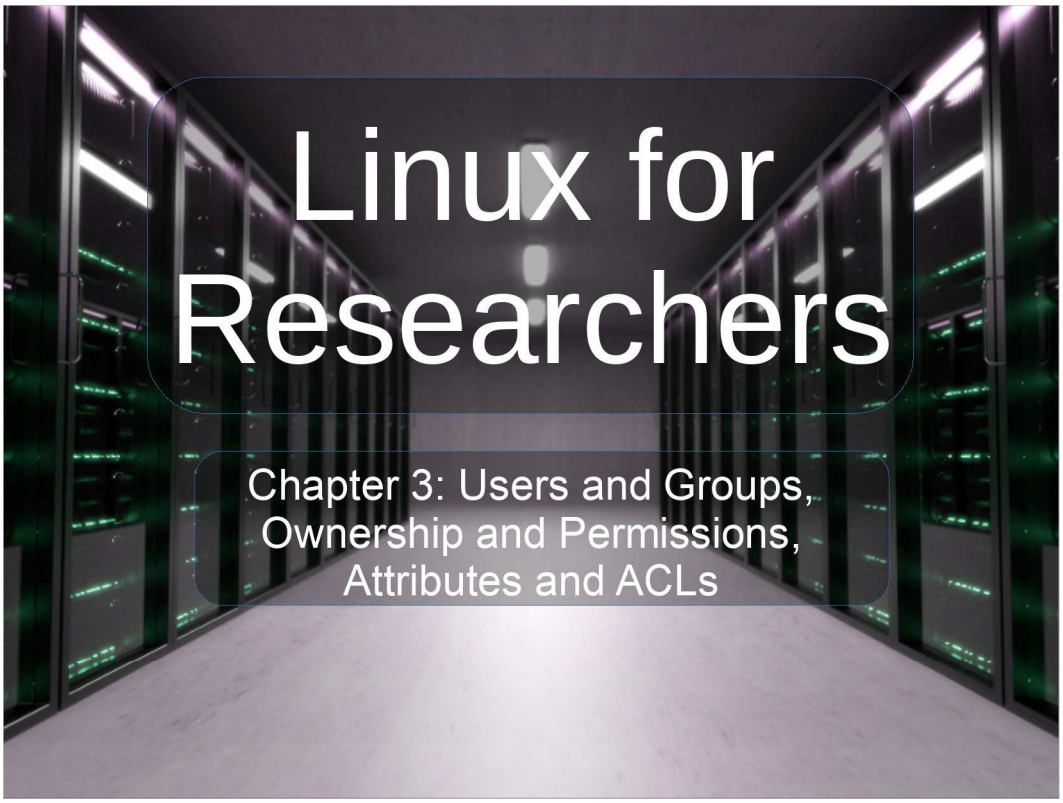


Until now, most of the things we've talked about have been for regular users. Now we'll start talking about real administrative tasks that only privileged users can perform.

Part 1: Users

Typical Computer User:

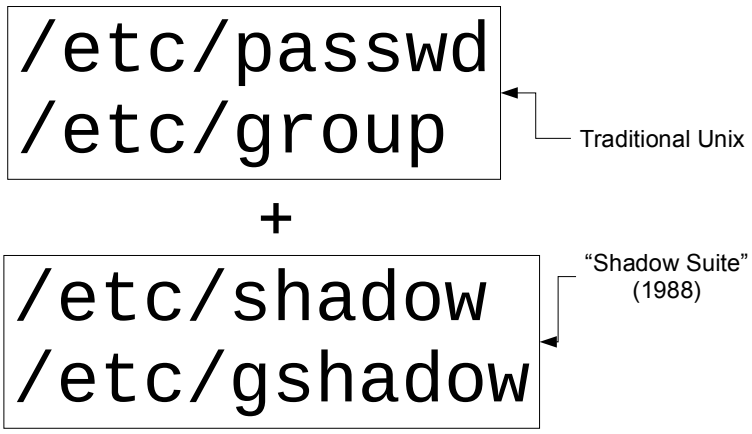
The first thing every user does when he or she sits down at a computer (even if he uses Windows) is log in. He:

1. Tells the computer who he is, and
2. Proves that he's who he says he is, by supplying a password (or equivalent).

Today we're going to talk about how this process works under Linux.

