Mainstays of management: people and data

Managing users and data are two of the most important categories of system administration duties. We will look at what it means to add and remove both people and data.

Adding local users: Unix/Linux

☞ Straightforward, but tedious

Adding local users: Unix

⇶ A number of C library calls (getpwent(), etc.) exist to access entries in the password file (/etc/passwd). Many UNIX commands depend on the file being available, readable, with the proper format.

⇶ Create an entry in /etc/passwd, selecting a unique login name, unique UID, appropriate GID, unique home directory and appropriate shell.

Adding local users: Unix

⇶ Older Unix/Linux systems limited username to 8 characters – newer ones often don't, but some tools still only show 8 characters. For instance, look at the difference in **w** and **who** output for long usernames:

Adding local users: Unix

```
$ w
 06:44:59 up 2 days,  1:45,  6 users,  load average: 0.00, 0.00, 0.05
USER      TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
testtest tty2      -                 06:41    2:41   0.00s  0.00s -bash
testtest tty3      -                 06:42    1:57   0.00s  0.00s -bash
$ who
testtest01 tty2         2008-06-04 06:41
testtest02 tty3         2008-06-04 06:42
[fsucs@acer1 Slides]$
```

Adding local users: Unix

⇶ The password file requires 7 ":" separated fields:

⇶ Name:Password   (encrypted):UID:GID:GECOS:Home Directory:Shell

⇶ Example:

`user1:f9cPz5ilB5N0o:501:501:USER1:/home/faculty/user1:/bin/tcsh`

Adding users: Unix

☞ Some UNIXes provide **vipw**, which will lock out others from editing the /etc/passwd file simultaneously and may also include some syntax checking, just like **visudoer**

Unix users: grouping them

☞ Make sure the group in /etc/passwd exists in /etc/group, which has the format:

☞ groupname:password:gid:user-list

⇛ groupname is the name of the group.

Unix users: grouping them

⇶ gid is the group's numerical ID within the system; it must be unique.

⇶ user-list is a comma-separated list of users allowed in the group (used for multiple-group memberships by an individual).

Unix users: grouping them

☞ Example

```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
tty:x:5:
disk:x:6:root
lp:x:7:daemon,lp
mem:x:8:
kmem:x:9:
wheel:x:10:root
```

Unix/Linux: making user accounts

☞ Give the user a password: **passwd username** (as root)

☞ Edit their disk quota (if disk quotas are in use) via **edquota**. Type **edquota -p protouser username**. (How do users see their current quota usage? **quota -v**)

☞ (NOTE: Not all UNIXes support disk quotas!)

Unix: making user accounts

☞ Make sure the home directory exists and has the right permissions and that the appropriate default startup files are installed in the home directory (`.login`, `.cshrc`, `.Xdefaults`, etc.):

☞ Then do something like these:

Unix: making user accounts

```
mkdir /home/faculty/user1
cp /etc/skel/.[A-Za-z]* /home/faculty/user1
chmod 700 /home/faculty/user1
chown -R user1:user1 /home/faculty/user1

   [ OR, IF YOU DON'T HAVE THE ':' SYNTAX ]

chown -R user1 /home/faculty/user1
chgrp -R user1 /home/faculty/user1
```

Unix: making user accounts

You can do these steps manually, use a vendor-supplied script/program, or write your own.

☞ OpenSolaris: **useradd**, **usermod**, **userdel**

☞ FreeBSD: **adduser**

☞ Linux: **adduser**

☞ Linux: **useradd**

Unix: making user accounts

Also, there are often GUIs, or you can use webmin.

Unix/Linux: shadow password files

Most Unix/Linux distributions now use a "shadow" password file in addition to the main password file – a shadow password file moves the encrypted password out of the publicly-readable /etc/passwd file and into a root-accessible-only file. Why is this a good idea? See "John the Ripper" or "Ophcrack" – any hacker can try to systematically guess passwords with such programs.

Unix/Linux: shadow password files

Also allows for creation of new fields to support password rules, password aging, etc. Examples:

Unix/Linux: shadow password files

☞ OpenSolaris: `/etc/shadow` (See "man shadow")

☞ FreeBSD: `/etc/master.passwd` (See "man passwd")

☞ Redhat/CentOS: `/etc/shadow` (See "man shadow")

Unix/Linux: removing users

Removing Unix/Linux users – you can just undo the steps above!

