7.8.2-40.1.x86_64
logrotate is needed by (installed) xdm-1.1.11-10.1.x86_64
logrotate is needed by (installed) libvirt-daemon-1.2.18.4-14.2.x86_64
logrotate is needed by (installed) apache2-2.4.16-15.1.x86_64
```

Note that the exact package dependency tree depends on both the distribution and choice of installed software.

✍ Exercise 6.2: Rebuilding the RPM Database



Please Note

To do these labs you need to have access to a system that is **RPM**-based, such as **RHEL**, **CentOS**, **Fedora**, **SUSE**, or **openSUSE**.

There are conditions under which the **RPM** database stored in `/var/lib/rpm` can be corrupted. In this exercise we will construct a new one and verify its integrity.

This lab will work equally well on **Red Hat** and **SUSE**-based systems.

1. Backup the contents of `/var/lib/rpm` as the rebuild process will overwrite the contents. If you neglect to do this and something goes wrong you are in serious trouble.
2. Rebuild the data base.
3. Compare the new contents of the directory with the backed up contents; don't examine the actual file contents as they are binary data, but note the number and names of the files.
4. Get a listing of all **rpms** on the system. You may want to compare this list with one generated before you actually do the rebuild procedure. If the query command worked, your new database files should be fine.
5. Compare again the two directory contents. Do they have the same files now?
6. You could delete the backup (probably about 100 MB in size) but you may want to keep it around for a while to make sure your system is behaving properly before trashing it.

✔ Solution 6.2

1. `$ cd /var/lib`
 `$ sudo cp -a rpm rpm_BACKUP`
2. `$ sudo rpm --rebuilddb`
3. `$ ls -l rpm rpm_BACKUP`
4. `$ rpm -qa | tee /tmp/rpm-qa.output`
5. `$ ls -l rpm rpm_BACKUP`
6. Probably you should not do this until you are sure the system is fine!
 `$ sudo rm -rf rpm_BACKUP`

Chapter 7

dpkg



7.1 Labs

Exercise 7.1: Using dpkg



Please Note

To do these labs you need to have access to a system that is **Debian**-based, such as **Debian**, **Ubuntu**, or **Linux Mint**.

Here we will just do a number of simple operations for querying and verifying **Debian** packages.

1. Find out what package the file `/etc/logrotate.conf` belongs to.
2. List information about the package including all the files it contains.
3. Verify the package installation.
4. Try to remove the package.

Solution 7.1

1.

```
$ dpkg -S /etc/logrotate.conf
logrotate: /etc/logrotate.conf
```
2.

```
$ dpkg -L logrotate
...
```
3.

```
$ dpkg -V logrotate
```
4.

```
$ sudo dpkg -r logrotate
dpkg: dependency problems prevent removal of logrotate:
 libvirt-bin depends on logrotate.
 ubuntu-standard depends on logrotate.
```

```
dpkg: error processing package logrotate (--remove):  
dependency problems - not removing  
Errors were encountered while processing:  
logrotate
```

Chapter 8

yum



8.1 Labs

Exercise 8.1: Basic YUM Commands



Please Note

To do these labs you need to have access to a system that is **yum**-based, such as **RHEL**, **CentOS**, or **Fedora**.

1. Check to see if there are any available updates for your system.
2. Update a particular package.
3. List all installed kernel-related packages, and list all installed or available ones.
4. Install the **httpd-devel** package, or anything else you might not have installed yet. Doing a simple:

```
$ sudo yum list
```

will let you see a complete list; you may want to give a wildcard argument to narrow the list.

Solution 8.1

1.

```
$ sudo yum update
```

```
$ sudo yum check-update
```

```
$ sudo yum list updates
```

Only the first form will try to do the installations.

2.

```
$ sudo yum update bash
```
3.

```
$ sudo yum list installed "kernel*"
```

```
$ sudo yum list "kernel*"
```
4.

```
$ sudo yum install httpd-devel
```

Exercise 8.2: Using yum to Find Information About a Package

**Please Note**

To do these labs you need to have access to a system that is **yum**-based, such as **RHEL**, **CentOS**, or **Fedora**.

Using **yum** (and not **rpm** directly), find:

1. All packages that contain a reference to **bash** in their name or description.
2. Installed and available **bash** packages.
3. The package information for **bash**.
4. The dependencies for the **bash** package.

Try the commands you used above both as **root** and as a regular user. Do you notice any difference?

Solution 8.2**On RedHat**

Note: on **RHEL 7** you may get some permission errors if you don't use **sudo** with the following commands, even though we are just getting information.

1. `$ sudo yum search bash`
2. `$ sudo yum list bash`
3. `$ sudo yum info bash`
4. `$ sudo yum deplist bash`

However, all the commands above should work for both regular users and the **root** user.

Exercise 8.3: Managing Groups of Packages with yum**Please Note**

To do these labs you need to have access to a system that is **yum**-based, such as **RHEL**, **CentOS**, or **Fedora**.

**Please Note**

On **RHEL 7** you may get some permission errors if you don't use **sudo** with some of the following commands, even when we are just getting information.

yum provides the ability to manage groups of packages.

1. Use the following command to list all package groups available on your system:
`$ yum grouplist`
2. Identify the Backup Client group and generate the information about this group using the command
`$ yum groupinfo "Backup Client"`
3. Install using:
`$ sudo yum groupinstall "Backup Client"`

4. Identify a package group that's currently installed on your system and that you don't need. Remove it using `yum groupremove` as in:

```
$ sudo yum groupremove "Backup Client"
```

Note you will be prompted to confirm removal so you can safely type the command to see how it works.

You may find that the `groupremove` does **not** remove everything that was installed; whether this is a bug or a feature can be discussed.

Exercise 8.4: Adding a New yum Repository



Please Note

To do these labs you need to have access to a system that is **yum**-based, such as **RHEL**, **CentOS**, or **Fedora**.

According to its authors (at <http://www.webmin.com/index.htm>):

“**Webmin** is a web-based interface for system administration for Unix. Using any modern web browser, you can setup user accounts, Apache, DNS, file sharing and much more. Webmin removes the need to manually edit Unix configuration files like `/etc/passwd`, and lets you manage a system from the console or remotely.”

We are going to create a repository for installation and upgrade. While we could simply go the download page and get the current **rpm**, that would not automatically give us any upgrades.

1. Create a new repository file called `webmin.repo` in the `/etc/yum.repos.d` directory. It should contain the following:



webmin.repo

```
[Webmin]
name=Webmin Distribution Neutral
baseurl=http://download.webmin.com/download/yum
mirrorlist=http://download.webmin.com/download/yum/mirrorlist
enabled=1
gpgcheck=0
```

(Note you can also cut and paste the contents from <http://www.webmin.com/download.html>.)

2. Install the webmin package.

```
$ sudo yum install webmin
```


Chapter 9

zypper



9.1 Labs

Exercise 9.1: Basic zypper Commands



Please Note

To do these labs you need to have access to a system that is **zypper**-based, such as **SUSE**, or **openSUSE**.

1. Check to see if there are any available updates for your system.
2. Update a particular package.
3. List all repositories the system is aware of, enabled or not.
4. List all installed kernel-related packages, and list all installed or available ones.
5. Install the **apache2-devel** package, or anything else you might not have installed yet. (Note **httpd** is **apache2** on **SUSE** systems.) Doing a simple:

```
$ sudo zypper search
```

will let you see a complete list; you may want to give a wildcard argument to narrow the list.

Solution 9.1

1.

```
$ zypper list-updates
```
2.

```
$ sudo zypper update bash
```
3.

```
$ zypper repos
```
4.

```
$ zypper search -i kernel
$ zypper search kernel
```
5.

```
$ sudo zypper install apache2-devel
```

Exercise 9.2: Using zypper to Find Information About a Package



Please Note

To do these labs you need to have access to a system that is **zypper**-based, such as **SUSE**, or **openSUSE**.

Using **zypper** (and not **rpm** directly), find:

1. All packages that contain a reference to **bash** in their name or description.
2. Installed and available **bash** packages.
3. The package information for **bash**.
4. The dependencies for the **bash** package.

Try the commands you used above both as **root** and as a regular user. Do you notice any difference?

Solution 9.2

1. `$ zypper search -d bash`

Without the `-d` option only packages with **bash** in their actual name are reported. You may have to do `zypper info` on the package to see where **bash** is mentioned.

2. `$ zypper search bash`

3. `$ zypper info bash`

4. `$ zypper info--requires bash`

will give a list of files **bash** requires. Perhaps the easiest way to see what depends on having **bash** installed is to do

```
$ sudo zypper remove --dry-run bash
```

For this exercise **bash** is a bad choice since it is so integral to the system; you really can't remove it anyway.

Chapter 10

APT



10.1 Labs

Exercise 10.1: Basic APT Commands



Please Note

To do these labs you need to have access to a system that is **Debian**-based, such as **Debian**, **Ubuntu**, or **Linux Mint**.

1. Check to see if there are any available updates for your system.
2. List all installed kernel-related packages, and list all installed or available ones.
3. Install the **apache2-dev** package, or anything else you might not have installed yet. Doing a simple:

```
$ apt-cache pkgnames
```

will let you see a complete list; you may want to give a wildcard argument to narrow the list.

Solution 10.1

1. First synchronize the package index files with remote repositories:

```
$ sudo apt update
```

To actually upgrade:

```
$ sudo apt upgrade
```

```
$ sudo apt -u upgrade
```

(You can also use `dist-upgrade` as discussed earlier.) Only the first form will try to do the installations.

2.

```
$ apt-cache search "kernel"
```

```
$ apt-cache search -n "kernel"
```

```
$ apt-cache pkgnames "kernel"
```

The second and third forms only find packages that have `kernel` in their name.

```
$ dpkg --get-selections "*kernel*"
```

to get only installed packages. Note that on **Debian**-based systems you probably should use `linux` not `kernel` for kernel-related packages as they don't usually have `kernel` in their name.

3. `$ sudo apt install apache2-dev`

Exercise 10.2: Using APT to Find Information About a Package



Please Note

To do these labs you need to have access to a system that is **Debian**-based, such as **Debian**, **Ubuntu**, or **Linux Mint**.

Using **apt-cache** and **apt** (and not **dpkg**), find:

1. All packages that contain a reference to **bash** in their name or description.
2. Installed and available **bash** packages.
3. The package information for **bash**.
4. The dependencies for the **bash** package.

Try the commands you used above both as `root` and as a regular user. Do you notice any difference?

Solution 10.2

1. `$ apt-cache search bash`
2. `$ apt-cache search -n bash`
3. `$ apt-cache show bash`
4. `$ apt-cache depends bash`
`$ apt-cache rdepends bash`

Exercise 10.3: Managing Groups of Packages with APT



Please Note

To do these labs you need to have access to a system that is **Debian**-based, such as **Debian**, **Ubuntu**, or **Linux Mint**.

APT provides the ability to manage groups of packages, similarly to the way **yum** does it, through the use of **metapackages**. These can be thought of as **virtual packages**, that collect related packages that must be installed and removed as a group.

To get a list of available **metapackages**:

```
$ apt-cache search metapackage

bacula - network backup service - metapackage
bacula-client - network backup service - client metapackage
bacula-server - network backup service - server metapackage
cloud-utils - metapackage for installation of upstream cloud-utils source
compiz - OpenGL window and compositing manager
emacs - GNU Emacs editor (metapackage)
....
```

You can then easily install them like regular single packages, as in:

```
$ sudo apt install bacula-client
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  bacula-common bacula-console bacula-fd bacula-traymonitor
Suggested packages:
  bacula-doc kde gnome-desktop-environment
The following NEW packages will be installed:
  bacula-client bacula-common bacula-console bacula-fd bacula-traymonitor
0 upgraded, 5 newly installed, 0 to remove and 0 not upgraded.
Need to get 742 kB of archives.
After this operation, 1,965 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Select an uninstalled metapackage and then remove it.

Chapter 11

System Monitoring



11.1 Labs

Exercise 11.1: Using stress

stress is a **C** language program written by Amos Waterland at the University of Oklahoma, licensed under the **GPL v2**. It is designed to place a configurable amount of stress by generating various kinds of workloads on the system.

If you are lucky you can install **stress** directly from your distribution's packaging system. Otherwise, you can obtain the source from <http://people.seas.harvard.edu/~apw/stress>, and then compile and install by doing:

```
$ tar zxvf stress-1.0.4.tar.gz
$ cd stress-1.0.4
$ ./configure
$ make
$ sudo make install
```

There may exist pre-packaged downloadable binaries in the **.deb** and **.rpm** formats; see the home page for details and locations.

Once installed, you can do:

```
$ stress --help
```

for a quick list of options, or

```
$ info stress
```

for more detailed documentation.

As an example, the command:

```
$ stress -c 8 -i 4 -m 6 -t 20s
```

will:

- Fork off 8 CPU-intensive processes, each spinning on a `sqrt()` calculation.
- Fork off 4 I/O-intensive processes, each spinning on `sync()`.

- Fork off 6 memory-intensive processes, each spinning on `malloc()`, allocating 256 MB by default. The size can be changed as in `--vm-bytes 128M`.
- Run the stress test for 20 seconds.

After installing **stress**, you may want to start up your system's graphical system monitor, which you can find on your application menu, or run from the command line, which is probably **gnome-system-monitor** or **ksysguard**.

Now begin to put stress on the system. The exact numbers you use will depend on your system's resources, such as the number of CPU's and **RAM** size.

For example, doing

```
$ stress -m 4 -t 20s
```

puts only a memory stressor on the system.

Play with combinations of the switches and see how they impact each other. You may find the **stress** program useful to simulate various high load conditions.

Chapter 12

Process Monitoring



12.1 Labs

Exercise 12.1: Processes

1. Run **ps** with the options **-ef**. Then run it again with the options **aux**. Note the differences in the output.
2. Run **ps** so that only the process ID, priority, nice value, and the process command line are displayed.
3. Start a new **bash** session by typing **bash** at the command line. Start another **bash** session using the **nice** command but this time giving it a nice value of 10.
4. Run **ps** as in step 2 to note the differences in priority and nice values. Note the process ID of the two **bash** sessions.
5. Change the nice value of one of the **bash** sessions to 15 using **renice**. Once again, observe the change in priority and nice values.
6. Run **top** and watch the output as it changes. Hit **q** to stop the program.

Solution 12.1

1.

```
$ ps -ef
$ ps aux
```
2.

```
$ ps -o pid,pri,ni,cmd
    PID PRI  NI CMD
    2389  19   0 bash
    22079  19   0 ps -o pid,pri,ni,cmd
```

(Note: There should be no spaces between parameters.)
3.

```
$ bash
$ nice -n 10 bash
$ ps -o pid,pri,ni,cmd
    2389  19   0 bash
    22115  19   0 bash
    22171   9  10 bash
    22227   9  10 ps -o pid,pri,ni,cmd
```

4.

```
$ renice 15 -p 22171
```



```
$ ps -o pid,pri,ni,cmd
```

| PID | PRI | NI | CMD |
|-------|-----|----|----------------------|
| 2389 | 19 | 0 | bash |
| 22115 | 19 | 0 | bash |
| 22171 | 4 | 15 | bash |
| 22246 | 4 | 15 | ps -o pid,pri,ni,cmd |
5.

```
$ top
```

Exercise 12.2: Monitoring Process States

1. Use **dd** to start a background process which reads from `/dev/urandom` and writes to `/dev/null`.
2. Check the process state. What should it be?
3. Bring the process to the foreground using the **fg** command. Then hit **Ctrl-Z**. What does this do? Look at the process state again, what is it?
4. Run the **jobs** program. What does it tell you?
5. Bring the job back to the foreground, then terminate it using **kill** from another window.

Solution 12.2

1.

```
$ dd if=/dev/urandom of=/dev/null &
```
2.

```
$ ps -C dd -o pid,cmd,stat
```

```
25899 dd if=/dev/urandom of=/dev/ R
```

Should be S or R.
3.

```
$ fg
```



```
$ ^Z
```



```
$ ps -C dd -o pid,cmd,stat
```

| PID | CMD | STAT |
|-------|-----------------------------|------|
| 25899 | dd if=/dev/urandom of=/dev/ | T |

State should be T.
4. Type the **jobs** command. What does it tell you?

```
$ jobs
```

```
[1]+  Stopped                  dd if=/dev/urandom of=/dev/null
```
5. Bring the job back to the foreground, then kill it using the **kill** command from another window.

```
$ fg
```



```
$ kill 25899
```

Chapter 13

Memory Monitoring and Usage



13.1 Labs

Exercise 13.1: Invoking the OOM Killer

Examine what swap partitions and files are present on your system by examining `/proc/swaps`.

Turn off all swap with the command

```
$ sudo /sbin/swapoff -a
```

Make sure you turn it back on later, when we are done, with

```
$ sudo /sbin/swapon -a
```

Now we are going to put the system under increasing memory pressure. One way to do this is to exploit the **stress** program we installed earlier, running it with arguments such as:

```
$ stress -m 8 -t 10s
```

which would keep 2 GB busy for 10 seconds.

You should see the **OOM** (Out of Memory) killer swoop in and try to kill processes in a struggle to stay alive. You can see what is going on by running **dmesg** or monitoring `/var/log/messages` or `/var/log/syslog`, or through graphical interfaces that expose the system logs.

Who gets clobbered first?

Chapter 14

I/O Monitoring and Tuning



14.1 Labs

Exercise 14.1: bonnie++

bonnie++ is a widely available benchmarking program that tests and measures the performance of drives and filesystems. It is descended from **bonnie**, an earlier implementation.

Results can be read from the terminal window or directed to a file, and also to a **csv** format (**c**omma **s**eparated **v**alue). Companion programs, **bon.csv2html** and **bon.csv2txt**, can be used convert to html and plain text output formats.

We recommend you read the **man** page for **bonnie++** before using as it has quite a few options regarding which tests to perform and how exhaustive and stressful they should be. A quick synopsis is obtained with:

```
$ bonnie++ --help

bonnie++: invalid option -- 'h'
usage:
bonnie++ [-d scratch-dir] [-c concurrency] [-s size(MiB)[:chunk-size(b)]]
        [-n number-to-stat[:max-size[:min-size][:num-directories[:chunk-size]]]]
        [-m machine-name] [-r ram-size-in-MiB]
        [-x number-of-tests] [-u uid-to-use:gid-to-use] [-g gid-to-use]
        [-q] [-f] [-b] [-p processes | -y] [-z seed | -Z random-file]
        [-D]
```

Version: 1.96

A quick test can be obtained with a command like:

```
$ time sudo bonnie++ -n 0 -u 0 -r 100 -f -b -d /mnt
```

where:

- **-n 0** means don't perform the file creation tests.
- **-u 0** means run as root.
- **-r 100** means pretend you have 100 MB of RAM.
- **-f** means skip per character I/O tests.

The **fs.mark** benchmark gives a low level bashing to file systems, using heavily asynchronous I/O across multiple directories and drives. It's a rather old program written by Rick Wheeler that has stood the test of time.

It can be downloaded from <http://sourceforge.net/projects/fsmark/> Once you have obtained the tarball, you can unpack it and compile it with:

```
$ tar zxvf fs_mark-3.3.tgz
$ cd fs_mark
$ make
```

Read the README file as we are only going to touch the surface.

If the compile fails with an error like:

```
$ make
....
/usr/bin/ld: cannot find -lc
```

it is because you haven't installed the **static** version of **glibc**. You can do this on **Red Hat**-based systems by doing:

```
$ sudo yum install glibc-static
```

and on **SUSE**-related systems with:

```
$ sudo zypper install glibc-devel-static
```

On **Debian**-based systems the relevant static library is installed along with the shared one so no additional package needs to be sought.

For a test we are going to create 1000 files, each 10 KB in size, and after each write we'll perform an **fsync** to flush out to disk. This can be done in the **/tmp** directory with the command:

```
$ fs_mark -d /tmp -n 1000 -s 10240
```

While this is running, gather extended **iostat** statistics with:

```
$ iostat -x -d /dev/sda 2 20
```

in another terminal window.

The numbers you should surely note are the number of files per second reported by **fs.mark** and the percentage of CPU time utilized reported by **iostat**. If this is approaching 100 percent, you are I/O-bound.

Depending on what kind of filesystem you are using you may be able to get improved results by changing the **mount** options. For example, for **ext3** or **ext4** you can try:

```
$ mount -o remount,barrier=1 /tmp
```

or for **ext4** you can try:

```
$ mount -o remount,journal_async_commit /tmp
```

See how your results change.

Note that these options may cause problems if you have a power failure, or other ungraceful system shutdown; i.e., there is likely to be a trade-off between stability and speed.

Documentation about some of the **mount** options can be found with the kernel source under [Documentation/filesystems](#) and the **man** page for **mount**.