File-based Authentication Mechanism:

Traditionally, in Unix-like systems, authentication information was stored in two files, /etc/passwd and /etc/group. In the late 80s and early 90s, two more files were added, called /etc/shadow and /etc/gshadow, in an effort to increase security. These four files are still the most commonly used mechanism for storing authentication information on Unix-like computers.



There are other ways of storing authentication information: NIS, LDAP, Kerberos and others. These are usually used when several computers need to share a common database of users. For standalone computers, file-based authentication is still the most common mechanism, though.

One exception is Mac OS X (another Unix-like operating system). Before version 10.5 ("Leopard") OS X used a different type of user database, called NetInfo. After that, OS X switched to storing user information in "plist" files (XML-formatted files) in the directory /var/db/dslocal/nodes/default/users. Each of these files contains the information you'd find in the Unix-style password files, along with other OS X-specific information.

The /etc/passwd File:

```
~/demo> less /etc/passwd
root:x:0:0:root:/root:/bin/tcsh
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
.
.
.
bkw1a:x:500:501:Bryan K. Wright:/home/bkw1a:/bin/tcsh
elvis:x:502:503:Elvis Aaron Presley:/home/elvis:/bin/tcsh
```

The file has one account per line, with each account's information given as a colon-separated list of fields:

```
username:password:UID:GID:GECOS:homedir:shell
```

The files we're going to talk about are all plain text files, that can be edited with any text editor.

Passwd Fields:

```
elvis:x:502:503:Elvis Aaron Presley:/home/elvis:/bin/tcsh
```
username password uid gid GECOS homedir shell

The name under which the user logs in.

Either (1) a password hash, (2) a “*” to disable logins, or (3) an “X” to indicate that the password is stored elsewhere.

A unique numerical identifier for this user.

The numerical identifier for the user's primary group.

Originally “General Electric Comprehensive Operating System”. Now used to optionally store the user's full name, office and phone numbers.

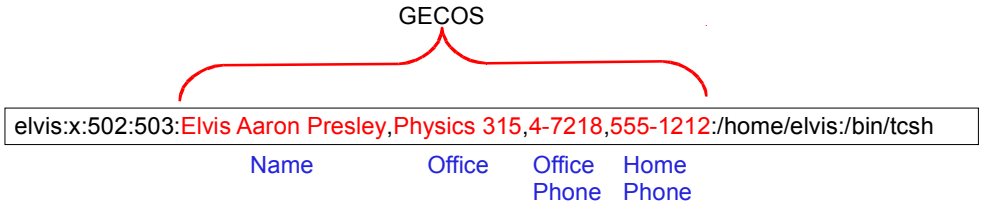
The user's home directory.

The user's login shell.

The GECOS Field:

From the passwd documentation:
"GECOS means General Electric Comprehensive Operating System, which has been renamed to GCOS when GE's large systems division was sold to Honeywell. Dennis Ritchie has reported: 'Sometimes we sent printer output or batch jobs to the GCOS machine. The gcos field in the password file was a place to stash the information for the $IDENTcard. Not elegant.'"

These days, the GECOS field is used to store an optional comma-separated list of information about the user. Often, the only item present is the user's real name, but the field may even be completely blank.



The Root User:

In Linux systems only one UID has any special privileges, and this is UID=0. UID=0 is reserved for the "root" user, or root equivalents. Usually, there is an account with the username "root" that has this UID. Any account with UID=0 has complete control of the computer.

For example, the root user can:

- read or write all files*,
- change any user's password, and
- can take on the identity of any other user without the user's password.

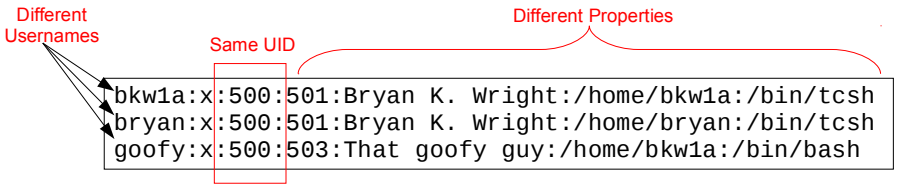
System vs. Real Accounts:
UID values under 500 are, by convention, reserved for "system accounts", like root. Accounts for real users usually begin with UID 500. But this is only a convention, and has no real significance.

* But he/she may first need to make the file writeable or readable.

Note that the "UID 500" convention is changing. Most Linux distributions are changing to a range of zero to 1000 for system accounts. But again, this is only a way of keeping things organized. The only uid number that has any special privileges is uid=zero.