However, it can be problematic to find all files owned by the user, if you gave them access to directories outside of their home directory.

Unix/Linux: removing users

☞ Or, **find / -user USERNAME -print** – but that only works as long as the username is still in the password file. Otherwise, you need to use **find / -uid UID -print**

Unix: removing users

☞ Don't forget their unread mailbox, often something like (`/var/spool/mail/username`)

☞ Don't forget any other system files that might have their name (e.g., `/etc/aliases.`)

Unix/Linux: removing users

You usually will want to archive (or otherwise preserve) the user data.

Unix/Linux: **disabling user accounts**

The easiest is usually to disable their login shell:

```
user1:x:501:501:USER1:/home/user1:/sbin/nologin
user2:x:502:502:USER2:/home/user2:/bin/false
```

You can put text into `/etc/nologin.txt` to modify the message from the **nologin** program, but it isn't customizable per user.

Unix/Linux /etc/shells

/etc/shells keeps a list of trusted shells users can change to via "chsh" /etc/shells is also consulted by other programs to make sure that a shell is a "legitimate" one for that system; in the past, even **sendmail** used to consult this file.

Unix/Linux /etc/shells

In general, this file is becoming much less used than it was in the past. Here's a current Fedora /etc/shells, which is very minimalistic:

```
$ cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
```

/bin/passwd binary

1. Require a reasonable choice of password.

2. Have a central password server where all must connect.

treating `/etc/passwd` **as a critical file**

1. On a busy machine, you might create a cron script to make backups, something like:

```
cp /saved/passwd.1 /saved/passwd.2
cp /saved/passwd.0 /saved/passwd.1
cp /etc/passwd /saved/passwd.0
```

treating `/etc/passwd` as a critical file

2. A rare but serious problem is having the "root" file system fill up and the password file getting truncated to a zero-length file. What is the biggest problem now? How can you get around it?

3. Use **pwck** (and **grpck**) on BSD systems to make cursory check of these important files.

treating /etc/passwd as a critical file

```
[root@sophie root]# pwck
user adm: directory /var/adm does not exist
user gopher: directory /var/gopher does not exist
user ident: directory /home/ident does not exist
user pcap: directory /var/arpwatch does not exist
user vmail: directory /home/vmail does not exist
pwck: no changes
```

treating `/etc/passwd` as a critical file

4. You can occasionally run password crackers to see if your users are putting in obvious passwords (notice this is less of a problem if you require them to have good passwords).

Setting up specialized accounts

Sometimes it is desirable to create limited accounts that serve only a single purpose, such as we saw with the old "sync" user login.

Windows user management

Adding local Windows users is quite easy, but the actual bitkeeping of the user information is far more recondite.

Data: UNIX Filesystems

Turning to the data management side, we have seen many new filesystems in the last few years: zfs, nilfs, btrfs, ext4 for example, and we have seen neat ideas such as FUSE which give users the ability to handle items structured as a filesystem in user-space, such as `sshfs` provides.

UNIX Filesystems

Traditionally, we have created our device files in `/dev`. Device files provide a connection between a device and standard UNIX system calls referencing those devices.

For UNIX filesystems, this has been a steady and monotonic weakening of connection between physical disk drive partitions and the eventual mount point.

The UNIX Filesystem

Identified by a "major" and a "minor" device number, as well as type "b" (block) or "c" (character, or raw device) – these examples are from Linux:

The UNIX Filesystem

```
root# ls -l /dev/

 [ ... ]

brw-rw----     1 root      disk        3,    0 Sep  9  2004 hda
brw-rw----     1 root      disk        3,    1 Sep  9  2004 hda1
brw-rw----     1 root      disk        3,   10 Sep  9  2004 hda10
brw-rw----     1 root      disk        3,   11 Sep  9  2004 hda11
brw-rw----     1 root      disk        3,   12 Sep  9  2004 hda12
```