Chapter 15

I/O Scheduling **



15.1 Labs

Exercise 15.1: Comparing I/O Schedulers

We provide a script which is to be used to compare I/O schedulers which can be extracted from your downloaded SOLUTIONS file as [lab_iosched.sh](#).



lab_iosched.sh

```
#!/bin/bash

NMAX=8
NMEGS=100
[[ -n $1 ]] && NMAX=$1
[[ -n $2 ]] && NMEGS=$2

echo Doing: $NMAX parallel read/writes on: $NMEGS MB size files

TIMEFORMAT= "%R %U %S"

#####
# simple test of parallel reads
do_read_test(){
    for n in $(seq 1 $NMAX) ; do
        cat file$n > /dev/null &
    done
    # wait for previous jobs to finish
    wait
}

# simple test of parallel writes
do_write_test(){
    for n in $(seq 1 $NMAX) ; do
        [[ -f fileout$n ]] && rm -f fileout$n
        (cp file1 fileout$n && sync) &
    done
}
```

SH

```

done
# wait for previous jobs to finish
wait
}

# create some files for reading, ok if they are the same
create_input_files(){
    [[ -f file1 ]] || dd if=/dev/urandom of=file1 bs=1M count=$NMEGS
    for n in $(seq 1 $NMAX) ; do
        [[ -f file$n ]] || cp file1 file$n
    done
}

echo -e "\ncreating as needed random input files"
create_input_files

#####
# begin the actual work

# do parallel read test
echo -e "\ndoing timings of parallel reads\n"
echo -e " REAL    USER    SYS\n"
for iosched in noop deadline cfq ; do
    echo testing IOSCHED = $iosched
    echo $iosched > /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
#    echo -e "\nclearing the memory caches\n"
    echo 3 > /proc/sys/vm/drop_caches
    time do_read_test
done
#####
# do parallel write test
echo -e "\ndoing timings of parallel writes\n"
echo -e " REAL    USER    SYS\n"
for iosched in noop deadline cfq ; do
    echo testing IOSCHED = $iosched
    echo $iosched > /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    time do_write_test
done
#####

```

If you are taking the online self-paced version of this course, the script is available for download from your **Lab** screen.

Remember to make it executable by doing: by doing:

```
$ chmod +x ioscript.sh
```

The following explains how the script was written and how to use it.

The script should:

- Cycle through the available I/O schedulers on a hard disk while doing a configurable number of parallel reads and writes of files of a configurable size.
- Test reads and writes as separate steps.
- When testing reads make sure you're actually reading from disk and not from cached pages of memory; you can flush out the cache by doing:

```
$ echo 3 > /proc/sys/vm/drop_caches
```

before doing the reads. You can **cat** into `/dev/null` to avoid writing to disk.

- Make sure all reads are complete before obtaining timing information; this can be done by issuing a **wait** command under the shell.
- Test writes by simply copying a file (which will be in cached memory after the first read) multiple times simultaneously. To make sure you wait for all writes to complete before you get timing information you can issue a **sync** call.

The provided script takes two arguments. The first is the number of simultaneous reads and writes to perform. The second is the size (in MB) of each file.

This script must be run as root as it echoes values into the `/proc` and `/sys` directory trees.

Compare the results you obtain using different I/O schedulers.

Extra Credit

For additional exploring you might try changing some of the tunable parameters and see how results vary.

Chapter 16

Linux Filesystems and the VFS



16.1 Labs

Exercise 16.1: The **tmpfs** Special Filesystem

tmpfs is one of many special filesystems used under **Linux**. Some of these are not really used as filesystems, but just take advantage of the filesystem abstraction. However, **tmpfs** is a real filesystem that applications can do I/O on.

Essentially, **tmpfs** functions as a **ramdisk**; it resides purely in memory. But it has some nice properties that old-fashioned conventional ramdisk implementations did not have:

1. The filesystem adjusts its size (and thus the memory that is used) dynamically; it starts at zero and expands as necessary up to the maximum size it was mounted with.
2. If your RAM gets exhausted, **tmpfs** can utilize swap space. (You still can't try to put more in the filesystem than its maximum capacity allows, however.)
3. **tmpfs** does not require having a normal filesystem placed in it, such as **ext3** or **vfat**; it has its own methods for dealing with files and I/O that are aware that it is really just space in memory (it is not actually a block device), and as such are optimized for speed.

Thus there is no need to pre-format the filesystem with a **mkfs** command; you merely just have to mount it and use it.

Mount a new instance of **tmpfs** anywhere on your directory structure with a command like:

```
$ sudo mkdir /mnt/tmpfs
$ sudo mount -t tmpfs none /mnt/tmpfs
```

See how much space the filesystem has been given and how much it is using:

```
$ df -h /mnt/tmpfs
```

You should see it has been allotted a default value of half of your RAM; however, the usage is zero, and will only start to grow as you place files on **/mnt/tmpfs**.

You could change the allotted size as a mount option as in:

```
$ sudo mount -t tmpfs -o size=1G none /mnt/tmpfs
```

You might try filling it up until you reach full capacity and see what happens. Do not forget to unmount when you are done with:

```
$ sudo umount /mnt/tmpfs
```

Virtually all modern **Linux** distributions mount an instance of **tmpfs** at `/dev/shm`:

```
$ df -h /dev/shm
```

```
Filesystem      Type  Size  Used Avail Use% Mounted on
tmpfs           tmpfs 3.9G   24M   3.9G   1% /dev/shm
```

Many applications use this such as when they are using **POSIX** shared memory as an inter-process communication mechanism. Any user can create, read and write files in `/dev/shm`, so it is a good place to create temporary files in memory.

Create some files in `/dev/shm` and note how the filesystem is filling up with **df**.

In addition, many distributions mount multiple instances of **tmpfs**; for example, on a **RHEL 7** system:

```
$ df -h | grep tmpfs
```

```
devtmpfs          devtmpfs 3.9G     0 3.9G    0% /dev
tmpfs             tmpfs    3.9G   24M 3.9G    1% /dev/shm
tmpfs             tmpfs    3.9G   9.2M 3.9G    1% /run
tmpfs             tmpfs    3.9G     0 3.9G    0% /sys/fs/cgroup
/tmp/vmware-coop/564d9ea7-8e8e-29c0-2682-e5d3de3a51d8 tmpfs    3.3G     0 3.3G    0% /tmp/vmware-coop/
564d9ea7-8e8e-29c0-2682-e5d3de3a51d8
/tmp/vmware-coop/564d7668-ec55-ee45-f33e-c8e97e956190 tmpfs    2.3G   2.0G 256M   89% /tmp/vmware-coop/
564d7668-ec55-ee45-f33e-c8e97e956190
none             tmpfs    1.0G   1.0G     0 100% /tmp/ohno
```

Notice this was run on a system with 8 GB of ram, so clearly you can't have all these **tmpfs** filesystems actually using the 4 GB they have each been allotted!



Please Note

Some distributions (such as **Fedora**) may (by default) mount `/tmp` as a **tmpfs** system; in such cases one has to avoid putting large files in `/tmp` to avoid running out of memory. Or one can disable this behavior as we discussed earlier when describing `/tmp`.

Chapter 17

Disk Partitioning



17.1 Labs

Exercise 17.1: Using a File as a Disk Partition Image

The exercises in this section will make simple use of **mkfs** for formatting filesystems, and **mount** for mounting them at places in the root filesystem tree. These commands will be explained in detail in the next session.

For the purposes of the exercises in this course you will need unpartitioned disk space. It need not be large, certainly one or two GB will suffice.

If you are using your own native machine, you either have it or you do not. If you do not, you will have shrink a partition and the filesystem on it (first!) and then make it available, using **gparted** and/or the steps we have outlined or will outline.

Or you can use the **loop device** mechanism with or without the **parted** program, as we will do in the first two exercises in this section.



Please Note

If you have real physical unpartitioned disk space you do not **need** to do the following procedures, but it is still a very useful learning exercise.

We are going to create a file that will be used as a container for a full hard disk partition image, and for all intents and purposes can be used like a real hard partition. In the next exercise we will show how to put more than one partition on it and have it behave as an entire disk.

1. Create a file full of zeros 1 GB in length:

```
$ dd if=/dev/zero of=imagefile bs=1M count=1024
```

You can make a much smaller file if you like or do not have that much available space in the partition you are creating the file on.

2. Put a filesystem on it:

```
$ mkfs.ext4 imagefile
```

```
mke2fs 1.42.9 (28-Dec-2013)
```

```
imagefile is not a block special device.
Proceed anyway? (y,n) y
Discarding device blocks: done
.....
```

Of course you can format with a different filesystem, doing **mkfs.ext3**, **mkfs.vfat**, **mkfs.xfs** etc.

3. Mount it somewhere:

```
$ mkdir mntpoint
$ sudo mount -o loop imagefile mntpoint
```

You can now use this to your heart's content, putting files etc. on it.

4. When you are done unmount it with:

```
$ sudo umount mntpoint
```

An alternative method to using the `loop` option to mount would be:

```
$ sudo losetup /dev/loop2 imagefile
$ sudo mount /dev/loop2 mntpoint
....
$ sudo umount mntpoint
$ sudo losetup -d /dev/loop2
```

We will discuss **losetup** in a subsequent exercise, and you can use `/dev/loop[0-7]` but you have to be careful they are not already in use, as we will explain.

You should note that using a loop device file instead of a real partition can be useful, but it is pretty worthless for doing any kind of measurements or benchmarking. This is because you are placing one filesystem layer on top of another, which can only have a negative effect on performance, and mostly you just use the behavior of the underlying filesystem the image file is created on.

Exercise 17.2: Partitioning a Disk Image File

The next level of complication is to divide the container file into multiple partitions, each of which can be used to hold a filesystem, or a swap area.

You can reuse the image file created in the previous exercise or create a new one.

1. Run **fdisk** on your imagefile:

```
$ sudo fdisk -C 130 imagefile

Device does not contain a recognized partition table
Building a new DOS disk label with disk identifier 0x6280ced3.
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help):
```

The `-C 130` sets the number of phony cylinders in the drive, and is only necessary in old versions of **fdisk**, which unfortunately you will find on **RHEL 6**. However, it will do no harm on other distributions.

2. Type `m` to get a list of commands:

```
m

Command (m for help): m
Command action
a   toggle a bootable flag
b   edit bsd disklabel
c   toggle the dos compatibility flag
d   delete a partition
g   create a new empty GPT partition table
```



```

G   create an IRIX (SGI) partition table
l   list known partition types
m   print this menu
n   add a new partition
o   create a new empty DOS partition table
p   print the partition table
q   quit without saving changes
s   create a new empty Sun disklabel
t   change a partition's system id
u   change display/entry units
v   verify the partition table
w   write table to disk and exit
x   extra functionality (experts only)

```

Command (m for help):

3. Create a new primary partition and make it 256 MB (or whatever size you would like:

Command (m for help): n

Partition type:

```

p   primary (0 primary, 0 extended, 4 free)
e   extended

```

Select (default p): p

Partition number (1-4, default 1): 1

First sector (2048-2097151, default 2048):

Using default value 2048

Last sector, +sectors or +sizeK,M,G (2048-2097151, default 2097151): +256M

Partition 1 of type Linux and of size 256 MiB is set

4. Add a second primary partition also of 256 MB in size:

Command (m for help): n

Partition type:

```

p   primary (1 primary, 0 extended, 3 free)
e   extended

```

Select (default p): p

Partition number (2-4, default 2): 2

First sector (526336-2097151, default 526336):

Using default value 526336

Last sector, +sectors or +sizeK,M,G (526336-2097151, default 2097151): +256M

Partition 2 of type Linux and of size 256 MiB is set

Command (m for help): p

Disk imagefile: 1073 MB, 1073741824 bytes, 2097152 sectors

Units = sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk label type: dos

Disk identifier: 0x6280ced3

| Device | Boot | StartEnd | Blocks | Id | System |
|------------|------|----------|---------|--------|----------|
| imagefile1 | | 2048 | 526335 | 262144 | 83 Linux |
| imagefile2 | | 526336 | 1050623 | 262144 | 83 Linux |

5. Write the partition table to disk and exit:

Command (m for help): w

The partition table has been altered!

Syncing disks.

While this has given us some good practice, we haven't yet seen a way to use the two partitions we just created. We'll start over in the next exercise with a method that lets us do so.

Exercise 17.3: Using losetup and parted

We are going to experiment more with:

- Loop devices and **losetup**
- **parted** to partition at the command line non-interactively.

We expect that you should read the **man pages** for **losetup** and **parted** before doing the following procedures.

Once again, you can reuse the image file or, better still, zero it out and start freshly or with another file.

1. Associate the image file with a **loop** device:

```
$ sudo losetup -f
/dev/loop1
$ sudo losetup /dev/loop1 imagefile
```

where the first command finds the first **free** loop device. The reason to do this is you may already be using one or more loop devices. For example, on the system that this is being written on, before the above command is executed:

```
$ losetup -a
/dev/loop0: []: (/usr/src/KERNELS.sqfs)
```

a **squashfs** compressed, read-only filesystem is already mounted using `/dev/loop0`. (The output of this command will vary with distribution.) If we were to ignore this and use **losetup** on `/dev/loop0` we would almost definitely corrupt the file.

2. Create a disk partition label on the loop device (image file):

```
$ sudo parted -s /dev/loop1 mklabel msdos
```

3. Create three primary partitions on the loop device:

```
$ sudo parted -s /dev/loop1 unit MB mkpart primary ext4 0 256
$ sudo parted -s /dev/loop1 unit MB mkpart primary ext4 256 512
$ sudo parted -s /dev/loop1 unit MB mkpart primary ext4 512 1024
```

4. Check the partition table:

```
$ fdisk -l /dev/loop1

Disk /dev/loop1: 1073 MB, 1073741824 bytes, 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x00050c11
```

| Device | Boot | Start | End | Blocks | Id | System |
|--------------|------|---------|---------|--------|----|--------|
| /dev/loop1p1 | | 1 | 500000 | 250000 | 83 | Linux |
| /dev/loop1p2 | | 500001 | 1000000 | 250000 | 83 | Linux |
| /dev/loop1p3 | | 1000001 | 2000000 | 500000 | 83 | Linux |

5. What happens next depends on what distribution you are on. For example, on **RHEL 7** and **Ubuntu 16.04** you will find new device nodes have been created:

```
$ ls -l /dev/loop1*

brw-rw---- 1 root disk  7, 1 Oct  7 14:54 /dev/loop1
brw-rw---- 1 root disk 259, 0 Oct  7 14:54 /dev/loop1p1
brw-rw---- 1 root disk 259, 3 Oct  7 14:54 /dev/loop1p2
brw-rw---- 1 root disk 259, 4 Oct  7 14:54 /dev/loop1p3
```

and we will use them in the following.

6. Put filesystems on the partitions:

```
$ sudo mkfs.ext3 /dev/loop1p1
$ sudo mkfs.ext4 /dev/loop1p2
$ sudo mkfs.vfat /dev/loop1p3
```

7. Mount all three filesystems and show they are available:

```
$ mkdir mnt1 mnt2 mnt3

$ sudo mount /dev/loop1p1 mnt1
$ sudo mount /dev/loop1p2 mnt2
$ sudo mount /dev/loop1p3 mnt3

$ df -Th
```

| Filesystem | Type | Size | Used | Avail | Use% | Mounted on |
|--------------|------|------|------|-------|------|------------|
| /dev/sda1 | ext4 | 29G | 8.5G | 19G | 32% | / |
| | | | | | | |
| /dev/loop1p1 | ext3 | 233M | 2.1M | 219M | 1% | mnt1 |
| /dev/loop1p2 | ext4 | 233M | 2.1M | 215M | 1% | mnt2 |
| /dev/loop1p3 | vfat | 489M | 0 | 489M | 0% | mnt3 |

8. After using the filesystems to your heart's content you can unwind it all:

```
$ sudo umount mnt1 mnt2 mnt3
$ rmdir mnt1 mnt2 mnt3
$ sudo losetup -d /dev/loop1
```

Exercise 17.4: Partitioning a Real Hard Disk

If you have real hard disk un-partitioned space available, experiment with **fdisk** to create new partitions, either primary or logical within an extended partition. Write the new partition table to disk and then format and mount the new partitions.

Chapter 18

Filesystem Features: Attributes, Creating, Checking, Mounting



18.1 Labs

Exercise 18.1: Working with File Attributes

1. With your normal user account use **touch** to create an empty file named `/tmp/appendit`.
2. Use **cat** to append the contents of `/etc/hosts` to `/tmp/appendit`.
3. Compare the contents of `/tmp/appendit` with `/etc/hosts`; there should not be any differences.
4. Try to add the append-only attribute to `/tmp/appendit` by using **chattr**. You should see an error here. Why?
5. As root, retry adding the append-only attribute; this time it should work. Look at the file's extended attributes by using **lsattr**.
6. As a normal user, try and use **cat** to copy over the contents of `/etc/passwd` to `/tmp/appendit`. You should get an error. Why?
7. Try the same thing again as root. You should also get an error. Why?
8. As the normal user, again use the append redirection operator (`>>`) and try appending the `/etc/passwd` file to `/tmp/appendit`. This should work. Examine the resulting file to confirm.
9. As root, set the immutable attribute on `/tmp/appendit`, and look at the extended attributes again.
10. Try appending output to `/tmp/appendit`, try renaming the file, creating a hard link to the file, and deleting the file as both the normal user and as root.
11. We can remove this file by removing the extended attributes. Do so.

Solution 18.1

1.

```
$ cd /tmp
$ touch appendit
$ ls -l appendit
```

```

-rw-rw-r-- 1 coop coop 0 Oct 23 19:04 appendit
2. $ cat /etc/hosts > appendit
3. $ diff /etc/hosts appendit
4. $ chattr +a appendit
   chattr: Operation not permitted while setting flags on appendit
5. $ sudo chattr +a appendit
   $ lsattr appendit
   -----a-----e-- appendit
6. $ cat /etc/passwd > appendit
   bash: appendit: Operation not permitted
7. $ sudo su
   $ cat /etc/passwd > appendit
   bash: appendit: Operation not permitted
   $ exit
8. $ cat /etc/passwd >> /tmp/appendit
   $ cat appendit
9. $ sudo chattr +i appendit
   $ lsattr appendit
   ----ia-----e- appendit
10. $ echo hello >> appendit
    -bash: appendit: Permission denied
    $ mv appendit appendit.rename
    mv: cannot move `appendit' to `appendit.rename': Operation not permitted
    $ ln appendit appendit.hardlink
    ln: creating hard link `appendit.hardlink' => `appendit': Operation not permitted
    $ rm -f appendit
    rm: cannot remove `appendit': Operation not permitted
    $ sudo su
    $ echo hello >> appendit
    -bash: appendit: Permission denied
    $ mv appendit appendit.rename
    mv: cannot move `appendit' to `appendit.rename': Operation not permitted
    $ ln appendit appendit.hardlink
    ln: creating hard link `appendit.hardlink' => `appendit': Operation not permitted
    $ rm -f appendit
    rm: cannot remove `appendit': Operation not permitted
    $ exit
11. $ sudo su
    $ lsattr appendit
    ----ia-----e- appendit
    $ chattr -ia appendit
    $ rm appendit
    rm: remove regular file `appendit'? y

```

```
$ ls appendit
ls: cannot access appendit: No such file or directory
```

Exercise 18.2: Mounting Options

In this exercise you will need to either create a fresh partition, or use a loopback file. The solution will differ slightly and we will provide details of both methods.

1. Use **fdisk** to create a new 250 MB partition on your system, probably on `/dev/sda`. Or create a file full of zeros to use as a loopback file to simulate a new partition.
2. Use **mkfs** to format a new filesystem on the partition or loopback file just created. Do this three times, changing the block size each time. Note the locations of the superblocks, the number of block groups and any other pertinent information, for each case.
3. Create a new subdirectory (say `/mnt/tempdir`) and mount the new filesystem at this location. Verify it has been mounted.
4. Unmount the new filesystem, and then remount it as read-only.
5. Try to create a file in the mounted directory. You should get an error here, why?
6. Unmount the filesystem again.
7. Add a line to your `/etc/fstab` file so that the filesystem will be mounted at boot time.
8. Mount the filesystem.
9. Modify the configuration for the new filesystem so that binary files may not be executed from the filesystem (change defaults to `noexec` in the `/mnt/tempdir` entry). Then remount the filesystem and copy an executable file (such as `/bin/ls`) to `/mnt/tempdir` and try to run it. You should get an error: why?

When you are done you will probably want to clean up by removing the entry from `/etc/fstab`.

Solution 18.2



Physical Partition Solution

1. We won't show the detailed steps in **fdisk**, as it is all ground covered earlier. We will assume the partition created is `/dev/sda11`, just to have something to show.

```
$ sudo fdisk /dev/sda
.....
w
$ partprobe -s
```

Sometimes the **partprobe** won't work, and to be sure the system knows about the new partition you have to reboot.

2.

```
$ sudo mkfs -t ext4 -v /dev/sda11
$ sudo mkfs -t ext4 -b 2048 -v /dev/sda11
$ sudo mkfs -t ext4 -b 4096 -v /dev/sda11
```

Note the `-v` flag (verbose) will give the requested information; you will see that for a small partition like this the default is 1024 byte blocks.

3.

```
$ sudo mkdir /mnt/tempdir
$ sudo mount /dev/sda11 /mnt/tempdir
$ mount | grep tempdir
```
4.

```
$ sudo umount /mnt/tempdir
$ sudo mount -o ro /dev/sda11 /mnt/tempdir
```

If you get an error while unmounting, make sure you are not currently in the directory.