The Difference Between Username and UID:

Internally, accounts are identified by their unique numerical UID, but any UID may have more than one username associated with it. Each username may have different properties (password, home directory, shell, etc.). You might think of these as the same user, but with different default settings.



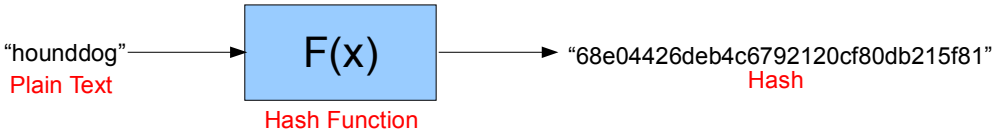
The ownership of a file is stored as a UID, not a username. When "ls" displays a file's owner, it looks up the UID in the list of accounts, and shows the first username that matches that UID. You can cause ls to show you the numerical user and group ownership of a file instead by using the "-n" switch:

```
~/demo> ls -l readme.txt
-rw-r----- 1 bkw1a demo 72 Jan 18 10:52 readme.txt

~/demo> ls -ln readme.txt
-rw-r----- 1 500 505 72 Jan 18 10:52 readme.txt
```

In the example above, if we wanted to have "ls" display the file's owner as "goofy" instead of "bkw1a", all we'd need to do is move the line for goofy above the line for bkw1a in /etc/passwd. We could do this with any text editor.

Cryptographic Hashes:

(also called “message digests”)



A cryptographic hash function, F(x), has the following properties:

• It's deterministic. For any particular input, F(x) will always produce the same output (called a “hash” or a “message digest”).

• The output of F(x) is always the same length.

• The output is almost always unique. Different inputs are very unlikely to produce the same output, even if they differ only slightly.

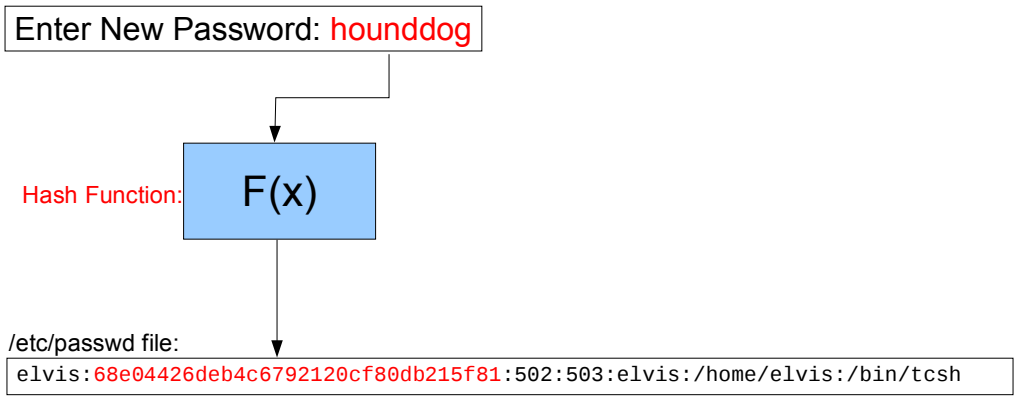
• F(x) is relatively easy to compute.

• The inverse function is extremely difficult to compute. It's very hard (ideally, impossible) to determine what plain text produced a given hash. This is called “trapdoor encryption”.

So why is this relevant to what we're talking about?
We'll see in a minute.