The UNIX Filesystem

```
brw-rw----    1 root     disk       3,  13 Sep  9  2004 hda13
brw-rw----    1 root     disk       3,  14 Sep  9  2004 hda14
brw-rw----    1 root     disk       3,  15 Sep  9  2004 hda15
brw-rw----    1 root     disk       3,  16 Sep  9  2004 hda16
brw-rw----    1 root     disk       3,  17 Sep  9  2004 hda17
brw-rw----    1 root     disk       3,  18 Sep  9  2004 hda18
brw-rw----    1 root     disk       3,  19 Sep  9  2004 hda19
brw-rw----    1 root     disk       3,   2 Sep  9  2004 hda2

 [ ... ]
```

Unix/Linux: fsck

One task that is important for administrators but rarely to users is that of `fsck`: file system consistency checking. As a system administrator, running `fsck` can be one of the more exciting tasks after an unclean shutdown – use the -y option for fastest runs.

Unix/Linux: devices and filesystems

Filesystems have to exist somewhere: they can be in memory, such as a "RAM" filesystem, they can be on a hard disk drive, or more exotic venues. But the ones that system administrators generally care the most about are those that are on hard disk drives. While there has been quite a bit of movement to add logical layers to the device management process, we still use partitions directly to host filesystems — just take a look with "df" on your systems.

Unix/Linux: Device Naming conventions

The naming conventions and major/minor device numbers are **very** machine-specific. See LAH for some specifics on disk and tape device names. For Linux machines, you can also do a **locate devices.txt** to see if you can find a local copy, or for the most recent version, go to `http://www.lanana.org/docs/device-list/`.

Unix/Linux: Device Naming conventions

On modern Linux machines, `MAKEDEV` is a binary or a shell script (Debian, for instance, uses a shell version very similar to the old BSD version), usually located in `/sbin` (in the old days, this program was often in `/dev`!)

Unix/Linux: Device Naming conventions

As a shell script, typically **/sbin/MAKEDEV** would call the program **mknod**, which was a wrapper around calls to the `mknod(2)`:

Unix: Device Naming conventions

```
int mknod(const char *pathname, mode_t mode,
        dev_t dev);
```

DESCRIPTION

The   system   call mknod creates a filesystem node
(file, device special file or named pipe) named
pathname, with attributes specified by mode
and dev.

   [...]

Unix: Device Naming conventions

```
The  file  type  must  be  one of S_IFREG, S_IFCHR,
S_IFBLK, S_IFIFO or S_IFSOCK to specify a normal
file (which will be created empty),  character
special  file,  block  special  file, FIFO (named
pipe), or Unix domain socket, respectively.
(Zero file type  is  equivalent  to  type S_IFREG.)

If the file type is S_IFCHR or S_IFBLK then dev
specifies the major and minor numbers of the newly
created device special file; otherwise it is
ignored.
```

Unix: Device Naming Conventions

Note that the naming conventions vary even between different versions of the operating system. Solaris, for example, provides backwards compatibility with the old names via soft links:

Unix: Device Naming Conventions

```
Solaris->ls -l /dev/sd0a /dev/rsd0a
lrwxrwxrwx 1 root root 13 May 4 1995 /dev/rsd0a -> rdsk/c0t3d0s0
lrwxrwxrwx 1 root root 12 May 4 1995 /dev/sd0a -> dsk/c0t3d0s0
Solaris->ls -l rdsk/c0t3d0s0 dsk/c0t3d0s0
lrwxrwxrwx 1 root root 86 May 4 1995 dsk/c0t3d0s0 ->
   ../../devices/iommu@0,10000000/sbus@0,10001000/espdma@4,8400000/
     esp@4,8800000/sd@3,0:a
lrwxrwxrwx 1 root root 90 May 4 1995 rdsk/c0t3d0s0 ->
   ../../devices/iommu@0,10000000/sbus@0,10001000/espdma@4,8400000/
     esp@4,8800000/sd@
```