5. `$ sudo touch /mnt/tempdir/afile`
 6. `$ sudo umount /mnt/tempdir`
 7. Put this line in `/etc/fstab`:
`/dev/sda11 /mnt/tempdir ext4 defaults 1 2`
 8. `$ sudo mount /mnt/tempdir`
`$ sudo mount | grep tempdir`
 9. Change the line in `/etc/fstab` to:
`/dev/sda11 /mnt/tempdir ext4 noexec 1 2`
- Then do:
- ```
$ sudo mount -o remount /mnt/tempdir
$ sudo cp /bin/ls /mnt/tempdir
$ /mnt/tempdir/ls
```

You should get an error here, why?



### Loopback File Solution

1. `$ sudo dd if=/dev/zero of=/imagefile bs=1M count=250`
2. `$ sudo mkfs -t ext4 -v`  
`$ sudo mkfs -t ext4 -b 2048 -v /imagefile`  
`$ sudo mkfs -t ext4 -b 4096 -v /imagefile`

You will get warned that this is a file and not a partition, just proceed.

Note the `-v` flag (verbose) will give the requested information; you will see that for a small partition like this the default is 1024 byte blocks.

3. `$ sudo mkdir /mnt/tempdir`  
`$ sudo mount -o loop /imagefile /mnt/tempdir`  
`$ mount | grep tempdir`
4. `$ sudo umount /mnt/tempdir`  
`$ sudo mount -o ro,loop /imagefile /mnt/tempdir`

If you get an error while unmounting, make sure you are not currently in the directory.

5. `$ sudo touch /mnt/tempdir/afile`
6. `$ sudo umount /mnt/tempdir`
7. Put this line in `/etc/fstab`:



in `/etc/fstab`

```
/imagefile /mnt/tempdir ext4 loop 1 2
```

8. `$ sudo mount /mnt/tempdir`  
`$ sudo mount | grep tempdir`
9. Change the line in `/etc/fstab` to:



in `/etc/fstab`

```
/imagefile /mnt/tempdir ext4 loop,noexec 1 2
```

Then do:



```
$ sudo mount -o remount /mnt/tempdir
$ sudo cp /bin/ls /mnt/tempdir
$ /mnt/tempdir/ls
```

You should get an error here, why?



## Chapter 19

# Filesystem Features: Swap, Quotas, Usage



### 19.1 Labs

#### Exercise 19.1: Managing Swap Space

Examine your current swap space by doing:

```
$ cat /proc/swaps
```

| Filename   | Type      | Size    | Used | Priority |
|------------|-----------|---------|------|----------|
| /dev/sda11 | partition | 4193776 | 0    | -1       |

We will now add more swap space by adding either a new partition or a file. To use a file we can do:

```
$ dd if=/dev/zero of=swpfile bs=1M count=1024
```

```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 1.30576 s, 822 MB/s
```

```
$ mkswap swpfile
```

```
Setting up swapspace version 1, size = 1048572 KiB
no label, UUID=85bb62e5-84b0-4fdd-848b-4f8a289f0c4c
```

(For a real partition just feed **mkswap** the partition name, but be aware all data on it will be erased!)

Activate the new swap space:

```
$ sudo swapon swpfile
```

```
swapon: /tmp/swpfile: insecure permissions 0664, 0600 suggested.
swapon: /tmp/swpfile: insecure file owner 500, 0 (root) suggested.
```



#### On RedHat / CentOS

Notice **RHEL 7** warns us we are being insecure, we really should fix with:

```
$ sudo chown root:root swpfile
$ sudo chmod 600 swpfile
```



and ensure it is being used:

```
$ cat /proc/swaps
```

| Filename     | Type      | Size    | Used | Priority |
|--------------|-----------|---------|------|----------|
| /dev/sda11   | partition | 4193776 | 0    | -1       |
| /tmp/swpfile | file      | 1048572 | 0    | -2       |

Note the Priority field; swap partitions or files of lower priority will not be used until higher priority ones are filled.

Remove the swap file from use and delete it to save space:

```
$ sudo swapoff swpfile
$ sudo rm swpfile
```

## Exercise 19.2: Filesystem Quotas



### Please Note

The subsection describing this material was marked as optional, so you may not have covered the material necessary to do this exercise.

1. Change the entry in `/etc/fstab` for your new filesystem to use user quotas (change `noexec` to `usrquota` in the entry for `/mnt/temppdir`). Then remount the filesystem.
2. Initialize quotas on the new filesystem, and then turn the quota checking system on.
3. Now set some quota limits for the normal user account: a soft limit of 500 blocks and a hard limit of 1000 blocks.
4. As the normal user, attempt to use `dd` to create some files to exceed the quota limits. Create `bigfile1` (200 blocks) and `bigfile2` (400 blocks).  
You should get a warning. Why?
5. Create `bigfile3` (600 blocks).  
You should get an error message. Why? Look closely at the file sizes.
6. Eliminate the persistent mount line you inserted in `/etc/fstab`.

## Solution 19.2

1. Change `/etc/fstab` to have one of the following two lines according to whether you are using a real partition or a loopback file:



in `/etc/fstab`

```
/dev/sda11 /mnt/temppdir ext4 usrquota 1 2
/imagefile /mnt/temppdir ext4 loop,usrquota 1 2
```

Then remount:

```
$ sudo mount -o remount /mnt/temppdir
```

2. 

```
$ sudo quotacheck -u /mnt/temppdir
$ sudo quotaon -u /mnt/temppdir
$ sudo chown student.student /mnt/temppdir
```

(You won't normally do the line above, but we are doing it to make the next part easier).

3. Substitute your user name for the student user account.

4. `$ sudo edquota -u student`

5. `$ cd /mnt/tempdir`

```
$ dd if=/dev/zero of=bigfile1 bs=1024 count=200

200+0 records in
200+0 records out
204800 bytes (205 kB) copied, 0.000349604 s, 586 MB/s
```

`$ quota`

```
Disk quotas for user student (uid 500):
Filesystem blocks quota lim grace files qu lim gr
/dev/sda11 200 500 1000 1 0 0
```

`$ dd if=/dev/zero of=bigfile2 bs=1024 count=400`

```
sda11: warning, user block quota exceeded.
400+0 records in
400+0 records out
409600 bytes (410 kB) copied, 0.000654847 s, 625 MB/s
```

Create `bigfile3` (600 blocks).

6. `$ quota`

```
Disk quotas for user student (uid 500):
Filesystem blocks quota limit grace files qu lim gr
/dev/sda11 600* 500 1000 6days 2 0 0
```

`$ dd if=/dev/zero of=bigfile3 bs=1024 count=600`

```
sda11: write failed, user block limit reached.
dd: writing `bigfile3': Disk quota exceeded
401+0 records in
400+0 records out
409600 bytes (410 kB) copied, 0.00177744 s, 230 MB/s
```

`$ quota`

```
Disk quotas for user student (uid 500):
Filesystem blocks quota limit grace files quota limit grace
/dev/sda11 1000* 500 1000 6days 3 0 0
```

`$ ls -l`

```
total 1068
-rw----- 1 root root 7168 Dec 10 18:56 aquota.user
-rw-rw-r-- 1 student student 204800 Dec 10 18:58 bigfile1
-rw-rw-r-- 1 student student 409600 Dec 10 18:58 bigfile2
-rw-rw-r-- 1 student student 409600 Dec 10 19:01 bigfile3
drwx----- 2 root root 16384 Dec 10 18:47 lost+found
-rwxr-xr-x 1 root root 41216 Dec 10 18:52 more
```

Look closely at the file sizes.

7. Get rid of the line in `/etc/fstab`.



## Chapter 20

# The Ext2/Ext3/Ext4 Filesystems



### 20.1 Labs

#### Exercise 20.1: Defragmentation

Newcomers to **Linux** are often surprised at the lack of mention of filesystem **defragmentation** tools, since such programs are routinely used in the **Windows** world.

However, native filesystems in **UNIX**-type operating systems, including **Linux**, tend not to suffer serious problems with filesystem fragmentation.

This is primarily because they do not try to cram files onto the innermost disk regions where access times are faster. Instead, they spread free space out throughout the disk, so that when a file has to be created there is a much better chance that a region of free blocks big enough can be found to contain the entire file in either just one or a small number of pieces.

For modern hardware, the concept of innermost disk regions is obscured by the hardware anyway; and for **SSDs** defragmentation would actually shorten the lifespan of the storage media due to finite read/erase/write cycles.

Furthermore, the newer **journalling** filesystems (including **ext4**) work with **extents** (large contiguous regions) by design.

However, there does exist a tool for de-fragmenting **ext4** filesystems:

```
$ sudo e4defrag
```

```
Usage : e4defrag [-v] file...| directory...| device...
 : e4defrag -c file...| directory...| device...
```

**e4defrag** is part of the **e2fsprogs** package and should be on all modern **Linux** distributions.

The only two options are:

- **-v**: Be verbose.
- **-c**: Don't actually do anything, just analyze and report.

The argument can be:

- A file

- A directory
- An entire device

Examples:

```
$ sudo e4defrag -c /var/log
```

```
<Fragmented files>
1. /var/log/lastlog now/best size/ext
2. /var/log/sa/sa24 5/1 9 KB
3. /var/log/rhsm/rhsm.log 3/1 80 KB
4. /var/log/messages 2/1 142 KB
5. /var/log/Xorg.1.log.old 2/1 4590 KB
 1/1 36 KB

Total/best extents 120/112
Average size per extent 220 KB
Fragmentation score 1
[0-30 no problem: 31-55 a little bit fragmented: 56- needs defrag]
This directory (/var/log) does not need defragmentation.
Done.
```

```
$ sudo e4defrag /var/log
```

```
ext4 defragmentation for directory(/var/log)
[2/152]/var/log/Xorg.2.log: 100% [OK]
[3/152]/var/log/Xorg.0.log.old: 100% [OK]
[4/152]/var/log/messages-20141019.gz: 100% [OK]
[5/152]/var/log/boot.log: 100% [OK]
[7/152]/var/log/cups/page_log-20140924.gz: 100% [OK]
[8/152]/var/log/cups/access_log-20141019.gz: 100% [OK]
[9/152]/var/log/cups/access_log: 100% [OK]
[10/152]/var/log/cups/error_log-20141018.gz: 100% [OK]
[11/152]/var/log/cups/error_log-20141019.gz: 100% [OK]
[12/152]/var/log/cups/access_log-20141018.gz: 100% [OK]
[14/152]/var/log/cups/page_log-20141018.gz: 100% [OK]
...
[152/152]/var/log/Xorg.1.log.old: 100% [OK]

Success: [112/152]
Failure: [40/152]
```

Try running **e4defrag** on various files, directories, and entire devices, always trying with **-c** first.

You will generally find that **Linux** filesystems only tend to need defragmentation when they get very full, over 90 percent or so, or when they are small and have relatively large files, like when a **boot** partition is used.

## Exercise 20.2: Modifying Filesystem Parameters with tune2fs

We are going to fiddle with some properties of a formatted **ext4** filesystem. This does not require unmounting the filesystem first.

In the below you can work with an image file you create as in:

```
$ dd if=/dev/zero of=imagefile bs=1M count=1024
$ mkfs.ext4 imagefile
```

or you can substitute **/dev/sdaX** (using whatever partition the filesystem you want to modify is mounted on) for **imagefile**.

1. Using **dumpe2fs**, obtain information about the filesystem whose properties you want to adjust.

Display:

- The maximum mount count setting (after which a filesystem check will be forced.)
- The Check interval (the amount of time after which a filesystem check is forced)



- The number of blocks reserved, and the total number of blocks
2. Change:
    - The maximum mount count to 30.
    - The Check interval to three weeks.
    - The percentage of blocks reserved to 10 percent.
  3. Using **dumpe2fs**, once again obtain information about the filesystem and compare with the original output.

## ✓ Solution 20.2

1. `$ dumpe2fs imagefile > dumpe2fs-output-initial`  

```
dumpe2fs 1.42.9 (28-Dec-2013)
$ grep -i -e "Mount count" -e "Check interval" -e "Block Count" dumpe2fs-output-initial
Block count: 262144
Reserved block count: 13107
Mount count: 0
Maximum mount count: -1
Check interval: 0 (<none>)
```
2. `$ tune2fs -c 30 imagefile`  

```
tune2fs 1.42.9 (28-Dec-2013)
Setting maximal mount count to 30
$ tune2fs -i 3w imagefile
tune2fs 1.42.9 (28-Dec-2013)
Setting interval between checks to 1814400 seconds
$ tune2fs -m 10 imagefile
tune2fs 1.42.9 (28-Dec-2013)
Setting reserved blocks percentage to 10% (26214 blocks)
```
3. `$ dumpe2fs imagefile > dumpe2fs-output-final`  

```
dumpe2fs 1.42.9 (28-Dec-2013)
$ grep -i -e "Mount count" -e "Check interval" -e "Block Count" dumpe2fs-output-final
```
4. `Block count:` 262144  

```
Reserved block count: 26214
Mount count: 0
Maximum mount count: 30
Check interval: 1814400 (3 weeks)
$ diff dumpe2fs-output-initial dumpe2fs-output-final
14c14
< Reserved block count: 13107

> Reserved block count: 26214
29c29
< Last write time: Wed Oct 26 14:26:19 2016

> Last write time: Wed Oct 26 14:26:20 2016
31c31
< Maximum mount count: -1

> Maximum mount count: 30
33c33,34
< Check interval: 0 (<none>)

> Check interval: 1814400 (3 weeks)
> Next check after: Wed Nov 16 13:26:16 2016
```



## Chapter 21

# The XFS and BTRFS Filesystems \*\*



### 21.1 Labs

#### Exercise 21.1: Finding Out More About xfs



#### Please Note

We do not have a detailed lab exercise you can do with **xfs**; many systems still will not have the kernel modules and relevant user utilities installed. However, if your **Linux** kernel and distribution does support it, you can easily create a filesystem with `mkfs -t xfs`.

You can find out about available **xfs**-related utilities with:

```
$ man -k xfs
```

|                                |                                                             |
|--------------------------------|-------------------------------------------------------------|
| <code>attr (1)</code>          | - extended attributes on XFS filesystem objects             |
| <code>filesystems (5)</code>   | - Linux file-system types: minix, ext, ext2, ext3, ext4,... |
| <code>fs (5)</code>            | - Linux file-system types: minix, ext, ext2, ext3, ext4,... |
| <code>fsck.xfs (8)</code>      | - do nothing, successfully                                  |
| <code>fsfreeze (8)</code>      | - suspend access to a filesystem (Linux Ext3/4, ReiserFS... |
| <code>mkfs.xfs (8)</code>      | - construct an XFS filesystem                               |
| <code>pmdaxfs (1)</code>       | - XFS filesystem performance metrics domain agent (PMDA)    |
| <code>xfs (5)</code>           | - layout of the XFS filesystem                              |
| <code>xfs_admin (8)</code>     | - change parameters of an XFS filesystem                    |
| <code>xfs_bmap (8)</code>      | - print block mapping for an XFS file                       |
| <code>xfs_copy (8)</code>      | - copy the contents of an XFS filesystem                    |
| <code>xfs_db (8)</code>        | - debug an XFS filesystem                                   |
| <code>xfs_estimate (8)</code>  | - estimate the space that an XFS filesystem will take       |
| <code>xfs_freeze (8)</code>    | - suspend access to an XFS filesystem                       |
| <code>xfs_fsr (8)</code>       | - filesystem reorganizer for XFS                            |
| <code>xfs_growfs (8)</code>    | - expand an XFS filesystem                                  |
| <code>xfs_info (8)</code>      | - expand an XFS filesystem                                  |
| <code>xfs_io (8)</code>        | - debug the I/O path of an XFS filesystem                   |
| <code>xfs_logprint (8)</code>  | - print the log of an XFS filesystem                        |
| <code>xfs_mdrestore (8)</code> | - restores an XFS metadump image to a filesystem image      |
| <code>xfs_metadump (8)</code>  | - copy XFS filesystem metadata to a file                    |

```

xfs_mkfile (8) - create an XFS file
xfs_ncheck (8) - generate pathnames from i-numbers for XFS
xfs_quota (8) - manage use of quota on XFS filesystems
xfs_repair (8) - repair an XFS filesystem
xfs_rtcp (8) - XFS realtime copy command
xfsdump (8) - XFS filesystem incremental dump utility
xfsinvtutil (8) - xfsdump inventory database checking and pruning utility
xfsrestore (8) - XFS filesystem incremental restore utility
xqmstats (8) - Display XFS quota manager statistics from /proc

```

Read about these utility programs and see if you can play with them on the filesystem you created.

## Exercise 21.2: Finding Out More About btrfs



### Please Note

We do not have a detailed lab exercise you can do with **btrfs**; many systems still will not have the kernel modules and relevant user utilities installed. However, if your **Linux** kernel and distribution support it, you can easily create a filesystem with `mkfs -t btrfs`.

You can find out about available **btrfs**-related utilities with:

```
$ man -k btrfs
```

```

btrfs-image (8) - create/restore an image of the filesystem
btrfs-show (8) - scan the /dev directory for btrfs partitions and print...
btrfsck (8) - check a btrfs filesystem
btrfsctl (8) - control a btrfs filesystem
mkfs.btrfs (8) - create an btrfs filesystem
btrfs (8) - control a btrfs filesystem
btrfs-convert (8) - convert ext2/3/4 to btrfs.
btrfs-debug-tree (8) - dump Btrfs filesystem metadata into stdout.
btrfs-find-root (8) - filter to find btrfs root.
btrfs-map-logical (8) - map btrfs logical extent to physical extent
btrfs-show-super (8) - show btrfs superblock information stored in devices
btrfs-zero-log (8) - clear out log tree.
btrfstune (8) - tune various filesystem parameters.

```

Read about these utility programs and see if you can play with them on the filesystem you created.

## Chapter 22

# Encrypting Disks



### 22.1 Labs

#### Exercise 22.1: Disk Encryption

In this exercise, you will encrypt a partition on the disk in order to provide a measure of security in the event that the hard drive or laptop is stolen. Reviewing the **cryptsetup** documentation first would be a good idea (`man cryptsetup` and `cryptsetup --help`).

1. Create a new partition for the encrypted block device with **fdisk**. Make sure the kernel is aware of the new partition table. A reboot will do this but there are other methods.
2. Format the partition with **cryptsetup** using **LUKS** for the crypto layer.
3. Create the un-encrypted pass through device by opening the encrypted block device, i.e., `secret-disk`.
4. Add an entry to `/etc/crypttab` so that the system prompts for the passphrase on reboot.
5. Format the filesystem as an **ext4** filesystem.
6. Create a mount point for the new filesystem, i.e. `/secret`.
7. Add an entry to `/etc/fstab` so that the filesystem is mounted on boot.
8. Try and mount the encrypted filesystem.
9. Validate the entire configuration by rebooting.

#### Solution 22.1

1. `$ sudo fdisk /dev/sda`

Create a new partition (in the below `/dev/sda4` to be concrete) and then either issue:

```
$ sudo partprobe -s
```

to have the system re-read the modified partition table, or reboot (which is far safer).

**Note:** If you can't use a real partition, use the technique in the previous chapter to use a loop device or image file for the same purpose.

2. `$ sudo cryptsetup luksFormat /dev/sda4`
3. `$ sudo cryptsetup luksOpen /dev/sda4 secret-disk`
4. Add the following to `/etc/crypttab`:

```

in /etc/crypttab
secret-disk /dev/sda4

```

5. `$ sudo mkfs -t ext4 /dev/mapper/secret-disk`
6. `$ sudo mkdir -p /secret`
7. Add the following to `/etc/fstab`:

```

in /etc/fstab
/dev/mapper/secret-disk /secret ext4 defaults 1 2

```

8. Mount just the one filesystem:
 

```
$ sudo mount /secret
```

 or mount all filesystems mentioned in `/etc/fstab`:
 

```
$ sudo mount -a
```
9. Reboot.

## ✍ Exercise 22.2: Encrypted Swap

In this exercise, we will be encrypting the **swap partition**. Data written to the swap device can contain sensitive information. Because swap is backed by an actual partition, it is important to consider the security implications of having an unencrypted swap partition.

The process for encrypting is similar to the previous exercise, except we will not create a file system on the encrypted block device.

In this case, we are also going to use the existing swap device by first de-activating it and then formatting it for use as an encrypted swap device. It would be a little bit safer to use a fresh partition below, or you can safely reuse the encrypted partition you set up in the previous exercise. At the end we explain what to do if you have problems restoring.

You may want to revert back to the original unencrypted partition when we are done by just running **mkswap** on it again when it is not being used, as well as reverting the changes in the configuration files, `/etc/crypttab` and `/etc/fstab`.

1. Find out what partition you are currently using for swap and then deactivate it:

```

$ cat /proc/swaps

Filename Type Size Used Priority
/dev/sda11 partition 4193776 0 -1

$ sudo swapoff /dev/sda11

```

2. Do the same steps as in the previous exercise to set up encryption:

```

$ sudo cryptsetup luksFormat /dev/sda11 # may use --cipher aes option
$ sudo cryptsetup luksOpen /dev/sda11 swapcrypt

```

3. Format the encrypted device to use with swap:


```
$ sudo mkswap /dev/mapper/swapcrypt
```

4. Now test to see if it actually works by activating it:

```
$ sudo swapon /dev/mapper/swapcrypt
$ cat /proc/swaps
```

5. To ensure the encrypted swap partition can be activated at boot you need to do two things:

- (a) Add a line to `/etc/crypttab` so that the system prompts for the passphrase on reboot:



**in /etc/crypttab**

```
swapcrypt /dev/sda11 /dev/urandom swap,cipher=aes-cbc-essiv:sha256,size=256
```

(Note `/dev/urandom` is preferred over `/dev/random` for reasons involving potential **entropy shortages** as discussed in the **man** page for `crypttab`.) You don't need the detailed options that follow, but we give them as an example of what more you can do.