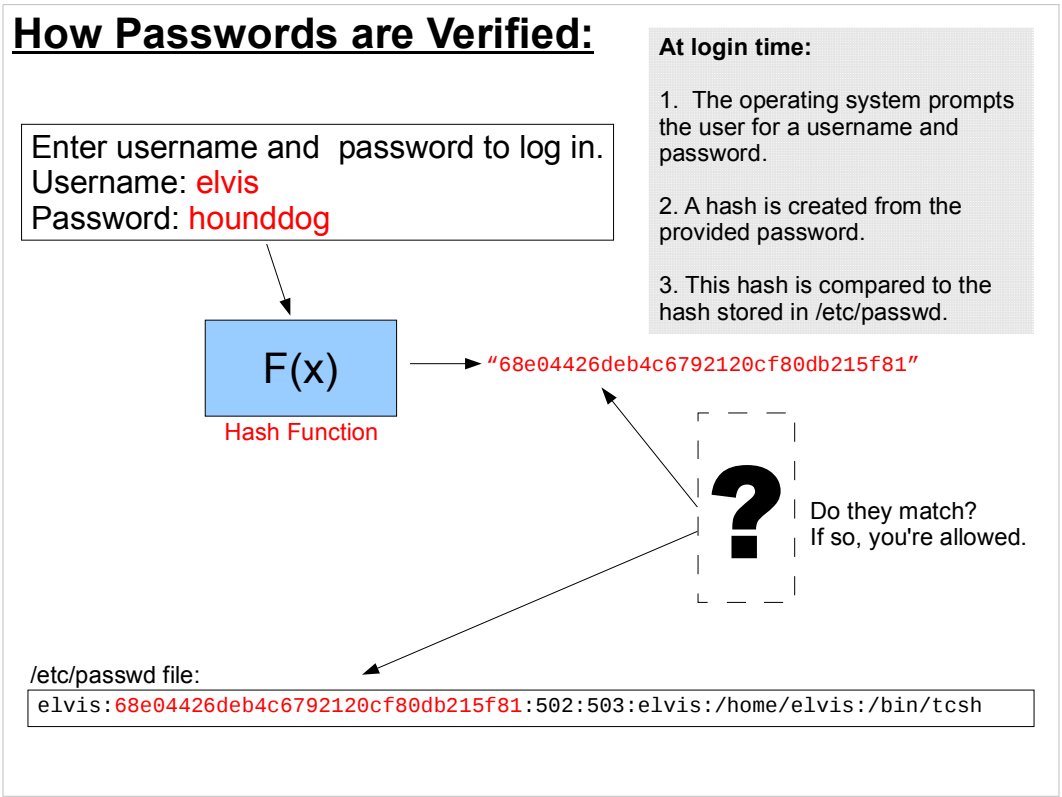
How Passwords are Stored:

When a new account is created, or when a password is changed, here's what happens:



The plain text password is never stored. Only the cryptographic hash is stored. Because hash functions are almost impossible to invert, nobody but “elvis” can know what his password is. So how can the system verify that a user knows his or her own password?

I'm simplifying things a little here. We'll explore some of the complications soon.



This means that the operating system doesn't need to know the password in order the verify that the user knows the password. Clever!

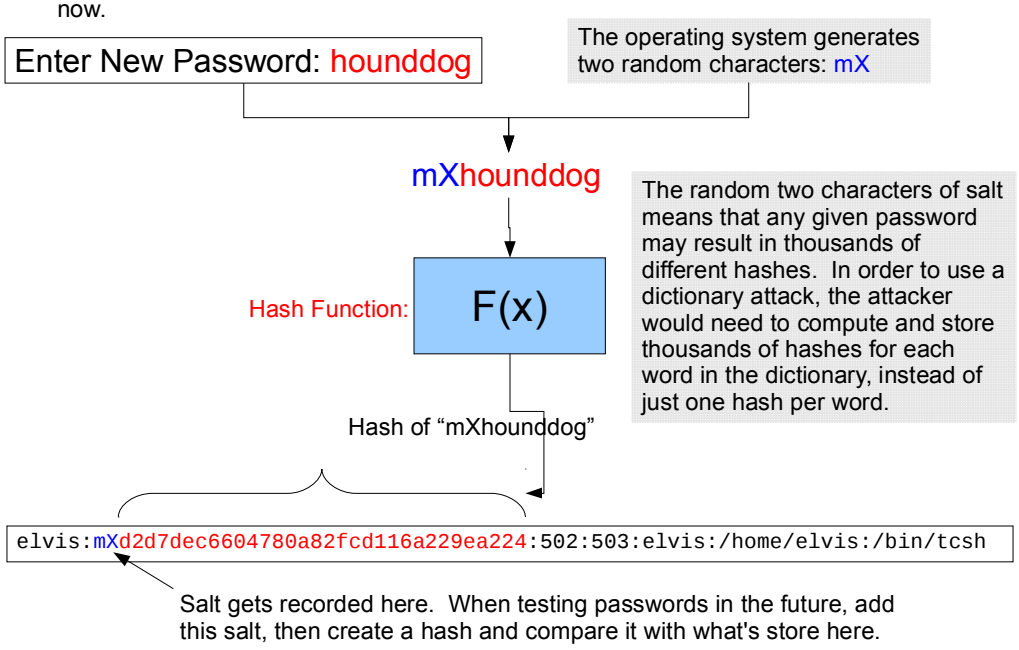
Dictionary Attacks:

Word	Hash
aardvark	88571e5d5e13a4a60f82cea7802f6255
abnegation	f8ec9b74a9b9aa1131abbfc0b8dca989
acrimony	a246c59fdccea42bc48202156a5f72de
adumbration	e6b65039a16dc9ffd4cca73e7f1b973f
advil	a3d4b48aebd5c6b9aaf57583601f1857
aerie	70f1f8799ad6af309af5434cb065bd59
affinity	1474047fb00b2d8d95646f7436837ed0
agnatha	c6bf04438cd39591695454ea4c755acb
aherne	e177eedf9e5b91c39d0ec9940c9870b9
aida	2991a6ba1f1420168809c49ed39dba8b
ajax	2705a83a5a0659cce34583972637eda5
AKKA	004ab7976e8b4799a9c56589838d97a6
algae	4360ef4885ef72b644fb783634a7f958
anthocyanin	7c425ea7f2e8113f6ca1e6e5c3a554a9
antidisestablishmentarianism	2a3ec66488847e798c29e6b500a1bcc6
anxiety	d3af37c0435a233662c1e99dbff0664d
apple	1f3870be274f6c49b3e31a0c6728957f
aurora	99c8ef576f385bc322564d5694df6fc2

Even though it's computationally expensive to compute the inverse of F(x), it's easy to generate hashes for thousands of words and store the results in a file. Once we've created this dictionary of hashes, we can just look up a given hash on the right, and match it with the plain text on the left. Since users are inclined to use common words as passwords, this is a security problem.

Adding Salt:

One way to reduce the effectiveness of dictionary attacks is to add some random "salt" to the plain text before creating the hash. This is always done now.



Password Hash Algorithms:

DES:
Based on National Bureau of Standards' Data Encryption Standard (DES).

"hounddog" = `mX1EcouR8fX7w`

MD5:
RSA's "Message Digest 5" algorithm.

"hounddog" = `$1$SG.2TL0i$vQuZcYnvA7kgMwg3gEB2RA`

SHA-256:
NSA "Secure Hash Algorithm (256-bit)".

"hounddog" = `$5$SG.2TL0i$NScvuF0OzU/lkdIModlq8Sn59mU`

and...

There's also an SHA-512 algorithm. These hash strings will begin with "6" when used in a password file.

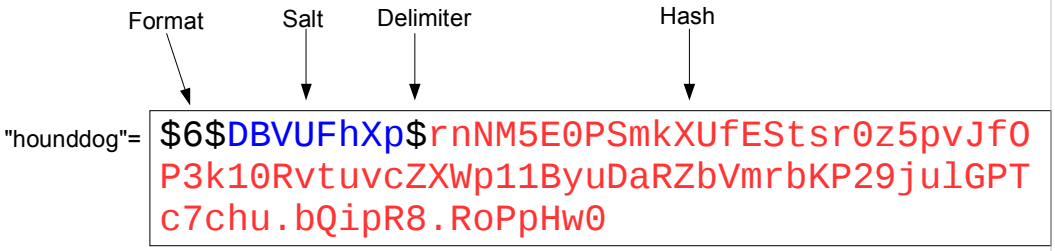
The SHA-256 and SHA-512 are used in current Linux distributions. The person behind this effort is Ulrich Drepper, the same person who wrote the long document about PC memory, that we referred to in an earlier talk. Here's his documentation about the SHA password hash standards:
http://people.redhat.com/drepper/SHA-crypt.txt

See the following Wikipedia article for more information about various password hash formats:
http://en.wikipedia.org/wiki/Crypt_(Unix)

Password Hash Algorithms (cont'd):

SHA-512:

NSA “Secure Hash Algorithm (512-bit)”.



This is the format you'll usually see these days in /etc/password or /etc/shadow.

The /etc/shadow File:

For many reasons, all users need to be able to read the /etc/passwd file. Commands run by users, such as “ls”, use information in /etc/passwd to map UIDs to usernames, for example. Most of the information in /etc/passwd isn't confidential, and for many years, it was assumed that, since the password field was stored as a hash, it was safe to let users see it.

Advances in computing power have made it more and more likely that hashed passwords can be “cracked”. Because of this, in the 1990s systems started moving away from storing password hashes in /etc/passwd.

The alternative location for password hashes was /etc/shadow, the “shadow password file”. This file is readable only by root. The completely new file also gave room to add more information about each account.