Unix: Device Naming Conventions

```
Solaris->ls -l /dev/sd0a /dev/rsd0a
lrwxrwxrwx 1 root root 13 May 4 1995 /dev/rsd0a -> rdsk/c0t3d0s0
lrwxrwxrwx 1 root root 12 May 4 1995 /dev/sd0a -> dsk/c0t3d0s0
Solaris->ls -l rdsk/c0t3d0s0 dsk/c0t3d0s0
lrwxrwxrwx 1 root root 86 May 4 1995 dsk/c0t3d0s0 ->
   ../../devices/iommu@0,10000000/sbus@0,10001000/espdma@4,8400000/
   esp@4,8800000/sd@3,0:a
lrwxrwxrwx 1 root root 90 May 4 1995 rdsk/c0t3d0s0 ->
   ../../devices/iommu@0,10000000/sbus@0,10001000/espdma@4,8400000/
   esp@4,8800000/sd@3,0:a,raw
```

Unix: Device Naming Conventions

Luckily, the actual naming convention that counts is the one that is used by the various sysadmin tools (fsck, mount, etc.).

Inside the filesystem: UNIX Symbolic links

As mentioned earlier, UNIX symbolic links are a very useful system administration tool.

☞ **ln -s file_to_link_to name_of_link**

☞ Can span file systems

☞ Can become "stale" and have "broken links".

Inside the filesystem: UNIX Symbolic links

As previously mentioned, symbolic links are nothing but a regular file with a bit set to indicate that it is a symbolic link; the contents of the file are the link value itself:

```
[langley@sophie Slides]$ ln -s /etc/passwd
[langley@sophie Slides]$ ls -l passwd
lrwxrwxrwx    1 langley  langley         11 Jan 24 12:01 passwd -> /etc/passwd
```

Inside the filesystem: UNIX setuid and setgid bits

    setuid and setgid on executables − the effective UID and GID of the user executing the program temporarily becomes the UID and GID of the owner of the file, if the suid and guid bits are set ("chmod 4xxx", "chmod 2xxx", "chmod 6xxx", "chmod u+s", "chmod g+s", etc. − see "man chmod" for details).

Inside the filesystem: UNIX setuid and setgid bits

```
# ls -l /usr/lib/sendmail
-r-s--x--x 1 root sys 397768 Nov 24 1998 /usr/lib/sendmail
```

Inside the UNIX filesystem: the "sticky" bit

Old semantics: On a plain file, the sticky bit indicates that the binary should remain in memory after the last user finishes executing the text segment − the program "sticks" in memory. Typically only settable by root and used to keep commonly-used programs in memory for quicker response.

Inside the UNIX filesystem: the "sticky" bit

This older use of the sticky bit has pretty much fallen out of use with quicker machines and kernels with better memory models.

UNIX Sticky bit

On a directory, the sticky bit still does mean something useful (from "man -s 2 chmod"):

UNIX Sticky bit

If a directory is writable and has S_ISVTX (the sticky bit) set, files within that directory can be removed or renamed only if one or more of the following is true (see `unlink(2)` and `rename(2)`):

☞ the user owns the file

☞ the user owns the directory

UNIX Sticky bit

☞ the file is writable by the user

☞ the user is a privileged user

UNIX Sticky bit

Example: shared writeable directories - `/tmp` and `/var/spool/mail`

```
drwxrwsrwt  3 bin staff 512 Jan 27 11:40 /tmp
```

Inside the filesystem: UNIX permissions extended

Most Unix kernels and file systems such as those for Linux and OpenSolaris extend the 9-bit "rwxrwxrwx" permissions to generalized access control lists (ACLs). You can control file access with more flexibility, using commands like "aclget", "aclput", "setfacl", or "getfacl".

UNIX permissions extended

UNIX directory permissions

☞ 'r' bit allows one to read directory

☞ 'x' allows one to enter directory

Inside the filesystem: UNIX inodes

UNIX file information data structure is contained in "inodes".

☞ Unique inode number per file per file system.

☞ The inode for a file holds most information about a file: size, pointer to 1st disk block, file permission bits, timestamps (file accessed ("ls -lu") , file modified ("ls -l"), inode modified "ls -lc"), etc.

UNIX inodes

☞ The directory entry only holds a name-inode pair

☞ The "ls" command is a window into the inode (try "ls -li")

Under the filesystem: LVM

Filesystems can also be built over logical devices which might reside on hardware that doesn't directly resemble the logical construction.

One approach is that of LVM, logical volume management. The idea here is to press physical volumes into a system of logical drives. We will talk later in the semester in some detail about LVM.