- (b) Add an entry to `/etc/fstab` so that the swap device is activated on boot.



**in /etc/fstab**

```
/dev/mapper/swapcrypt none swap defaults 0 0
```

6. You can validate the entire configuration by rebooting.

To restore your original unencrypted partition:

```
$ sudo swapoff /dev/mapper/swapcrypt
$ sudo cyyptsetup luksClose swapcrypt
$ sudo mkswap /dev/sda11
$ sudo swapon -a
```

If the **swapon** command fails it is likely because `/etc/fstab` no longer properly describes the swap partition. If this partition is described in there by actual device node (`/dev/sda11`) there won't be a problem. You can fix either by changing the line in there to be:



**in /etc/fstab**

```
/dev/sda11 swap swap defaults 0 0
```

or by giving a label when formatting and using it as in:

```
$ sudo mkswap -L SWAP /dev/sda11
```

and then putting in the file:



**in /etc/fstab**

```
LABEL=SWAP swap swap defaults 0 0
```





## Chapter 23

# Logical Volume Management (LVM)



### 23.1 Labs

#### Exercise 23.1: Logical Volumes

We are going to create a logical volume using two 250 MB partitions. We are going to assume you have real partitionable disk space available.

1. Create two 250 MB partitions of type logical volume (8e).
2. Convert the partitions to physical volumes.
3. Create a volume group named `myvg` and add the two physical volumes to it. Use the default extent size.
4. Allocate a 300 MB logical volume named `mylvm` from volume group `myvg`.
5. Format and mount the logical volume `mylvm` at `/mylvm`
6. Use `lvdisplay` to view information about the logical volume.
7. Grow the logical volume and corresponding filesystem to 350 MB.

#### Solution 23.1

1. Execute:

```
$ sudo fdisk /dev/sda
```

using whatever hard disk is appropriate, and create the two partitions. While in **fdisk**, typing `t` will let you set the partition type to 8e. While it doesn't matter if you don't set the type, it is a good idea to lessen confusion. Use `w` to rewrite the partition table and exit, and then

```
$ sudo partprobe -s
```

or reboot to make sure the new partitions take effect.

2. Assuming the new partitions are `/dev/sdaX` and `/dev/sdaY`:

```
$ sudo pvcreate /dev/sdaX
```

```
$ sudo pvcreate /dev/sdaY
```

```
$ sudo pvdisplay
```

3. 

```
$ sudo vgcreate myvg /dev/sdaX /dev/sdaY
$ sudo vgdisplay
```
4. 

```
$ sudo lvcreate -L 300M -n mylvm myvg
$ sudo lvdisplay
```
5. 

```
$ sudo mkfs.ext4 /dev/myvg/mylvm
$ mkdir /mylvm
$ sudo mount /dev/myvg/mylvm /mylvm
```

If you want the mount to be persistent, edit `/etc/fstab` to include the line:



in `/etc/fstab`

```
/dev/myvg/mylvm /mylvm ext4 defaults 0 0
```

6. 

```
$ sudo lvdisplay
```
7. 

```
$ df -h
$ sudo lvresize -r -L 350M /dev/myvg/mylvm
$ df -h
```

or

```
$ sudo lvresize -r -L +50M /dev/myvg/mylvm
```

or with older methods you can do:

```
$ df -h
$ sudo lvextend -L 350M /dev/myvg/mylvm
$ sudo resize2fs /dev/myvg/mylvm
$ df -h
```

# Chapter 24

## RAID \*\*



### 24.1 Labs

#### Exercise 24.1: Creating a RAID Device

Normally when creating a **RAID** device we would use partitions on separate disks. However, for this exercise we probably don't have such hardware available. Thus we will need to have two partitions on the same disk.

The process will be the same whether the partitions are on one drive or several (Although there is obviously little reason to actually create a **RAID** on a single device).

1. Create two 200 MB partitions of type **raid** (`fd`) either on your hard disk using **fdisk**, or using **LVM**.
2. Create a **RAID 1** device named `/dev/md0` using the two partitions.
3. Format the **RAID** device as an **ext4** filesystem. Then mount it at `/myraid` and make the mount persistent.
4. Place the information about `/dev/md0` in `/etc/mdadm.conf` file using **mdadm**. (Depending on your distribution, this file may not previously exist.)
5. Examine `/proc/mdstat` to see the status of your **RAID** device.

#### Solution 24.1

1. If you need to create new partitions do:

```
$ sudo fdisk /dev/sda
```

and create the partitions as we have done before. For purposes of being definite, we will call them `/dev/sdaX` and `/dev/sdaY`. You will need to run **partprobe** or **kpartx** or reboot after you are done to make sure the system is properly aware of the new partitions.

2. 

```
$ sudo mdadm -C /dev/md0 --level=1 --raid-disks=2 /dev/sdaX /dev/sdaY
```

3. 

```
$ sudo mkfs.ext4 /dev/md0
$ sudo mkdir /myraid
$ sudo mount /dev/md0 /myraid
```

and add to `/etc/fstab`



in `/etc/fstab`

```
/dev/md0 /myraid ext4 defaults 0 0
```

4. `$ mdadm --detail --scan >> /etc/mdadm.conf`

5. `$ cat /proc/mdstat`

```
Personalities : [raid1]
md0 : active raid1 dm-14[1] dm-13[0]
 204736 blocks [2/2] [UU]

unused devices: <none>
```

You should probably verify that with a reboot, the **RAID** volume is mounted automatically. When you are done, you probably will want to clean up by removing the line from `/etc/fstab`, and then getting rid of the partitions.

## Chapter 25

# Kernel Services and Configuration



### 25.1 Labs

#### Exercise 25.1: System Tunables with sysctl

1. Check if you can **ping** your own system.
2. Check the current value of `net.ipv4.icmp_echo_ignore_all`, which is used to turn on and off whether your system will respond to **ping**. A value of 0 allows your system to respond to pings.
3. Set the value to 1 using the **sysctl** command line utility and then check if pings are responded to.
4. Set the value back to 0 and show the original behavior is restored.
5. Now change the value by modifying `/etc/sysctl.conf` and force the system to activate this setting file without a reboot.
6. Check that this worked properly.

You will probably want to reset your system to have its original behavior when you are done.

#### Solution 25.1

You can use either `localhost`, `127.0.0.1` (loopback address) or your actual IP address for target of **ping** below.

1. `$ ping localhost`
2. `$ sysctl net.ipv4.icmp_echo_ignore_all`
3. `$ sudo sysctl net.ipv4.icmp_echo_ignore_all=1`  
`$ ping localhost`
4. `$ sudo sysctl net.ipv4.icmp_echo_ignore_all=0`  
`$ ping localhost`
5. Add the following line to `/etc/sysctl.conf`:

in `/etc/sysctl.conf`

```
net.ipv4.icmp_echo_ignore_all=1
```

and then do:

```
$ sysctl -p
```

6. 

```
$ sysctl net.ipv4.icmp_echo_ignore_all
```

```
$ ping localhost
```

Since the changes to `/etc/sysctl.conf` are persistent, you probably want to restore things to its previous state.

## Exercise 25.2: Changing the Maximum Process ID

The normal behavior of a **Linux** system is that process IDs start out at PID=1 for the **init** process, the first user process on the system, and then go up sequentially as new processes are constantly created (and die as well.)

However, when the PID reaches the value shown in `/proc/sys/kernel/pid_max`, which is conventionally 32768 (32K), they will wrap around to lower numbers. If nothing else, this means you can't have more than 32K processes on the system since there are only that many slots for PIDs.

1. Obtain the current maximum PID value.
2. Find out what current PIDs are being issued
3. Reset `pid_max` to a lower value than the ones currently being issued.
4. Start a new process and see what it gets as a PID.

## Solution 25.2

In the below we are going to use two methods, one involving **sysctl**, the other directly echoing values to `/proc/sys/kernel/pid_max`. Note that the **echo** method requires you to be root; **sudo** won't work. We'll leave it to you to figure out why, if you don't already know!

1. 

```
$ sysctl kernel.pid_max
```

```
$ cat /proc/sys/kernel/pid_max
```
2. Type:  

```
$ cat &
```

```
[1] 29222
```

```
$ kill -9 29222
```
3. 

```
$ sudo sysctl kernel.pid_max=24000
```

```
$ echo 24000 > /proc/sys/kernel/pid_max # This must be done as root
```

```
$ cat /proc/sys/kernel/pid_max
```
4. 

```
$ cat &
```

```
[2] 311
```

```
$ kill -9 311
```



### Please Note

Note that when starting over, the kernel begins at PID=300, not a lower value. You might notice that assigning PIDs to new processes is actually not trivial; since the system may have already turned over, the kernel always has to check when generating new PIDs that the PID is not already in use. The **Linux** kernel has a very efficient way of doing this that does not depend on the number of processes on the system.

# Chapter 26

## Kernel Modules



### 26.1 Labs

#### Exercise 26.1: Kernel Modules

1. List all currently loaded kernel modules on your system.
2. Load a currently unloaded module on your system.  
If you are running a distribution kernel, this is easy to find; you can simply look in the `/lib/modules/<kernel-version>/kernel/drivers/net` directory and grab one. (Distribution kernels come with drivers for every device, filesystem, network protocol etc. that a system might need.) However, if you are running a custom kernel you may not have many unloaded modules compiled.
3. Re-list all loaded kernel modules and see if your module was indeed loaded.
4. Remove the loaded module from your system.
5. Re-list again and see if your module was properly removed.

#### Solution 26.1

1. `$ lsmod`
2. In the following, substitute whatever module name you used for `8139too`. Either of these methods work but, of course, the second is easier.  

```
$ sudo insmod /lib/modules/$(uname -r)/kernel/drivers/net/ethernet/realtek/8139too.ko
$ sudo /sbin/modprobe 8139too
```
3. `$ lsmod | grep 8139too`
4. Once again, either method works.  

```
$ sudo rmmod 8139too
$ sudo modprobe -r 8139too
```
5. `$ lsmod | grep 8139too`





# Chapter 27

## Devices and udev



### 27.1 Labs

#### Exercise 27.1: udev

1. Create and implement a rule on your system that will create a symlink called `myusb` when a **USB** device is plugged in.
2. Plug in a **USB** device to your system. It can be a pen drive, mouse, webcam, etc.  
Note: If you are running a virtual machine under a hypervisor, you will have to make sure the **USB** device is seen by the guest, which usually is just a mouse click which also disconnects it from the host.
3. Get a listing of the `/dev` directory and see if your symlink was created.
4. Remove the **USB** device. (If it is a drive you should always **umount** it first for safety.)
5. See if your symbolic link still exists in `/dev`.

#### Solution 27.1

1. Create a file named `/etc/udev/rules.d/75-myusb.rules` and have it include just one line of content:  

```
$ cat /etc/udev/rules.d/75-myusb.rules
```

```
SUBSYSTEM=="usb", SYMLINK+="myusb"
```

Do not use the deprecated key value `BUS` in place of `SUBSYSTEM`, as recent versions of **udev** have removed it.

Note the name of this file really does not matter. If there was an `ACTION` component to the rule the system would execute it; look at other rules for examples.
2. Plug in a device.
3. 

```
$ ls -lF /dev | grep myusb
```
4. If the device has been mounted:  

```
$ umount /media/whatever
```

where `/media/whatever` is the mount point. Safely remove the device.
5. 

```
$ ls -lF /dev | grep myusb
```



## Chapter 28

# Virtualization Overview



### 28.1 Labs

#### Exercise 28.1: Making Sure KVM is Properly Set up



#### Very Important

- The following labs are best run on a physical machine running **Linux** natively.
- It may be possible to run them within a Virtual Machine running under a hypervisor, such as **VMWare** or **Virtual Box**, or even **KVM**. However, this requires **nested virtualization** to be running properly.
- Whether or not this works depends on the particular hypervisor used, the underlying host operating system (i.e., **Windows**, **Mac OS** or **Linux**) as well as the particular variant, such as which **Linux** or **Windows** version as well as the particular kernel.
- Furthermore it also depends on your particular hardware. For example, we have found nested virtualization working with **VMWare** on various **x86.64** machines but with **Oracle Virtual Box** only on some.
- If this works, performance will be poor as compared to running on native hardware, but that is not important for the simple demonstrative exercises we will do.
- Your mileage will vary! If it does not work we cannot be responsible for helping you trying to get it rolling.

1. First check that you have hardware virtualization available and enabled:

```
$ grep -e vmx -e svm /proc/cpuinfo
```

where **vmx** is for **INTEL** CPUs and **svm** for **AMD**. If you do not see either one of these:

- If you are on a physical machine, maybe you can fix this. Reboot your machine and see if you can turn on virtualization in the BIOS settings. Note that some IT personnel may make this impossible for “security” reasons, so try to get that policy changed.
  - You are on a virtual machine running under a hypervisor, and you do not have nested virtualization operable.
2. If for either of these reasons, you do not have hardware virtualization, you **may** be able to run **virt-manager**, but with weak performance.

3. You need all relevant packages installed on your system. One can work hard to construct an exact list. However, exact names and requirements change with time, and most enterprise distributions ship with all (or almost all) of the software you need.
4. The easiest and best procedure is to run the script we have already supplied to you:

```
$./ready-for.sh --install LFS201
```

where we have done the hard work.

Alternatively, on **RPM** systems you can do some overkill with:

```
$ sudo yum|dnf|zypper install kvm* qemu* libvirt*
```

It is not a large amount of storage space to do it this way.

On **Debian** package based systems including **Ubuntu** you will have to do the equivalent with your favorite package installing procedure.



### Very Important

- Do not run **libvirtd** at the same time as another hypervisor as dire consequences are likely to arise. This can easily include crashing your system and doing damage to any virtual machines being used.
- We recommend both **stopping** and **disabling** your other hypervisor as in:

```
$ sudo systemctl stop vmware
$ sudo systemctl disable vmware
```

or

```
$ sudo systemctl stop vboxdrv
$ sudo systemctl disable vboxdrv
```

## ✍ Exercise 28.2: Using virt-manager with KVM to Install a Virtual Machine and Run it

In this exercise we will use pre-built **iso** images built by **TinyCoreLinux** (<http://www.tinycorelinux.com>) because they are cooked up very nicely and are quite small.

If you would like, you can substitute any installation **iso** image for another **Linux** distribution, such as **Debian**, **CentOS**, **Ubuntu**, **Fedora**, **OpenSUSE** etc. The basic steps will be identical and only differ when you get to the installation phase for building your new VM; which is no different than building any fresh installation on an actual physical machine.

We will give step-by-step instructions with screen capture images; If you feel confident, please try to just launch **virt-manager** and see if you can work your way through the necessary steps, as the **GUI** is reasonably clearly constructed.

1. Make sure **libvirtd** is running and start **virt-manager** by typing:

```
$ sudo systemctl start libvirtd
$ sudo virt-manager
```

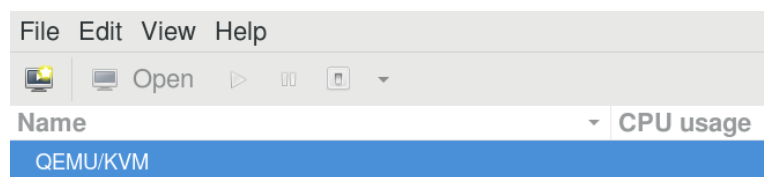


Figure 28.1: Starting virt-manager

- Click on File->Create New Machine :

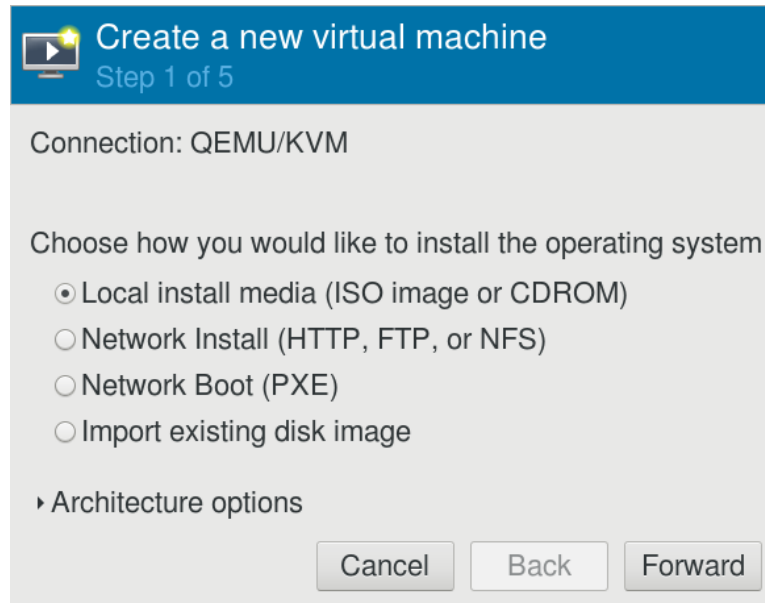


Figure 28.2: Creating a Virtual Machine with virt-manager

- We have included three different **iso** install images from **TinyCoreLinux** in the [RESOURCES/s\\_28](#) directory:

[Core-current.iso](#)

[CorePlus-current.iso](#)

[TinyCore-current.iso](#)

(You can check and see if there are newer versions upstream at <http://www.tinycorelinux.com> but these should be fine.)

**CorePlus**-current is largest and robust and we will use this as it will install with full graphics. The others are considerably quicker to use, however.

Navigate through your file system and pick the desired image:

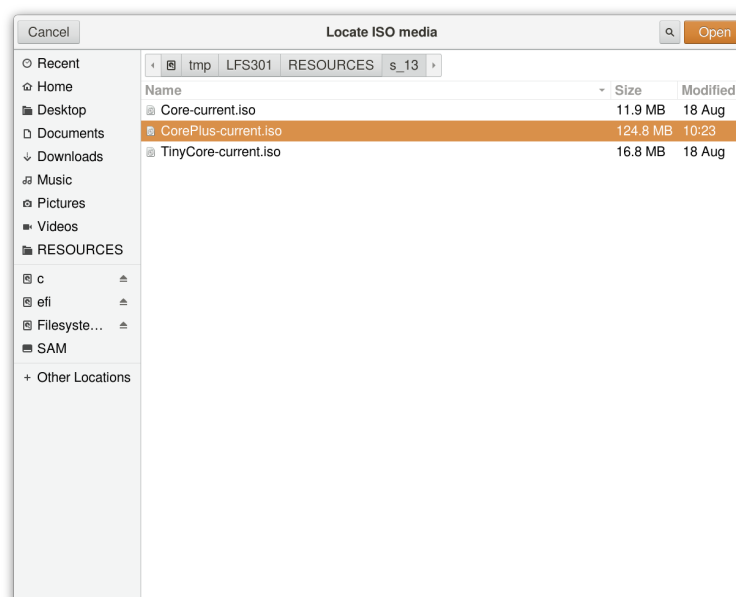


Figure 28.3: Selecting the TinyCoreLinux iso image in virt-manager

4. Next you have to request the amount of memory and number of **CPUs** or **cores** to use. These images are pretty minimal. A choice of 256 MB is more than enough; you may have fun seeing how low you can go!

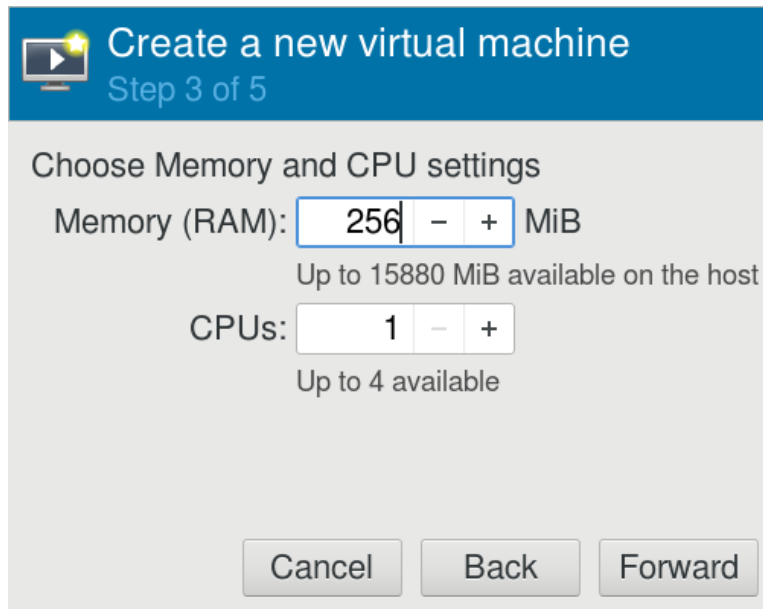


Figure 28.4: **Configuring Memory and CPUS in virt-manager**

5. Next you have to configure the location and size of the VM that is being created. You actually need very little for **TinyCoreLinux**, but the GUI will not let you choose less than 0.1 GB (about 100 MB.) (From the command line it is easy to configure less space.)

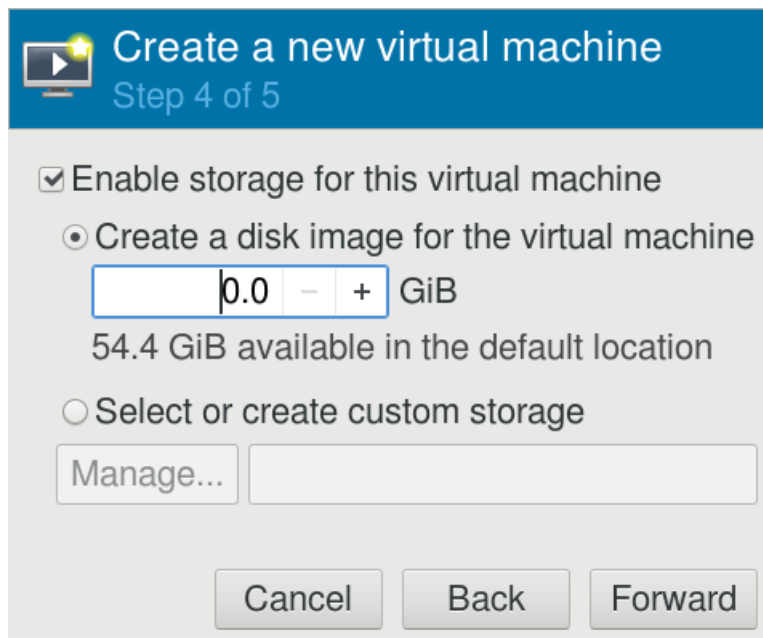


Figure 28.5: **Configuring Disk Storage in virt-manager**

If you do not click on **Select** or **Create custom storage** your image will be placed in `/var/lib/libvirt/images`. Since images can be quite large, you might want to configure to put it elsewhere. Or you can replace the images directory in `/var/lib/libvirt` with a symbolic link to somewhere else, as in:

```
$ sudo cd /var/lib/libvirt
```

```
$ sudo mv images images_ORIGINAL
$ sudo mkdir /tmp/images
$ sudo ln -s /tmp/images images
```

(You probably want a different location for the images files than `/tmp`, but you get the idea.)

6. You are now ready to begin installation of your own VM from the **TinyCoreLinux** installation disk:

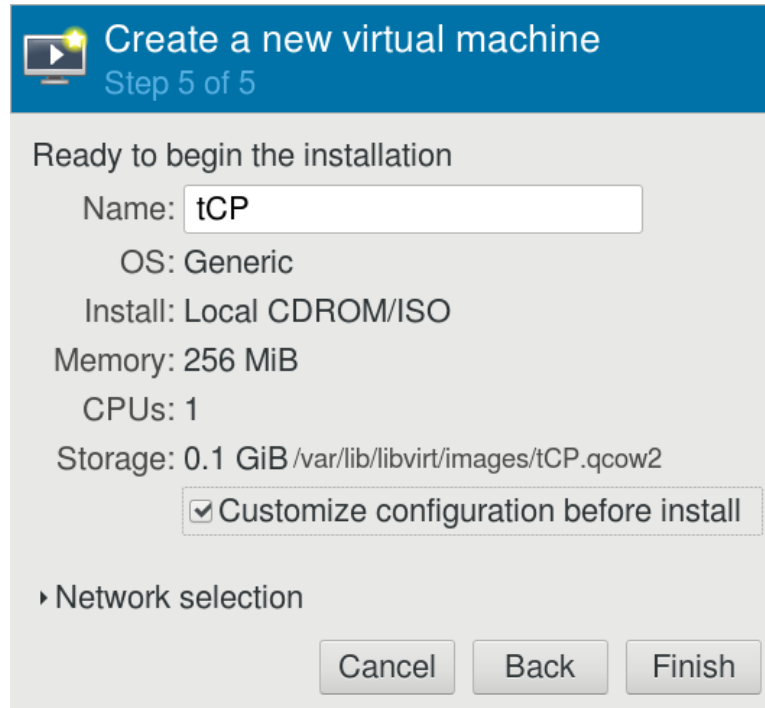


Figure 28.6: Beginning the VM installation in virt-manager



#### Please Note

We recommend clicking on **Customize** configuration before install. While you may want to make other changes, the mouse pointer is configured by default to be a **PS2** device; it is better to add a **USB tablet** input pointer.

Do this by clicking on Add Hardware on the next screen and then:

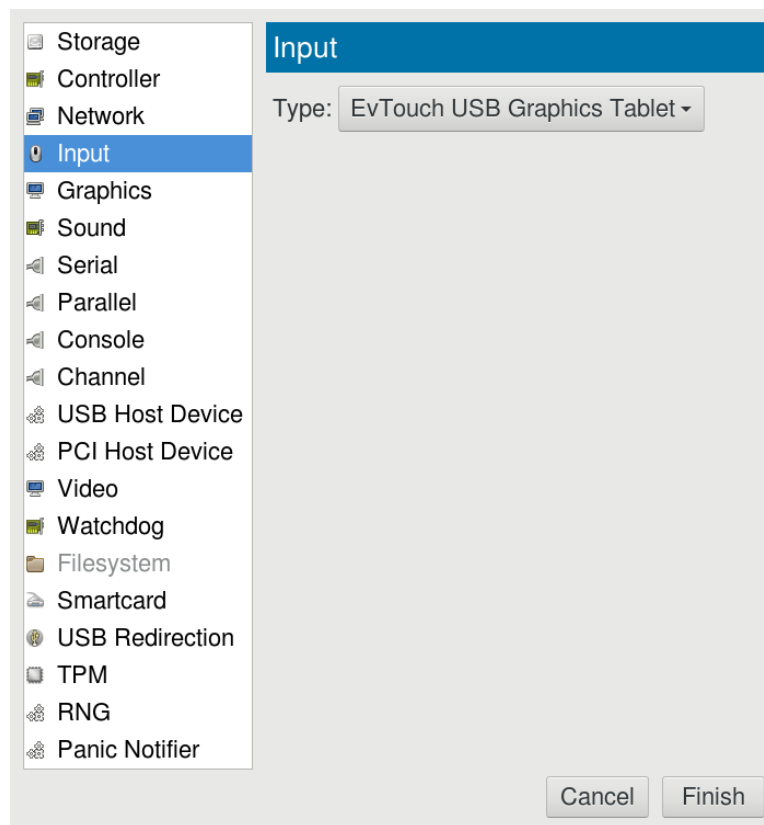


Figure 28.7: Adding an Input Device to the VM in virt-manager

7. Finally, we begin the installation:

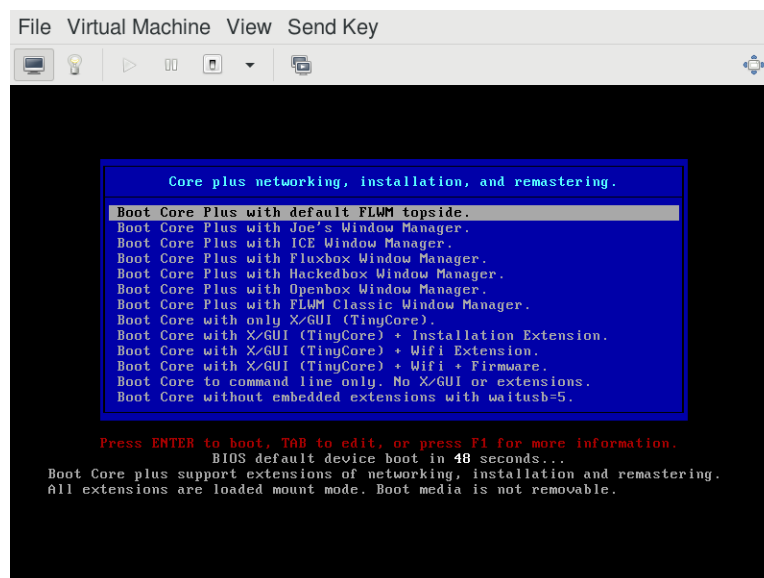


Figure 28.8: Booting into the Installation Media with virt-manager

You can make other choices for the graphical interface, here we just choose the first one, the default, and hit return.

8. This will take a while and eventually you will see the following screen:



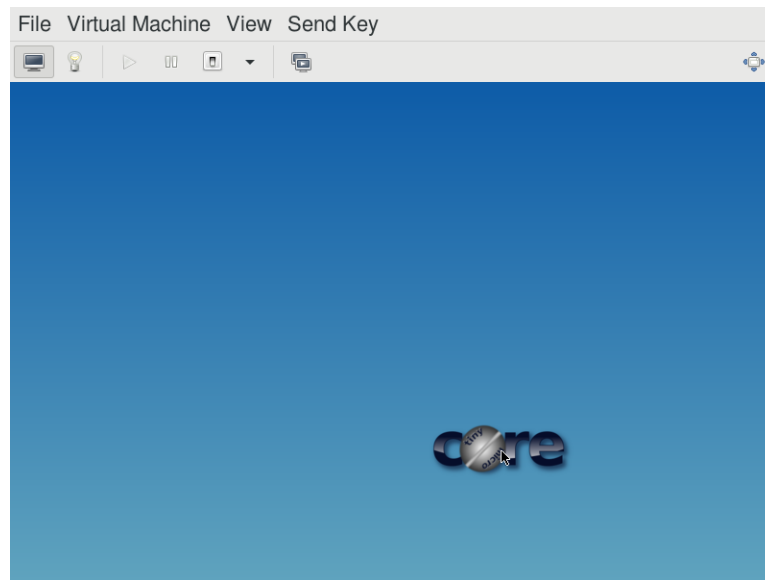


Figure 28.9: **First TinyCoreLinux Screen**

It is not obvious what to do here, but you need to see the icons at the bottom so you should resize and make the screen taller:

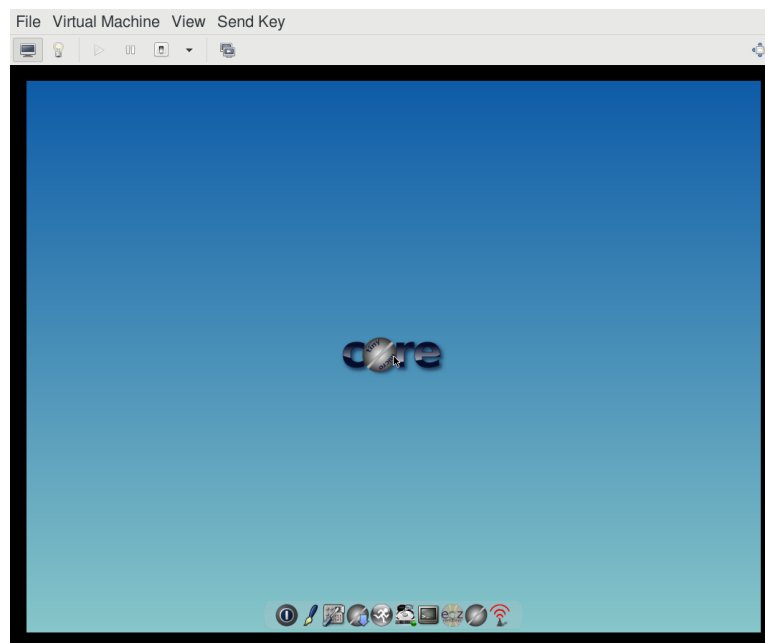


Figure 28.10: **First TinyCoreLinux Screen Resized**

9. Click on the terminal icon (or right click on the background and open up a terminal. Note the font is microscopic unfortunately. Then type

```
tc-install
```

in the window.

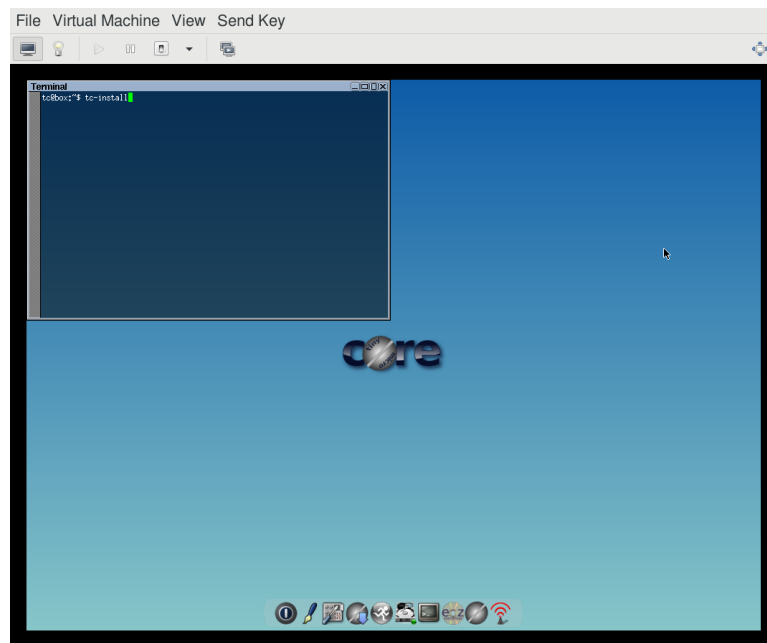


Figure 28.11: Running tc-install

10. Select `Whole` disk and `sda` and click the forward arrow:

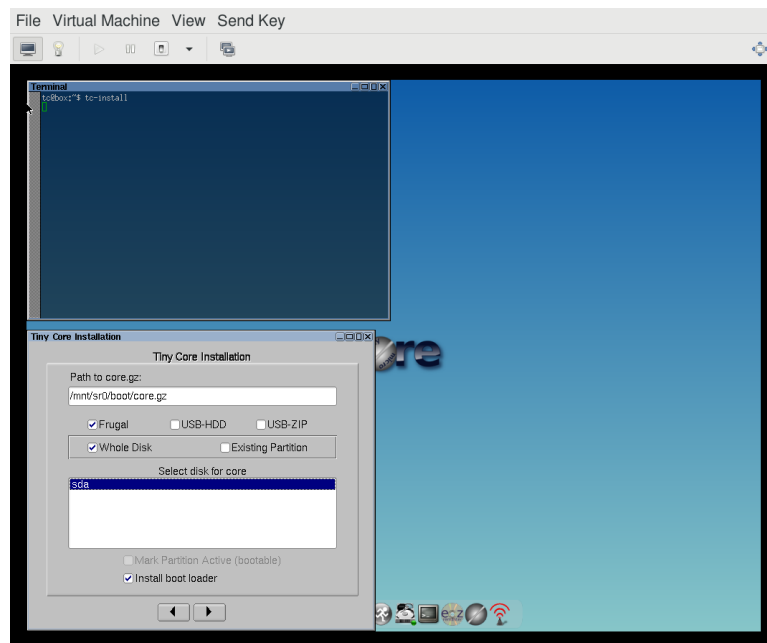


Figure 28.12: Selecting Disk in VM

11. Things will crank for a while and each step will be reflected in the output window.

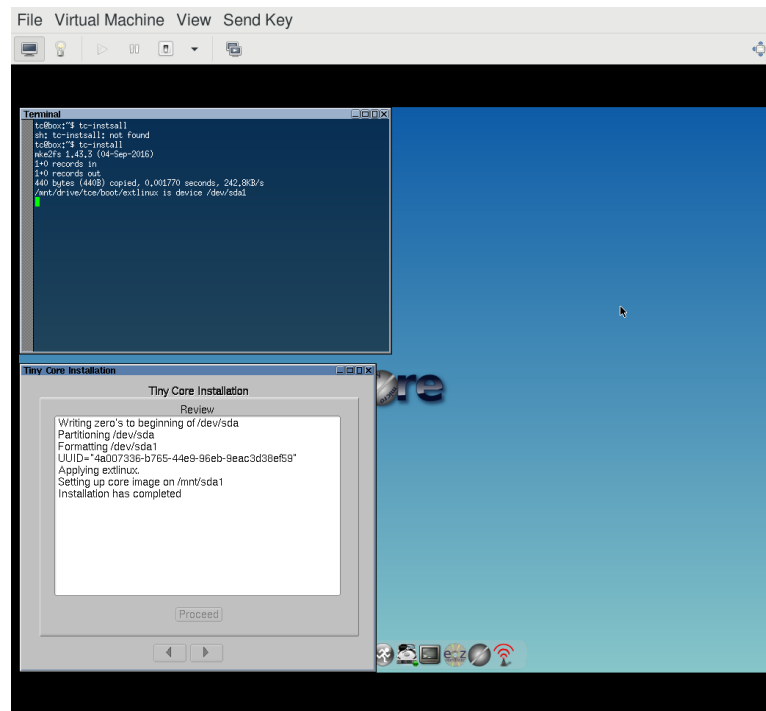


Figure 28.13: Finishing Installation in virt-manager

When installation is complete you can go to the **File** menu and shut down the virtual machine.

12. Start up **virt-manager** again (if you have killed it) and you should now see something like:

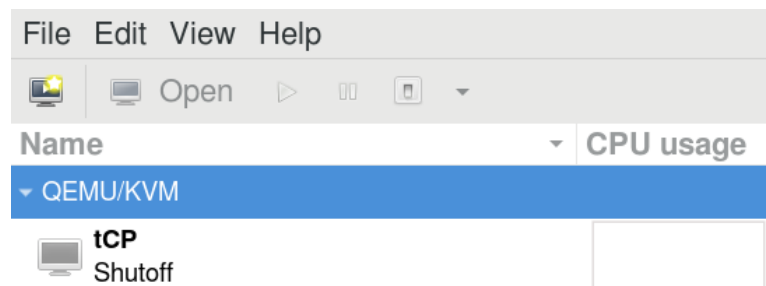


Figure 28.14: Running the new VM in virt-manager

Right click on the VM and open and run. Your new virtual machine should be up and running! (If you get confused and

think you are running the original install image, you can verify it is not that by noting there is no **tc-install** program in the new disk image.

### **Exercise 28.3: Extra Credit: Doing from the command line**

From the command line:

```
$ sudo qemu-img create -f qcow2 /var/lib/libvirt/d/myimg.qcow2 24M
$ sudo qemu-system-x86_64 -hda /var/lib/libvirt/d/myimg.qcow2 \
 -cdrom /teaching/LFCW/RESOURCES/LFS201/CorePlus-current.iso -usbdevice tablet
```

## Chapter 29

# Containers Overview



### 29.1 Labs

#### Exercise 29.1: Install and test Apache (httpd) as a Docker application.

In this exercise, we will install, run and test the **docker** package, and follow with getting and deploying **httpd**, the **Apache** web server container.



#### Please Note

**Docker** requires a **Linux** kernel of 3.10 or greater, which is not a problem for all recent **Linux** distributions. (**RedHat / CentOS 7** has a 3.10-based kernel, the others are all more recent.)

A 64-bit system is **required** for this exercise. Do not attempt on a 32-bit system.

1. Make sure **Docker** is installed. Pick the right command for your distribution from the below:

```
$ sudo yum install docker
$ sudo zypper install docker
$ sudo apt install docker.io
```

#### Reinstall Docker?

- If you get strange errors at later points in the exercise you might find it useful to **reinstall docker**. We have observed cases (for example, with **RHEL 7**) where **docker** configurations were broken, after a system upgrade,

2. Start the **docker** service.

```
$ sudo systemctl start docker
```

You may want to verify that it is running properly with `systemctl status docker`:

```

File Edit View Search Terminal Help
c7:/tmp>sudo systemctl status docker
● docker.service - Docker Application Container Engine
 Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
 Active: active (running) since Fri 2017-10-06 07:30:24 CDT; 3h 14min ago
 Docs: http://docs.docker.com
 Main PID: 6185 (dockerd-current)
 CGroup: /system.slice/docker.service
 └─6185 /usr/bin/dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --authoriz...
 └─6212 /usr/bin/docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --shim docker-containerd-shim --m...

Oct 06 07:30:24 c7 dockerd-current[6185]: time="2017-10-06T07:30:24.400523213-05:00" level=warning msg="Your kernel does not support cgroup cfs period"
Oct 06 07:30:24 c7 dockerd-current[6185]: time="2017-10-06T07:30:24.400531993-05:00" level=warning msg="Your kernel does not support cgroup cfs quotas"
Oct 06 07:30:24 c7 dockerd-current[6185]: time="2017-10-06T07:30:24.400778902-05:00" level=info msg="Loading containers: start."
Oct 06 07:30:24 c7 dockerd-current[6185]: time="2017-10-06T07:30:24.406788047-05:00" level=info msg="Firewalld running: true"
Oct 06 07:30:24 c7 dockerd-current[6185]: time="2017-10-06T07:30:24.740036679-05:00" level=info msg="Default bridge (docker0) is assigned with IP address 172.17.0.1"
Oct 06 07:30:24 c7 dockerd-current[6185]: time="2017-10-06T07:30:24.873256177-05:00" level=info msg="Loading containers: done."
Oct 06 07:30:24 c7 dockerd-current[6185]: time="2017-10-06T07:30:24.873424546-05:00" level=info msg="Daemon has completed initialization"
Oct 06 07:30:24 c7 dockerd-current[6185]: time="2017-10-06T07:30:24.873435907-05:00" level=info msg="Docker daemon" commit="c4618fb/1.12.6" g.o.n=1.12.6
Oct 06 07:30:24 c7 dockerd-current[6185]: time="2017-10-06T07:30:24.876818269-05:00" level=info msg="API listen on /var/run/docker.sock"
Oct 06 07:49:02 c7 dockerd-current[6185]: time="2017-10-06T07:49:02.019628490-05:00" level=info msg="{Action=ping, Username=root, LoginUID=0...D=8273}"
Hint: Some lines were ellipsized, use -l to show in full.
c7:/tmp>

```

Figure 29.1: Checking docker status

If you see anything indicating failure you should inspect `/var/log/messages` or whatever other logging file you have on your system for clues. If you are running a standard distribution kernel you should be fine, but if you are running a custom **Linux** kernel, it is likely you have to select the proper configuration options, especially as regards to networking. This is too complicated to go into here, so please stay with a distribution supplied kernel unless you want a challenging exercise!

### 3. Search for the **httpd** container, with `docker search apache`:

```

File Edit View Search Terminal Help
c7:/tmp>sudo docker search apache
INDEX NAME DESCRIPTION STARS OFFICIAL AUTOMATED
docker.io docker.io/tomcat Apache Tomcat is an open source implementa... 1510 [OK]
docker.io docker.io/httpd The Apache HTTP Server Project 1274 [OK]
docker.io docker.io/cassandra Apache Cassandra is an open-source distrib... 652 [OK]
docker.io docker.io/maven Apache Maven is a software project managem... 474 [OK]
docker.io docker.io/solr Solr is the popular, blazing-fast, open so... 438 [OK]
docker.io docker.io/eboraas/apache-php PHP5 on Apache (with SSL support), built o... 132 [OK]
docker.io docker.io/eboraas/apache Apache (with SSL support), built on Debian 85 [OK]
docker.io docker.io/webdevops/php-apache Apache with PHP-FPM (based on webdevops/php) 43 [OK]
docker.io docker.io/tomee Apache TomEE is an all-Apache Java EE cert... 40 [OK]
docker.io docker.io/nimmia/apache-php5 This is docker images of Ubuntu 14.04 LTS ... 36 [OK]
docker.io docker.io/fedora/apache Bitnami Apache Docker Image 33 [OK]
docker.io docker.io/bitnami/apache Bitnami Apache Docker Image 32 [OK]
docker.io docker.io/webdevops/php-apache-dev PHP with Apache for Development (eg. with ... 27 [OK]
docker.io docker.io/apacheignite/ignite Apache Ignite In-Memory docker image 23 [OK]
docker.io docker.io/linuxserver/apache An Apache container, brought to you by Lin... 14 [OK]
docker.io docker.io/apache/nutch Apache Nutch 13 [OK]
docker.io docker.io/chriswayn/apache-php Apache 2.4 web server with PHP 5.6 based o... 11 [OK]
docker.io docker.io/coreos/apache Basic install of Apache for use in example... 11 [OK]
docker.io docker.io/webdevops/apache Apache container 10 [OK]
docker.io docker.io/antage/apache2-php5 Docker image for running Apache 2.x with P... 4 [OK]
docker.io docker.io/lephare/apache Apache container 3 [OK]
docker.io docker.io/landlinternet/ubuntu-16-apache ubuntu-16-apache 2 [OK]
docker.io docker.io/landlinternet/ubuntu-16-apache-php-7.0 ubuntu-16-apache-php-7.0 1 [OK]
docker.io docker.io/mastertinner/apache-directory-index-resource A Concourse resource for the apache direct... 1 [OK]
docker.io docker.io/landlinternet/ubuntu-16-apache-php-5.6 ubuntu-16-apache-php-5.6 0 [OK]
redhat.com registry.access.redhat.com/rhsccl/httpd-24-rhel7 Apache HTTP 2.4 Server 0
redhat.com registry.access.redhat.com/rhsccl/s2i-core-rhel7 The s2i core container image serves as a b... 0
c7:/tmp>

```

Figure 29.2: Using docker search

(You could have used **httpd** instead of **apache** in the above command with very similar results.)

From now on we will not show detailed output since if you have gotten this far, things should be fine.

### 4. Retrieve the container:

```
$ sudo docker pull docker.io/httpd
```

This may take a couple of minutes while all the components download.

### 5. List the installed containers:

```
$ sudo docker images
```

### 6. List the components associated with the images.

```
$ sudo docker images --all
```

### 7. Start the **httpd** docker container. The terminal will appear to hang as it is now connected to the **httpd** daemon.

```
c7:/tmp>sudo docker run httpd
```

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set
.....
```

8. You can open a graphical web browser pointing to the IP address in the above output.

Or you can use a text-based browser if you are not in a graphical environment, by opening up a new terminal window (do not kill the one in which the **docker httpd** container is running) by doing one of the following commands:

```
$ lynx http://172.17.0.2
$ w3m http://172.17.0.2
$ elinks http://172.17.0.2
```

using whichever graphical browser is installed on your system.

9. Stop the container and **docker** service and clean up.

```
c7:/tmp>sudo docker ps
```

| CONTAINER ID | IMAGE | COMMAND            | CREATED        | STATUS        | PORTS  | NAMES         |
|--------------|-------|--------------------|----------------|---------------|--------|---------------|
| b936b0afeb23 | httpd | "httpd-foreground" | 41 seconds ago | Up 40 seconds | 80/tcp | boring_turing |

```
c7:/tmp>sudo docker stop b936b0afeb23
b936b0afeb23
```

```
c7:/tmp>sudo docker rmi -f docker.io/httpd
```

```
Untagged: docker.io/httpd:latest
```

```
Untagged: docker.io/httpd@sha256:cf774f082e92e582d02acdb76dc84e61dcf5394a90f99119d1ae39bcecbff075
```

```
Deleted: sha256:cf6b6d2e846326d2e49e12961ee0f63d8b5386980b5d3a11b8283151602fa756
```

```
c7:/tmp>sudo systemctl stop docker
```





## Chapter 30

# User Account Management



### 30.1 Labs

#### Exercise 30.1: Working with User Accounts

1. Examine `/etc/passwd` and `/etc/shadow`, comparing the fields in each file, especially for the normal user account. What is the same and what is different?
2. Create a `user1` account using **useradd**.
3. Login as `user1` using `ssh`. You can just do this with:  

```
$ ssh user1@localhost
```

It should fail because you need a password for `user1`; it was never established.
4. Set the password for `user1` to `user1pw` and then try to login again as `user1`.
5. Look at the new records which were created in the `/etc/passwd`, `/etc/group` and the `/etc/shadow` files.
6. Look at the `/etc/default/useradd` file and see what the current defaults are set to. Also look at the `/etc/login.defs` file.
7. Create a user account for `user2` which will use the **Korn** shell (**ksh**) as its default shell. (if you do not have `/bin/ksh` install it or use the **C** shell at `/bin/csh`.) Set the password to `user2pw`.
8. Look at `/etc/shadow`. What is the current expiration date for the `user1` account?
9. Use **chage** to set the account expiration date of `user1` to December 1, 2013.  
Look at `/etc/shadow` to see what the new expiration date is.
10. Use **usermod** to lock the `user1` account.  
Look at `/etc/shadow` and see what has changed about `user1`'s password. Reset the password to `userp1` on the account to complete this exercise.

#### Solution 30.1

1. `$ sudo grep student /etc/passwd /etc/shadow`

```
/etc/passwd:student:x:1000:100:LF Student:/home/student:/bin/bash
/etc/shadow:student:6jtoFVPIChhba$iGFFU8ctrtrt0GoistJ4/30DrNLi1FS66qnn0VbS6Mvm
luKI08SgbzT5.Ic0Ho5j/S0dCagZmF2RgzTvzLb11H0:16028:0:99999:7:::
```

(You can use any normal user name in the place of student.) About the only thing that matches is the user name field.

2. `$ sudo useradd user1`

3. `$ ssh user1@localhost`

```
user1@localhost's password:
```

Note you may have to first start up the **sshd** service as in:

```
$ sudo service sshd restart
```

or

```
$ sudo systemctl restart sshd.service
```

4. `$ sudo passwd user1`

```
Changing password for user user1.
New password:
```

5. `$ sudo grep user1 /etc/passwd /etc/shadow`

```
/etc/passwd:user1:x:1001:100::/home/user1:/bin/bash
/etc/shadow:user1:$6$0BE1mPMw$C1c7urbQ9ZSnyiniV0eJxKqLFu8fz4whfEexVem2
TFpucuwRN1CCHZ19XGhj4qVujslRIS.P4aCXd/y1U4utv.:16372:0:99999:7:::
```

6. On either **RHEL 7** or **openSUSE** systems for example:

```
$ cat /etc/default/useradd
```

```
useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

```
$ cat /etc/login.defs
```

```
....
```

We don't reproduce the second file as it is rather longer, but examine it on your system.

7. `$ sudo useradd -s /bin/ksh user2`

```
$ sudo passwd user2
```

```
Changing password for user user2.
New password:
```

8. `$ sudo grep user1 /etc/shadow`

```
user1:$6$0BE1mPMw$C1c7urbQ9ZSnyiniV0eJxKqLFu8fz4whfEexVem2TFpucuwRN1CCHZ
19XGhj4qVujslRIS.P4aCXd/y1U4utv.:16372:0:99999:7:::
```

There should be no expiration date.

9. `$ sudo chage -E 2013-12-1 user1`

```
$ sudo grep user1 /etc/shadow
```

```
user1:$6$0BE1mPMw$C1c7urbQ9ZSnyiniV0eJxKqLFu8fz4whfEexVem2TFpucuwRN1CCHZ
19XGhj4qVujslRIS.P4aCXd/y1U4utv.:16372:0:99999:7::16040:
```

10. `$ sudo usermod -L user1`

```
$ sudo passwd user1
```

## Exercise 30.2: Restricted Shells and Accounts

1. Start a restricted shell in your current window with:

```
$ bash -r
```

Try elementary options such as resetting the path or changing directories.

2. Set up a restricted account and verify its restricted nature, then clean up.

## Solution 30.2

1. 

```
c7:/tmp>rbash -r
c7:/tmp>cd $HOME

rbash: cd: restricted
c7:/tmp>PATH=$PATH:/tmp

rbash: PATH: readonly variable
c7:/tmp>exit

exit
```
2. 

```
c7:/home/coop>sudo ln /bin/bash /bin/rbash
c7:/home/coop>sudo useradd -s /bin/rbash fool
c7:/home/coop>sudo passwd fool

Changing password for user fool.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.

c7:/home/coop>sudo su - fool

Last failed login: Tue Oct 25 14:15:54 CDT 2016 on pts/1
There was 1 failed login attempt since the last successful login.
Attempting to create directory /home/fool/perl5

[fool@c7 ~]$ cd /tmp

-rbash: cd: restricted

[fool@c7 ~]$ PATH=$PATH:/tmp

-rbash: PATH: readonly variable

[fool@c7 ~]$ exit

logout

c7:/home/coop>sudo userdel -r fool
c7:/home/coop>sudo rm /bin/rbash
```



### Please Note

- On some distributions, notably recent **Ubuntu**-based versions, there is a bug which prevents this lab from behaving properly. On **RedHat**-based distributions, the above correct behaviour is observed.



# Chapter 31

## Group Management



### 31.1 Labs

#### Exercise 31.1: Working with Groups

1. Create two new user accounts (`rocky` and `bullwinkle` in the below) and make sure they have home directories.
2. Create two new groups, `friends` and `bosses` (with a GID of 490). Look at the `/etc/group` file. See what GID was given to each new group.
3. Add `rocky` to both new groups.  
Add `bullwinkle` to group `friends`.  
Look in the `/etc/group` file to see how it changed.
4. Login as `rocky`. Create a directory called `somedir` and set the group ownership to `bosses`. (Using `chgrp` which will be discussed in the next session.)  
(You will probably need to add execute privileges for all on `rocky`'s home directory.)
5. Login as `bullwinkle` and try to create a file in `/home/rocky/somedir` called `somefile` using the `touch` command.  
Can you do this? No, because of the group ownership and the `chmod a+x` on the directory.
6. Add `bullwinkle` to the `bosses` group and try again. Note you will have to log out and log back in again for the new group membership to be effective. do the following:

#### Solution 31.1

1. 

```
$ sudo useradd -m rocky
$ sudo useradd -m bullwinkle
$ sudo passwd rocky

Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
$ sudo passwd bullwinkle
```

```

Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully

$ ls -l /home

total 12
drwxr-xr-x 2 bullwinkle bullwinkle 4096 Oct 30 09:39 bullwinkle
drwxr-xr-x 2 rocky rocky 4096 Oct 30 09:39 rocky
drwxr-xr-x 20 student student 4096 Oct 30 09:18 student

```

2. \$ sudo groupadd friends  
 \$ sudo groupadd -g 490 bosses  
 \$ grep -e friends -e bosses /etc/group

```

friends:x:1003:
bosses:x:490:

```

3. \$ sudo usermod -G friends,bosses rocky  
 \$ sudo usermod -G friends bullwinkle

```

$ grep -e rocky -e bullwinkle /etc/group

rocky:x:1001:
bullwinkle:x:1002:
friends:x:1003:rocky,bullwinkle
bosses:x:490:rocky

$ groups rocky bullwinkle

rocky : rocky friends bosses
bullwinkle : bullwinkle friends

```

4. \$ ssh rocky@localhost  
 \$ cd ~  
 \$ mkdir somedir  
 \$ chgrp bosses somedir  
 \$ ls -l

```

total 16
-rw-r--r-- 1 rocky rocky 8980 Oct 4 2013 examples.desktop
drwxrwxr-x 2 rocky bosses 4096 Oct 30 09:53 somedir

$ chmod a+x .

```

5. \$ ssh bullwinkle@localhost  
 \$ touch /home/rocky/somedir/somefile

```

touch: cannot touch /home/rocky/somedir/somefile: Permission denied

$ exit

```

6. \$ sudo usermod -a -G bosses bullwinkle  
 \$ ssh bullwinkle@localhost  
 \$ touch /home/rocky/somedir/somefile  
 \$ ls -al /home/rocky/somedir

(note ownership of files)

## Chapter 32

# File Permissions and Ownership



### 32.1 Labs

#### Exercise 32.1: Using chmod

One can use either the octal digit or symbolic methods for specifying permissions when using **chmod**. Let's elaborate some more on the symbolic method.

It is possible to either give permissions directly, or add or subtract permissions. The syntax is pretty obvious. Try the following examples:

```
$ chmod u=r,g=w,o=x afile
$ chmod u+=w,g=-w,o+=rw afile
$ chmod ug=rwx,o=-rw afile
```

After each step do:

```
$ ls -l afile
```

to see how the permissions took, and try some variations.

#### Exercise 32.2: umask

Create an empty file with:

```
$ touch afile
$ ls -l afile
```

```
-rw-rw-r-- 1 coop coop 0 Jul 26 12:43 afile
```

which shows it is created by default with both read and write permissions for owner and group, but only read for world.

In fact, at the operating system level the default permissions given when creating a file or directory are actually read/write for owner, group **and** world (0666); the default values have actually been modified by the current **umask**.

If you just type **umask** you get the current value:

```
$ umask
```

```
0002
```

which is the most conventional value set by system administrators for users. This value is combined with the file creation permissions to get the actual result; i.e.,

```
0666 & ~002 = 0664; i.e., rw-rw-r--
```

Try modifying the **umask** and creating new files and see the resulting permissions, as in:

```
$ umask 0022
$ touch afile2
$ umask 0666
$ touch afile3
$ ls -l afile*
```

### Exercise 32.3: Using Access Control Lists

1. Create a file using your usual user name and run **getfacl** on it to see its properties.
2. Create a new user account with default properties (or reuse one from previous exercises).
3. Login as that user and try to add a line to the file you created in the first step. This should fail.
4. User **setfacl** to make the file writeable by the new user and try again.
5. User **setfacl** to make the file not readable by the new user and try again.
6. Clean up as necessary

### Solution 32.3

It is probably easiest to open two terminal windows, one to work in as your normal user account, and the other as the secondary one.

1. In window 1:

```
$ echo This is a file > /tmp/afile
$ getfacl /tmp/afile

getfacl: Removing leading '/' from absolute path names
file: tmp/afile
owner: coop
group: coop
user::rw-
group::rw-
other::r--
```

2. In window 1:

```
$ sudo useradd fool
$ sudo passwd fool
...
```

3. In window 2:

```
$ sudo su - fool
$ echo another line > /tmp/afile

-bash: /tmp/afile: Permission denied
```

4. In window 1:

```
$ setfacl -m u:fool:rw /tmp/afile
$ getfacl /tmp/afile

getfacl: Removing leading '/' from absolute path names
file: tmp/afile
owner: coop
group: coop
user::rw-
group::rw-
other::r--
```



```
user:fool:rw-
group::rw-
mask::rwx
other::r--
```

In window 2:

```
$ echo another line > /tmp/afile
```

5. In window 1:

```
$ setfacl -m u:fool:w /tmp/afile
```

In window 2:

```
$ echo another line > /tmp/afile
-bash: /tmp/afile: Permission denied
```

6. Cleaning up:

```
$ rm /tmp/afile
$ sudo userdel -r fool
```



## Chapter 33

# Pluggable Authentication Modules (PAM)



### 33.1 Labs

#### Exercise 33.1: PAM Configuration

One of the more common **PAM** configurations is to deny login access after a certain number of failed attempts. This is done with the `pam_tally2` module. In this exercise we are going to deny login through **ssh** after three failed login attempts.

1. Edit `/etc/pam.d/ssh` and configure it to deny login after three failed attempts. Hint: add the following two lines to the file:



in `/etc/pam.d/ssh`

```
auth required pam_tally2.so deny=3 onerr=fail
account required pam_tally2.so
```

2. Try to login three times as a particular user (who has an account) while mistyping the password.
3. Try to login as the same user with the correct password.
4. Check to see how many failed logins there are for the user.
5. Reset the failed login counter.
6. Check again to see how many failed logins there are.
7. Try to login again with the correct password.

#### Solution 33.1

1. Add the following two lines to `/etc/pam.d/ssh` :

 in `/etc/pam.d/ssh`

```
auth required pam_tally2.so deny=3 onerr=fail
account required pam_tally2.so
```

2. `$ ssh student@localhost`

```
Password:
Password:
Password:
Permission denied (publickey,keyboard-interactive).
```

3. `$ ssh student@localhost`

```
Password:
Account locked due to 3 failed logins
```

4. `$ sudo pam_tally2 -u student`

| Login   | Failures | Latest failure    | From      |
|---------|----------|-------------------|-----------|
| student | 3        | 11/01/14 20:41:12 | localhost |

5. `$ sudo pam_tally2 -u student -r`

| Login   | Failures | Latest failure    | From      |
|---------|----------|-------------------|-----------|
| student | 3        | 11/01/14 20:41:12 | localhost |

6. `$ sudo pam_tally2 -u student`

| Login   | Failures | Latest failure | From |
|---------|----------|----------------|------|
| student | 0        |                |      |

7. `$ ssh student@localhost`

```
Password:
Last failed login: Sat Nov 1 20:41:14 CDT 2014 from localhost on ssh:notty
There were 6 failed login attempts since the last successful login.
Last login: Sat Nov 1 20:28:38 2014 from localhost
Have a lot of fun...
```

## Chapter 34

# Network Addresses



### 34.1 Labs



#### Please Note

There are no lab exercises in this chapter. It just sets the stage for the following section on network configuration, which has several labs.



## Chapter 35

# Network Devices and Configuration



### 35.1 Labs

#### Exercise 35.1: Static Configuration of a Network Interface



#### Please Note

You may have to use a different network interface name than `eth0`. You can most easily do this exercise with **nmtui** or your system's graphical interface. We will present a command line solution, but beware details may not exactly fit your distribution flavor or fashion.

1. Show your current IP address, default route and **DNS** settings for `eth0`. Keep a copy of them for resetting later.
2. Bring down `eth0` and reconfigure to use a static address instead of **DCHP**, using the information you just recorded.
3. Bring the interface back up, and configure the nameserver resolver with the information that you noted before.  
Verify your hostname and then **ping** it.
4. Make sure your configuration works after a reboot.

You will probably want to restore your configuration when you are done.

#### Solution 35.1

1. 

```
$ ip addr show eth0
$ ip route
$ cp /etc/resolv.conf resolv.conf.keep
$ ifconfig eth0
$ route -n
$ cp /etc/resolv.conf resolv.conf.keep
```
2. 

```
$ sudo ip link set eth0 down
```

or

```
$ sudo ifconfig eth0 down
```



### On RedHat / CentOS

Make sure the following is in `/etc/sysconfig/network-scripts/ifcfg-eth0` on **Red Hat**-based systems:



in `/etc/sysconfig/network-scripts/ifcfg-eth0`

```
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
IPADDR=noted from step 1
NETMASK=noted from step 1
GATEWAY=noted from step 1
```



### On openSUSE , SLES, OpenSUSE, and Debian-based systems

On **SUSE**-based systems edit the file in `/etc/sysconfig/network` in the same way, and on **Debian**-based systems edit `/etc/networking/interfaces` to include:



in `/etc/sysconfig/network` or `/etc/networking/interfaces`

```
iface eth0 inet static
address noted from step 1
netmask noted from step 1
gateway noted from step 1
```

3. 

```
$ sudo ip link set eth0 up
```

  
or  

```
$ sudo ifconfig eth0 up
```

```
$ sudo cp resolv.conf.keep /etc/resolv.conf
```

```
$ cat /etc/sysconfig/network
```

```
$ cat /etc/hosts
```

```
$ ping yourhostname
```
4. 

```
$ sudo reboot
```

```
$ ping hostname
```

## Exercise 35.2: Adding a Static Hostname

In this exercise we will add entries to the local host database.

1. Open `/etc/hosts` and add an entry for `mysystem.mydomain` that will point to the IP address associated with your network card.
2. Add a second entry that will make all references to `ad.doubleclick.net` point to `127.0.0.1`.
3. As an optional exercise, download the host file from: <http://winhelp2002.mvps.org/hosts2.htm> or more directly from <http://winhelp2002.mvps.org/hosts.txt>, and install it on your system. Do you notice any difference using your browser with and without the new host file in place?

## Solution 35.2

1. 

```
$ sudo sh -c "echo 192.168.1.180 mysystem.mydomain >> /etc/hosts"
```

```
$ ping mysystem.mydomain
```



```

2. $ sudo sh -c "echo 127.0.0.1 ad.doubleclick.net >> /etc/hosts"
 $ ping ad.doubleclick.net

3. $ wget http://winhelp2002.mvps.org/hosts.txt
 --2014-11-01 08:57:12-- http://winhelp2002.mvps.org/hosts.txt
 Resolving winhelp2002.mvps.org (winhelp2002.mvps.org)... 216.155.126.40
 Connecting to winhelp2002.mvps.org (winhelp2002.mvps.org)|216.155.126.40|:80... connected.
 HTTP request sent, awaiting response... 200 OK
 Length: 514744 (503K) [text/plain]
 Saving to: hosts.txt

100%[=====] 514,744 977KB/s in 0.5s

2014-11-01 08:57:13 (977 KB/s) - hosts.txt saved [514744/514744]
 $ sudo sh -c "cat hosts.txt >> /etc/hosts"

```

### Exercise 35.3: Adding a Network Interface Alias/Address using nmcli

We are going to add an additional IPv4 address to your system and make it persistent. We will do this without editing files under `/dev` directly, using **nmcli**.

1. First obtain your current internet address and interface name:

```

$ sudo nmcli con

NAME UUID TYPE DEVICE
Auto Ethernet 1c46bf37-2e4c-460d-8b20-421540f7d0e2 802-3-ethernet ens33
virbr0 a84a332f-38e3-445a-a377-4363a8eb963f bridge virbr0

```

shows the name of the connection is Auto Ethernet.

```

$ sudo nmcli con show "Auto Ethernet" | grep IP4.ADDRESS
IP4.ADDRESS[1]: 172.16.2.135/24

```

shows the address as 172.16.2.135 Note that this command shows all information about the connection and you could have specified the UUID instead of the NAME as in:

```

$ nmcli con show 1c46bf37-2e4c-460d-8b20-421540f7d0e2

```

2. Add a new address that your machine can be seen by:

```

$ sudo nmcli con modify "Auto Ethernet" +ipv4.addresses 172.16.2.140/24

```

3. Activate it and test to see if it is there:

```

$ sudo nmcli con up "Auto Ethernet"
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/6)

$ ping -c 3 172.16.2.140
PING 172.16.2.140 (172.16.2.140) 56(84) bytes of data.
64 bytes from 172.16.2.140: icmp_seq=1 ttl=64 time=0.038 ms
64 bytes from 172.16.2.140: icmp_seq=2 ttl=64 time=0.034 ms
64 bytes from 172.16.2.140: icmp_seq=3 ttl=64 time=0.032 ms

--- 172.16.2.140 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.032/0.034/0.038/0.007 ms

```

4. Clean up by removing the alias:

```

$ sudo nmcli con modify "Auto Ethernet" -ipv4.addresses 172.16.2.140/24
...
$ sudo nmcli con up "Auto Ethernet"
...

```

## Exercise 35.4: Adding a Static Route using nmcli

We are going to add a static IPv4 route address to your system and make it persistent. We will do this without editing files under `/dev` directly, using **nmcli**.

1. Begin by examining your current routing tables, using both **route** and **ip**:

```
$ route

Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 172.16.2.2 0.0.0.0 UG 100 0 0 ens33
link-local * 255.255.0.0 U 1000 0 0 ens33
172.16.2.0 * 255.255.255.0 U 100 0 0 ens33
192.168.122.0 * 255.255.255.0 U 0 0 0 virbr0

$ ip route

default via 172.16.2.2 dev ens33 proto static metric 100
169.254.0.0/16 dev ens33 scope link metric 1000
172.16.2.0/24 dev ens33 proto kernel scope link src 172.16.2.135 metric 100
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
```

2. Add a new route using **nmcli**:

```
$ sudo nmcli conn mod "Auto Ethernet" +ipv4.routes "192.168.100.0/24 172.16.2.1"
```

3. Note it has not yet taken effect:

```
$ route

Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 172.16.2.2 0.0.0.0 UG 100 0 0 ens33
link-local * 255.255.0.0 U 1000 0 0 ens33
172.16.2.0 * 255.255.255.0 U 100 0 0 ens33
192.168.122.0 * 255.255.255.0 U 0 0 0 virbr0
```

4. Reload the interface to have it take effect and show it has:

```
$ sudo nmcli conn up "Auto Ethernet"

Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/25)

$ route

Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 172.16.2.2 0.0.0.0 UG 100 0 0 ens33
link-local * 255.255.0.0 U 1000 0 0 ens33
172.16.2.0 * 255.255.255.0 U 100 0 0 ens33
192.168.100.0 172.16.2.1 255.255.255.0 UG 100 0 0 ens33
192.168.122.0 * 255.255.255.0 U 0 0 0 virbr0
```

5. Reboot and verify the route has taken effect (i.e., it is **persistent**: If so remove it:

```
$ route

Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 172.16.2.2 0.0.0.0 UG 100 0 0 ens33
link-local * 255.255.0.0 U 1000 0 0 ens33
172.16.2.0 * 255.255.255.0 U 100 0 0 ens33
192.168.100.0 172.16.2.1 255.255.255.0 UG 100 0 0 ens33
192.168.122.0 * 255.255.255.0 U 0 0 0 virbr0

$ sudo nmcli conn mod "Auto Ethernet" -ipv4.routes "192.168.100.0/24 172.16.2.1"

$ sudo nmcli conn up "Auto Ethernet"
```

```
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/3)
$ route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 172.16.2.2 0.0.0.0 UG 100 0 0 ens33
link-local * 255.255.0.0 U 1000 0 0 ens33
172.16.2.0 * 255.255.255.0 U 100 0 0 ens33
192.168.122.0 * 255.255.255.0 U 0 0 0 virbr0
```

6. Note you can set a route with either **route** or **ip** from the command line but it won't survive a reboot as in:

```
$ sudo ip route add 192.168.100.0/24 via 172.16.2.1
$ sudo route
....
```

You can verify that a route established this way is not persistent.



# Chapter 36

## Firewalls



### 36.1 Labs

#### Exercise 36.1: Installing firewalld

While most recent **Linux** distributions have the **firewalld** package (which includes the **firewall-cmd** multi-purpose utility) available, it might not be installed on your system.

First you should check to see if it is already installed, with

```
$ which firewalld firewall-cmd

/usr/sbin/firewalld
/usr/bin/firewall-cmd
```

If you fail to find the program, then you need to install with one of the following, depending on your distribution in the usual way:

```
$ sudo yum install firewalld
$ sudo zypper install firewalld
$ sudo apt install firewalld
```

If this fails, the **firewalld** package is not available for your distribution. For example, this will be the case for the older **RHEL6/CentOS6** distributions. In this case you will have to install from source.

To do this, go to <https://fedorahosted.org/firewalld/> and you can get the **git** source repository, but it is much easier to download the most recent tarball ([firewalld-0.3.13.tar.bz2](#) as of this writing.)

Then you have to follow the common procedure for installing from source:

```
$ tar xvf firewalld-0.3.13.tar.bz2
$ cd firewalld-0.3.13
$./configure
$ make
$ sudo make install
```

Note this source also has an **uninstall** target:

```
$ sudo make uninstall
```

in case you have regrets.

You will have to deal with any inadequacies that come up in the `./configure` step, such as missing libraries etc. When you install from a packaging system, the distribution takes care of this for you, but from source it can be problematic. If you have run the **Linux Foundation's** `ready-for.sh` script on your system, you are unlikely to have problems.



### On openSUSE

**Note:** On **openSUSE 13.2**, even though the compile and install of **firewalld** will work, execution of **firewall-cmd** will still fail with a message about missing `python-slip`. Unfortunately, this package also doesn't exist in the **zypper** repositories, so you will have to download it from the same web site, <https://fedorahosted.org/firewalld/>, and then just do:

```
$ tar xvf /tmp/python-slip-0.6.1.tar.bz2
$ cd python-slip-0.6.1
$ make
$ sudo make install
```

substituting the actual name of the version you downloaded. Hopefully, the next edition of **openSUSE** will eliminate this need to compile from sources, as there have been requests to add **firewalld** properly to the available choices.

## Exercise 36.2: Examining firewall-cmd

We have only scratched the surface of how you can use the **firewalld** package. Almost everything is done by deploying **firewall-cmd** which is empowered to do a large variety of tasks, using options with very clear names.

To get a sense of this, there is really no substitute for just doing:

```
$ firewall-cmd --help

Usage: firewall-cmd [OPTIONS...]
....
Service Options
 --new-service=<service>
 Add a new service [P only]
 --delete-service=<service>
 Delete and existing service [P only]
....
```

which we will not reproduce here as it is 208 lines on a **RHEL 7** system.

For more detailed explanation of anything which piques your interest, do **man firewall-cmd** which explains things more deeply, and **man firewalld** which gives an overview, as well as a listing of other **man** pages that describe the various configuration files in `/etc`, and elucidate concepts such as **zones** and **services**.

## Exercise 36.3: Adding Services to a Zone

Add the **http** and **https** services to the **public zone** and verify that they are currently listed.

### Solution 36.3

```
$ sudo firewall-cmd --zone=public --add-service=http
success

$ sudo firewall-cmd --zone=public --add-service=https
success

$ sudo firewall-cmd --list-services --zone=public
dhcpv6-client http https ssh
```

Note if you had run

```
$ sudo firewall-cmd --reload
$ sudo firewall-cmd --list-services --zone=public
```

```
dhcpcv6-client ssh
```

after adding the new services, they would disappear from the list! This curious behavior is because we did not include the `--permanent` flag when adding the services, and the `--reload` option reloads the known persistent services only.

### Exercise 36.4: Using the firewall GUI

Each distribution has its own graphical interface for firewall administration. On **Red Hat**-based systems you can run **firewall-config**, on **Ubuntu** it is called **gufw**, and on **openSUSE** you can find it as part of **yast** on the graphical menu system.

We have concentrated on the command line approach simply because we want to be distribution-flexible. However, for most relatively simple firewall configuration tasks, you can probably do them efficiently with less memorization from the GUI.

Once you launch the firewall configuration GUI, do the previous exercise of adding **http** and **https** to the **public** zone, and verify that it has taken effect.

Make sure you take the time to understand the graphical interface.





## Chapter 37

# System Startup and Shutdown



### 37.1 Labs

#### Exercise 37.1: Shutdown VS. Halt VS. Reboot

NOTE: This exercise requires that it be run from the console (i.e., not over the network through SSH).

1. Reboot the system using **shutdown**.
2. Power off the system using **shutdown**.
3. Power the system back up.

#### Solution 37.1

1. `$ sudo shutdown -r now`
2. `$ sudo shutdown -h now`
3. Press the power button, or restart your virtual machine.



# Chapter 38

## GRUB



### 38.1 Labs

#### Exercise 38.1: Booting into Non-Graphical Mode Using GRUB



#### Please Note

This exercise requires that it be run from the console (i.e., not over **SSH**).

1. Reboot your machine and go into the **GRUB** interactive shell by hitting **e** (or whatever other key is required as listed on your screen.)
2. Make your system boot into non-graphical mode. How you do this depends on the system.  
On traditional systems that respect **runlevels** (which we will talk about in the next section) you can append a 3 to the kernel command line in the specific entry you pick from the **GRUB** menu of choices. This will still work on more recent **systemd** machines that emulate **SysVinit** runlevels.  
On some other systems you may need to append `text` instead.
3. Hit the proper key to make system continue booting.
4. After the system is fully operational in non-graphical mode, bring it up to graphical mode. Depending on your system, one of the following commands should do it:

```
$ sudo systemctl start gdm
$ sudo systemctl start lightdm
$ sudo telinit 5
$ sudo service gdm restart
$ sudo service lightdm restart
```



## Chapter 39

# Init, SystemV, Upstart, systemd



### 39.1 Labs

#### Exercise 39.1: Adding a New Startup Service with SysVinit

In this and the following exercise, we will create a simple startup service. First we will do it for a **SysVinit** system. Note that if you are using a **systemd**-based system everything should still work because of the backwards compatibility layer that all distributions utilize. However, in the next exercise we will do natively for **systemd**.



#### On Debian -based Distributions

If you are on a **Debian**-based system like **Ubuntu**, make sure you have installed the **sysvinit-utils** and **chkconfig** packages. However, recent versions of **Ubuntu** no longer package **chkconfig**; you'll have to use the **update-rc.d** utility instead.

First we have to create the service-specific script; you can create one of your own for fun, or to get the procedure down just (as root) create a file named `/etc/init.d/fake_service` (which can be extract from your downloaded SOLUTIONS file as `fake_service`) containing the following:

SH

#### `/etc/init.d/fake_service`

```
#!/bin/bash
fake_service
Starts up, writes to a dummy file, and exits
#
chkconfig: 35 69 31
description: This service doesn't do anything.
Source function library

. /etc/sysconfig/fake_service

case "$1" in
 start) echo "Running fake_service in start mode..."
```

SH

```

touch /var/lock/subsys/fake_service
echo "$0 start at $(date)" >> /var/log/fake_service.log
if [${VAR1} = "true"]
then
 echo "VAR1 set to true" >> /var/log/fake_service.log
fi
echo
;;
stop)
echo "Running the fake_service script in stop mode..."
echo "$0 stop at $(date)" >> /var/log/fake_service.log
if [${VAR2} = "true"]
then
 echo "VAR2 = true" >> /var/log/fake_service.log
fi
rm -f /var/lock/subsys/fake_service
echo
;;
*)
echo "Usage: fake_service {start | stop}"
exit 1
esac
exit 0

```

If you are taking the online self-paced version of this course, the script is available for download from your **Lab** screen.

Make the file above executable and give other proper permissions:

```
$ sudo chmod 755 /etc/init.d/fake_service
```

You'll notice the script includes the file `/etc/sysconfig/fake_service`. (On non-**RHEL** systems you should change this to `/etc/default/fake_service`.) Create it and give it the following contents:

SH

`/etc/sysconfig/fake_service` or `/etc/default/fake_service`

```

VAR1= "true"
VAR2= "true"

```

Test to see if the script works properly by running the following commands:

```

$ sudo service fake_service
$ sudo service fake_service start
$ sudo service fake_service stop

```

Look at the file named `/var/log/fake_service.log`. What does it contain?

For fun you can add additional modes like `restart` to the script file; look at other scripts in the directory to get examples of what to do.

Next we will want to have the ability to start **fake\_service** whenever the system starts, and stop it when it shuts down. If you do:

```
$ sudo chkconfig --list fake_service
```

you will get an error as it hasn't been set up yet for this. You can easily do this with:

```
$ sudo chkconfig --add fake_service
```

and you can turn it on or off at boot time with

```
$ sudo chkconfig fake_service on
$ sudo chkconfig fake_service off
```

To test this completely you'll have to reboot the system to see if it comes on automatically. You can also try varying the runlevels in which the service is running.

## Exercise 39.2: Adding a New Startup Service with systemd

As mentioned in the previous exercise, you can still use the **SysVinit** startup script procedure with **systemd** but this is deprecated.

The analogous procedure is to create (as root) a file directly under `/etc/systemd/system` or somewhere else in that directory tree; distributions have some varying tastes on this. For example a very minimal file named `/etc/systemd/system/fake2.service` (which can be extracted from your downloaded SOLUTIONS file as `fake2.service`) containing the following:

fake2.service

```
[Unit]
Description=fake2
After=network.target

[Service]
ExecStart=/bin/sh -c '/bin/echo I am starting the fake2 service ; /bin/sleep 30'
ExecStop=/bin/echo I am stopping the fake2 service

[Install]
WantedBy=multi-user.target
```

Now there are many things that can go in this **unit** file. The `After=network.target` means the service should start only after the network does, while the `WantedBy=multi-user.target` means it should start when we reach multiple-user mode. This is equivalent to runlevels 2 and 3 in **SysVinit**. Note `graphical.target` would correlate with runlevel 5.

Now all we have to do to start, stop and check the service status are to issue the commands:

```
$ sudo systemctl start fake2.service
$ sudo systemctl status fake2.service
$ sudo systemctl stop fake2.service
```

If you are fiddling with the unit file while doing this you'll need to reload things with:

```
$ sudo systemctl daemon-reload
```

as the system will warn you.

To keep an eye directly on the output you can do:

```
$ sudo tail -f /var/log/messages
```

(use `/var/log/syslog` on **Ubuntu**) either in background or in another windows while the service is running.

To set things up so the service turns on or off on system boot:

```
$ sudo systemctl enable fake2.service
$ sudo systemctl disable fake2.service
```

Once again, you really need to reboot to make sure it has taken effect.





## Chapter 40

# Backup and Recovery Methods



### 40.1 Labs

#### Exercise 40.1: Using tar for Backup

1. Create a directory called `backup` and in it place a compressed **tar** archive of all the files under `/usr/include`, with the highest level directory being `include`. You can use any compression method (**gzip**, **bzip2** or **xzip**).
2. List the files in the archive.
3. Create a directory called `restore` and unpack and decompress the archive.
4. Compare the contents with the original directory the archive was made from.

#### Solution 40.1

1. 

```
$ mkdir /tmp/backup
$ cd /usr ; tar zcvf /tmp/backup/include.tar.gz include
$ cd /usr ; tar jcvf /tmp/backup/include.tar.bz2 include
$ cd /usr ; tar Jcvf /tmp/backup/include.tar.xz include
```

or

```
$ tar -C /usr -zcf include.tar.gz include
$ tar -C /usr -jcf include.tar.bz2 include
$ tar -C /usr -Jcf include.tar.xz include
```

Notice the efficacy of the compression between the three methods:

```
$ du -sh /usr/include
```

```
55M /usr/include
```

2. 

```
$ ls -lh include.tar.*

c7:/tmp/backup>ls -lh
total 17M
-rw-rw-r-- 1 coop coop 5.3M Jul 18 08:17 include.tar.bz2
-rw-rw-r-- 1 coop coop 6.7M Jul 18 08:16 include.tar.gz
-rw-rw-r-- 1 coop coop 4.5M Jul 18 08:18 include.tar.xz
c7:/tmp/backup>
```

3. `$ tar tvf include.tar.xz`

```
qdrwxr-xr-x root/root 0 2014-10-29 07:04 include/
-rw-r--r-- root/root 42780 2014-08-26 12:24 include/unistd.h
-rw-r--r-- root/root 957 2014-08-26 12:24 include/re_comp.h
-rw-r--r-- root/root 22096 2014-08-26 12:24 include/regex.h
-rw-r--r-- root/root 7154 2014-08-26 12:25 include/link.h
.....
```

Note it is not necessary to give the `j`, `J`, or `z` option when decompressing; **tar** is smart enough to figure out what is needed.

4. `$ cd .. ; mkdir restore ; cd restore`  
`$ tar xvf ../backup/include.tar.bz2`

```
include/
include/unistd.h
include/re_comp.h
include/regex.h
include/link
.....
$ diff -qr include /usr/include
```

## Exercise 40.2: Using cpio for Backup

We are going to do essentially the same exercise now, but using **cpio** in place of **tar**. We'll repeat the slightly altered instructions for ease of use.

1. Create a directory called `backup` and in it place a compressed **cpio** archive of all the files under `/usr/include`, with the highest level directory being `include`. You can use any compression method (**gzip**, **bzip2** or **xzip**).
2. List the files in the archive.
3. Create a directory called `restore` and unpack and decompress the archive.
4. Compare the contents with the original directory the archive was made from.

## Solution 40.2

1. `$ (cd /usr ; find include | cpio -c -o > /home/student/backup/include.cpio)`

```
82318 blocks
```

or to put it in a compressed form:

```
$ (cd /usr ; find include | cpio -c -o | gzip -c > /home/student/backup/include.cpio.gz)
```

```
82318 blocks
```

```
$ ls -lh include*
```

```
total 64M
-rw-rw-r-- 1 coop coop 41M Nov 3 15:26 include.cpio
-rw-rw-r-- 1 coop coop 6.7M Nov 3 15:28 include.cpio.gz
-rw-rw-r-- 1 coop coop 5.3M Nov 3 14:44 include.tar.bz2
-rw-rw-r-- 1 coop coop 6.8M Nov 3 14:44 include.tar.gz
-rw-rw-r-- 1 coop coop 4.7M Nov 3 14:46 include.tar.xz
```

2. `$ cpio -ivt < include.cpio`

```
drwxr-xr-x 86 root root 0 Oct 29 07:04 include
-rw-r--r-- 1 root root 42780 Aug 26 12:24 include/unistd.h
-rw-r--r-- 1 root root 957 Aug 26 12:24 include/re_comp.h
-rw-r--r-- 1 root root 22096 Aug 26 12:24 include/regex.h
.....
```

Note the redirection of input; the archive is not an argument. One could also do:

```

$ cd ../restore
$ cat ../backup/include.cpio | cpio -ivt
$ gunzip -c include.cpio.gz | cpio -ivt

3. $ rm -rf include
$ cpio -id < ../backup/include.cpio
$ ls -lR include

or

$ cpio -idv < ../backup/include.cpio

$ diff -qr include /usr/include

```

### Exercise 40.3: Using rsync for Backup

1. Using **rsync**, we will again create a complete copy of `/usr/include` in your backup directory:

```

$ rm -rf include
$ rsync -av /usr/include .

sending incremental file list
include/
include/FlexLexer.h
include/_G_config.h
include/a.out.h
include/aio.h
.....

```

2. Let's run the command a second time and see if it does anything:

```

$ rsync -av /usr/include .

sending incremental file list

sent 127398 bytes received 188 bytes 255172.00 bytes/sec
total size is 41239979 speedup is 323.23

```

3. One confusing thing about **rsync** is you might have expected the right command to be:

```

$ rsync -av /usr/include include

sending incremental file list
...

```

However, if you do this, you'll find it actually creates a new directory, `include/include`!

4. To get rid of the extra files you can use the `--delete` option:

```

$ rsync -av --delete /usr/include .

sending incremental file list
include/
deleting include/include/xen/privcmd.h
deleting include/include/xen/evtchn.h
....
deleting include/include/FlexLexer.h
deleting include/include/

sent 127401 bytes received 191 bytes 85061.33 bytes/sec
total size is 41239979 speedup is 323.22

```

5. For another simple exercise, remove a subdirectory tree in your backup copy and then run **rsync** again with and without the `--dry-run` option:

```

$ rm -rf include/xen
$ rsync -av --delete --dry-run /usr/include .

```

```
sending incremental file list
include/
include/xen/
include/xen/evtchn.h
include/xen/privcmd.h

sent 127412 bytes received 202 bytes 255228.00 bytes/sec
total size is 41239979 speedup is 323.16 (DRY RUN)

$ rsync -av --delete /usr/include .
```

6. A simple script with a good set of options for using **rsync**:

#### script using rsync

```
#!/bin/sh
set -x

rsync --progress -avrxH -e "ssh -c blowfish" --delete $*
```

which will work on a local machine as well as over the network. Note the important `-x` option which stops **rsync** from crossing filesystem boundaries.

#### Extra Credit

For more fun, if you have access to more than one computer, try doing these steps with source and destination on different machines.

# Chapter 41

## Linux Security Modules



### 41.1 Labs

#### Exercise 41.1: SELinux: Contexts



#### Please Note

This exercise can only be performed on a system (such as **RHEL**) where **SELinux** is installed. While it is possible to install on **Debian**-based distributions, such as **Ubuntu**, it is not the easiest task and it is not often done.

1. Verify **SELinux** is enabled and in **enforcing** mode, by executing **getenforce** and **sestatus**. If not, edit `/etc/selinux/config`, reboot, and check again.
2. Install the **httpd** package (if not already present) which provides the **Apache** web server, and then verify that it is working:

```
$ sudo yum install httpd
$ elinks http://localhost
```

(You can also use **lynx** or **firefox** etc. as the browser, or use your graphical browser such as **firefox** or **chrome**, in this and succeeding steps.)

3. As superuser, create a small file in `/var/www/html`:  

```
$ sudo sh -c "echo file1 > /var/www/html/file1.html"
```

4. Verify you can see it:

```
$ elinks -dump http://localhost/file1.html
file1
```

Now create another small file in **root**'s home directory and **move** it to `/var/www/html`. (Do not copy it, move it!) Then try and view it:

```
$ sudo cd /root
$ sudo sh -c "echo file2 > file2.html"
$ sudo mv file2.html /var/www/html
$ elinks -dump http://localhost/file2.html
```

Forbidden

You don't have permission to access /file2.html on this server.

5. Examine the security contexts:

```
$ cd /var/www/html
$ ls -Z file*html

-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 file1.html
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 file2.html
```

6. Change the offending context and view again:

```
$ sudo chcon -t httpd_sys_content_t file2.html
$ elinks http://localhost/file2.html

file2
```

## Exercise 41.2: Explore the apparmor security



### Please Note

This exercise can only be performed on a system (such as **Ubuntu**) where **AppArmor** is installed.

The below was tested on **Ubuntu 17.04**, but should work on other **AppArmor**-enabled systems, such as **OpenSUSE**, where the **apt** commands should be replaced by **zypper**.

On **Ubuntu**, the **/bin/ping** utility runs with SUID enabled. For this exercise, we will copy **ping** to **ping-x** and adjust the capabilities so the program functions.

Then we will build an **AppArmor** profile, install and verify that nothing has changed. Modifying the **AppArmor** profile and adding capabilities will allow the program more functionality.

1. Make sure all necessary packages are installed:

```
student@ubuntu:~$ sudo apt install apparm*
```

2. Create a copy of **ping** (called **ping-x**) and verify it has no initial special permissions or capabilities. Furthermore, it cannot work when executed by **student**, a normal user:

```
student@ubuntu:~$ sudo cp /bin/ping /bin/ping-x
student@ubuntu:~$ sudo ls -l /bin/ping-x

-rwxr-xr-x 1 root root 64424 Oct 17 10:12 /bin/ping-x

student@ubuntu:~$ sudo getcap /bin/ping-x
student@ubuntu:~$

student@ubuntu:~$ ping-x -c3 -4 127.0.0.1
ping: socket: Operation not permitted
```

3. Set the **capabilities** and re-try **ping-x**:

```
student@ubuntu:~$ sudo setcap cap_net_raw+ep /bin/ping-x

student@ubuntu:~$ ping-x -c3 -4 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.092 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.093 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.086 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2034ms
rtt min/avg/max/mdev = 0.086/0.090/0.093/0.008 ms
```

The modified **ping-x** program now functions normally.

4. Verify there is no pre-existing **AppArmor** profile for **ping-x**, but there is a profile for **ping**. Determine the status of the current **ping** program:

```
student@ubuntu:~$ sudo aa-status
```

The output from **aa-status** is long, so we can **grep** for the interesting lines:

```
student@ubuntu:~$ sudo aa-status | grep -e "[[:alnum:]]" -e ping
```

```
apparmor module is loaded.
87 profiles are loaded.
51 profiles are in enforce mode.
 ping
36 profiles are in complain mode.
17 processes have profiles defined.
6 processes are in enforce mode.
11 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

We can see **ping** has a profile that is loaded and enabled for enforcement.

5. Next we will construct a new profile for **ping-x**. This step requires two terminal windows.

The first window (**window1**) will be running the **aa-genprof** command. This will generate a **AppArmor** profile by scanning **/var/log/syslog** for **AppArmor** errors.

The second window (**window2**) will be used to run **ping-x**. (See the **man** page for **aa-genprof** for additional information.)

In **window1**:

```
student@ubuntu:~$ sudo aa-genprof /bin/ping-x
Writing updated profile for /bin/ping-x.
Setting /bin/ping-x to complain mode.
```

Before you begin, you may wish to check if a profile already exists for the application you wish to confine. See the following wiki page for more information:  
<http://wiki.apparmor.net/index.php/Profiles>

Please start the application to be profiled in another window and exercise its functionality now.

Once completed, select the "Scan" option below in order to scan the system logs for AppArmor events.

For each AppArmor event, you will be given the opportunity to choose whether the access should be allowed or denied.

Profiling: /bin/ping-x

[(S)can system log for AppArmor events] / (F)inish

In **window2**:

```
student@ubuntu:~$ ping-x -c3 -4 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.099 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.120 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.114 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2041ms
rtt min/avg/max/mdev = 0.099/0.111/0.120/0.008 ms
```

In **window1**:

The command `ping-x` has completed, we must now instruct **aa-genprof** to scan for the required information to be added to the profile. It may require several **scans** to collect all of the information for the profile.

Enter **S** to scan:

```
Reading log entries from /var/log/syslog.
Updating AppArmor profiles in /etc/apparmor.d.
Complain-mode changes:
```

```
Profile: /bin/ping-x
Capability: net_raw
Severity: 8
```

```
[1 - capability net_raw,]
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

Enter **A** to allow the capability:

Adding capability `net_raw`, to profile.

```
Profile: /bin/ping-x
Network Family: inet
Socket Type: raw
```

```
[1 - network inet raw,]
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

Enter **A** to allow the network family:

Adding network `inet raw`, to profile.

```
Profile: /bin/ping-x
Network Family: inet
Socket Type: dgram
```

```
[1 - #include <abstractions/nameservice>]
2 - network inet dgram,
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

Enter **A** to add the socket type datagram to the profile:

Adding `#include <abstractions/nameservice>` to profile.

= Changed Local Profiles =

The following local profiles were changed. Would you like to save them?

```
[1 - /bin/ping-x]
(S)ave Changes / Save Selec(t)ed Profile / [(V)iew Changes] / View Changes b/w (C)lean profiles / Abo(r)t
```

Enter **S** to save the new profile:

Writing updated profile for `/bin/ping-x`.

Profiling: `/bin/ping-x`

```
[(S)can system log for AppArmor events] / (F)inish
```

Enter **F** to finish:

Setting `/bin/ping-x` to enforce mode.

Reloaded AppArmor profiles in enforce mode.

Please consider contributing your new profile!  
See the following wiki page for more information:  
<http://wiki.apparmor.net/index.php/Profiles>



Finished generating profile for /bin/ping-x.

6. View the created profile, which has been stored in `/etc/apparmor.d/bin.ping-x`.

```
student@ubuntu:~$ sudo cat /etc/apparmor.d/bin.ping-x
Last Modified: Tue Oct 17 11:30:47 2017
#include <tunables/global>

/bin/ping-x
#include <abstractions/base>
#include <abstractions/namespace>

capability net_raw,

network inet raw,

/bin/ping-x mr,
/lib/x86_64-linux-gnu/ld-*.so mr,
```

7. The **aa-genproc** utility installs and activates the new policy so it should be ready to use, and the policies can be reloaded on demand with the `systemctl reload apparmor` command. To avoid any potential issues, and verify the changes will survive, reboot the system.

Once the system has restarted, as the user `student`, verify **ping-x** still functions with the new profile enabled. Ping the localhost by ip address:

```
student@ubuntu:~$ ping-x -c3 -4 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.095 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.043/0.065/0.095/0.021 ms
```

8. This should work as expected. The profile is very specific, and **AppArmor** will not allow functionality outside of the specified parameters. To verify **AppArmor** is protecting this application, try to ping the **IPv6** localhost address.

This should fail:

```
student@ubuntu:~$ ping-x -c3 -6 ::1
ping: socket: Permission denied
```

(Note, the `-6` option means use only **IPv6** and `::1` is the local host in **IPv6**.)

The output indicates there is a socket issue. If the system log is examined it will be discovered that our **ping-x** program has no access to **IPv6** within **AppArmor**:

```
766:104): apparmor="DENIED" operation="create" profile="/bin/ping-x"
pid=2709 comm="ping-x" family="inet6" sock_type="raw" protocol=58
requested_mask="create" denied_mask="create"
```

9. To correct this deficiency, re-run **aa-genprof** as we did earlier, and in **window2**, ping the **IPv6** loopback and append the additional options



## Chapter 42

# Local System Security



### 42.1 Labs

#### Exercise 42.1: Security and Mount Options

We are going to mount a partition or loop device with the `noexec` option to prevent execution of programs that reside on the filesystem therein. You can certainly do this with a pre-existing and mounted partition, but you may not be able to easily change the behavior while the partition is mounted. Therefore, to demonstrate we'll use a loop device, which is a harmless procedure.

1. Set up an empty file, put a filesystem on it and mount it.
2. Copy an executable file to it from somewhere else on your system and test that it works in the new location.
3. Unmount it and remount with the `noexec` option.
4. Test if the executable still works. It should give you an error because of the `noexec` mount option.
5. Clean up.

#### Solution 42.1

1. 

```
$ dd if=/dev/zero of=image bs=1M count=100
$ sudo mkfs.ext3 image
$ mkdir mountpoint
$ sudo mount -o loop image mountpoint
```
2. 

```
$ sudo cp /bin/ls mountpoint
$ mountpoint/ls
```
3. 

```
$ sudo umount mountpoint
$ sudo mount -o noexec,loop image mountpoint
```

or

```
$ sudo mount -o noexec,remount image mountpoint
```
4. 

```
$ mountpoint/ls
```

```
5. $ sudo umount mountpoint
 $ rm image
 $ rmdir mountpoint
```

Note that this is not persistent. To make it persistent you would need to add the option to `/etc/fstab` with a line like:

in `/etc/fstab`

```
/home/student/image /home/student/mountpoint ext3 loop,rw,noexec 0 0
```

## Exercise 42.2: More on setuid and Scripts

Suppose we have the following **C** program (`./writeit.c`) which attempts to overwrite a file in the current directory named `afile`. This file can be extracted from your downloaded SOLUTIONS file as `writeit.c`.

`writeit.c`

```
1 /*
2 @*/
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <stdlib.h>
9 #include <sys/stat.h>
10
11 int main(int argc, char *argv[])
12 {
13 int fd, rc;
14 char *buffer = "TESTING A WRITE" ;
15 fd = open("./afile", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
16 rc = write(fd, buffer, strlen(buffer));
17 printf("wrote %d bytes \n ", rc);
18 close(fd);
19 exit(EXIT_SUCCESS);
20 }
```

If the program is called `writeit.c`, it can be compiled simply by doing:

```
$ make writeit
```

or equivalently

```
$ gcc -o writeit writeit.c
```

If (as a normal user) you try to run this program on a file owned by root you'll get

```
$ sudo touch afile
$./writeit
```

```
wrote -1 bytes
```

but if you run it as root:

```
$ sudo ./writeit
```

```
wrote 15 bytes
```

Thus, the root user was able to overwrite the file it owned, but a normal user could not.

Note that changing the owner of **writeit** to root does not help:

```
$ sudo chown root.root writeit
$./writeit
```

```
wrote -1 bytes
```

because it still will not let you clobber **afile**.

By setting the **setuid** bit you can make any normal user capable of doing it:

```
$ sudo chmod +s writeit
$./writeit
```

```
wrote 15 bytes
```



### Please Note

You may be asking, why didn't we just write a script to do such an operation, rather than to write and compile an executable program?

Under **Linux**, if you change the **setuid** bit on such an executable script, it won't do anything unless you actually change the **setuid** bit on the shell (such as **bash**) which would be a big mistake; anything running from then on would have escalated privilege!



## Chapter 43

# Basic Troubleshooting



### 43.1 Labs



#### Please Note

There are no lab exercises for in this chapter. It just summarizes points discussed earlier when considering configuring and monitoring the system, and in addition, sets the stage for the following section on system rescue, which has several labs.





# Chapter 44

## System Rescue



### 44.1 Labs

#### Exercise 44.1: Preparing to use Rescue/Recover Media



#### Very Important

In the following exercises we are going to deliberately damage the system and then recover through the use of rescue media. Thus, it is obviously prudent to make sure you can indeed boot off the rescue media before you try anything more ambitious.

So first make sure you have rescue media, either a dedicated rescue/recovery image, or an install or Live image on either an optical disk or usb drive.

Boot off it and make sure you know how to force the system to boot off the rescue media (you are likely to have to fiddle with the **BIOS** settings), and when the system boots, choose rescue mode.



#### Please Note

If you are using a virtual machine, the procedure is logically the same with two differences:

- Getting to the **BIOS** might be difficult depending on the hypervisor you use. Some of them require very rapid keystrokes, so read the documentation and make sure you know how to do it.
- You can use a physical optical disk or drive, making sure the virtual machine settings have it mounted, and if it is **USB** you may have some other hurdles to make sure the virtual machine can claim the physical device. It is usually easier to simply connect a `.iso` image file directly to the virtual machine.

If you are working with a virtual machine, obviously things are less dangerous, and if you are afraid of corrupting the system in an unfixable way, simply make a backup copy of the virtual machine image before you do these exercises, you can always replace the image with it later.

**Very Important**

Do not do the following exercises unless you are sure you can boot your system off rescue/recovery media!

## ✎ Exercise 44.2: Recovering from a Corrupted GRUB Configuration

1. Edit your **GRUB** configuration file (`/boot/grub/grub.cfg`, `/boot/grub2/grub.cfg` or `/boot/grub/grub.conf`), and modify the `kernel` line by removing the first character of the value in the field named `UUID`. Take note of which character you removed, you will replace it in rescue mode. (If your root filesystem is identified by either label or hard disk device node, make an analogous simple change.) Keep a backup copy of the original.
2. Reboot the machine. The system will fail to boot, saying something like `No root device was found`. You will also see that a panic occurred.
3. Insert into your machine the **installation** or **Live DVD** or **CD** or **USB** drive (or network boot media) if you have access to a functioning installation server). Reboot again. When the boot menu appears, choose to enter rescue mode.
4. As an alternative, you can try selecting a rescue image from the **GRUB** menu; most distributions offer this. You'll get the same experience as using rescue media, but it will not always work. For example, if the root filesystem is damaged it will be impossible to do anything.
5. In rescue mode, agree when asked to search for filesystems. If prompted, open a shell, and explore the rescue system by running utilities such as **mount** and **ps**.
6. Repair your broken system by fixing your **GRUB** configuration file, either by editing it or restoring from a backup copy.
7. Type `exit` to return to the installer, remove the boot media, and follow the instructions on how to reboot. Reboot your machine. It should come up normally.

## ✎ Exercise 44.3: Recovering from Password Failure

1. As root (not with **sudo**), change the root password. We will pretend we don't know what the new password is.
2. Log out and try to login again as root using the old password. Obviously you will fail.
3. Boot using the rescue media, and select **Rescue** when given the option. Let it mount filesystems and then go to a command line shell.
4. Go into your **chroot**-ed environment (so you have normal access to your systems):  

```
$ chroot /mnt/sysimage
```

and reset the root password back to its original value.
5. Exit, remove the rescue media, and reboot, you should be able to login normally now.

## ✎ Exercise 44.4: Recovering from Partition Table Corruption

**Very Important**

This exercise is dangerous and could leave to an unusable system. Make sure you really understand things before doing it

**Please Note**

The following instructions for an **MBR** system. if you have **GPT** you need to use **sgdisk** with the `--backup-file` and `--load-backup` options as discussed in the partitioning chapter

1. Login as root and save your **MBR**:

```
$ dd if=/dev/sda of=/root/mbrsave bs=446 count=1
1+0 records in
1+0 records out
446 bytes (446 B) copied, 0.00976759 s, 45.7 kB/s
```

Be careful: make sure you issue the exact command above and that the file saved has the right length:

```
$ sudo ls -l /root/mbrsave
-rw-r--r-- 1 root root 446 Nov 12 07:54 mbrsave
```

2. Now we are going to obliterate the **MBR** with:

```
$ dd if=/dev/zero of=/dev/sda bs=446 count=1
1+0 records in
1+0 records out
446 bytes (446 B) copied, 0.000124091 s, 3.6 MB/s
```

3. Reboot the system; it should fail.
4. Reboot into the rescue environment and restore the **MBR**:  

```
$ dd if=/mnt/sysimage/root/mbrsave of=/dev/sda bs=446 count=1
```
5. Exit from the rescue environment and reboot. The system should boot properly now.

## Exercise 44.5: Recovering Using the Install Image



### Please Note

This exercise has been specifically written for **Red Hat**-based systems. You should be able to easily construct the appropriate substitutions for other distribution families.

1. Remove the **zsh** package (if it is installed!):

```
$ yum remove zsh
```

or

```
$ rpm -e zsh
```

Note we have chosen a package that generally has no dependencies to simplify matters. If you choose something that does, you will have to watch your step in the below so that anything else you remove you reinstall as needed as well.

2. Boot into the rescue environment.
3. Re-install (or install) **zsh** from within the rescue environment. First, mount the install media at `/mnt/source`:

```
$ mount /dev/cdrom /mnt/source
```

Then reinstall the package:

```
$ rpm -ivh --force --root /mnt/sysimage /mnt/source/Packages/zsh*.rpm
```

The `--force` option tells **rpm** to use the source directory in determining dependency information etc. Note that if the install image is much older than your system which has had many updates the whole procedure might collapse!

4. Exit and reboot.
5. Check that **zsh** has been reinstalled:

```
$ rpm -q zsh
zsh-5.0.2-7.el7.x86_64
```

6. `$ zsh`

```
....
[coop@q7]/tmp/LFS201%
```