Passwords may still be stored in either file. If a password hash is present in /etc/passwd, it is used. If the password field in /etc/passwd contains an “x”, then /etc/shadow is consulted. All major distributions now store password hashes exclusively in /etc/shadow.

```
~/demo> ls -al /etc/passwd
-rw-r--r-- 1 root root 3565 Jan 25 00:12 /etc/passwd
```

Permissions

```
~/demo> ls -al /etc/shadow
-r-------- 1 root root 2024 Jul  1  2008 /etc/shadow
```

Permissions

One example, showing why it's not safe to allow just anyone to read password hashes:

When two different plain text strings produce the same hash, this is called a “hash collision”. Hash functions are designed to make hash collisions extremely unlikely, but they're still possible. Security researchers have recently found methods for producing hash collisions at will for the SHA-1 algorithm. Through these mechanisms, a plain text string can be produced which results in a given hash. The plain text may not be the same plain text that was used to originally produce the hash, but that doesn't matter for authentication purposes.

The Format of /etc/shadow:

```
elvis:$1$SG.2TL0i$5x2xlT7nWgLrRqKnJaYc59:14057:0:99999:7:::
```

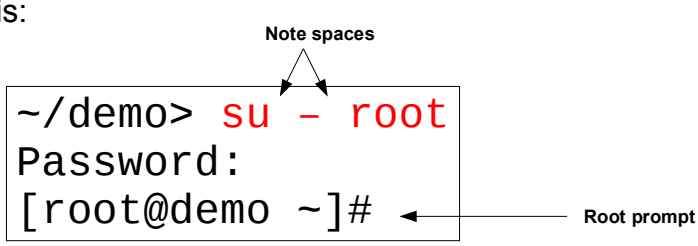
- username
- password hash
- days since Jan 1, 1970 that password was last changed
- days before password may be changed
- days after which password must be changed
- days before password is to expire that user is warned
- days after password expires that account is disabled
- days since Jan 1, 1970 that account is disabled
- a reserved field

Some Tools for Managing User Accounts:

su	“Set User”. Temporarily take on the identity of another user.
getent	Get information about users, groups, etc.
useradd	Add a new user account.
userdel	Delete a user account.
passwd	Change a user's password, enable/disable an account, change password expiration, etc.
usermod	Change a user's home directory, account expiration, etc.
chfn	Change the information in the user's GECOS field.
chsh	Change a user's login shell.

Using “su” to Become Root:

You can use the “su” command to temporarily change your identity. For example, if you wanted to temporarily become the root user, to perform some administrative task, you could enter a command like this:



Once you're done, type “exit” to return to your original unprivileged state.

Note that there are two ways to use “su”. If you use it as shown above, with a dash, the result will be just as though you'd logged out and logged back in as root. Root's login scripts (if any) will be executed, and you'll start in root's home diretory. Alternatively, if you type just “su root”, root's login scripts won't be executed, and you'll be left in the directory where you started. This may cause unexpected results.

Using “su” as root to Become Another User:

You can also use “su” as root to become another user:

```
[root@demo ~]# su – elvis
~/demo>
```

Note that root isn't required to enter a password to do this.

Getting User Information:

```
~/demo> getent passwd
root:x:0:0:root:/root:/bin/tcsh
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
.
.
.
bkw1a:x:500:501:Bryan K. Wright:/home/bkw1a:/bin/tcsh
elvis:x:502:503:Elvis Aaron Presley:/home/elvis:/bin/tcsh
```

```
~/demo> getent passwd elvis
elvis:x:502:503:Elvis Aaron Presley:/home/elvis:/bin/tcsh
```

Or just look at /etc/passwd. The advantage of getent is that it will work even if account information isn't stored in /etc/passwd. (Some alternative storage schemes are NIS, LDAP and Kerberos.)

The “finger” Command:

```
~/demo> finger elvis
Login: elvis                            Name: Elvis Aaron Presley
Directory: /home/elvis                  Shell: /bin/tcsh
Office: Physics 315, 4-7218             Home Phone: 555-1212
Last login Fri Dec 12 10:57 (EST) on pts/15 from
localhost.localdomain
Mail last read Fri Sep  5 08:14 2008 (EDT)
No Plan.
```

The “finger” command shows a user's GECOS information and information from other sources. If the user has a file called “.plan” in his or her home directory, the contents of this file will be displayed at the end of finger's output.

When he started Linux development in the early `90s, Linus Torvalds' .plan file read:

World Domination.

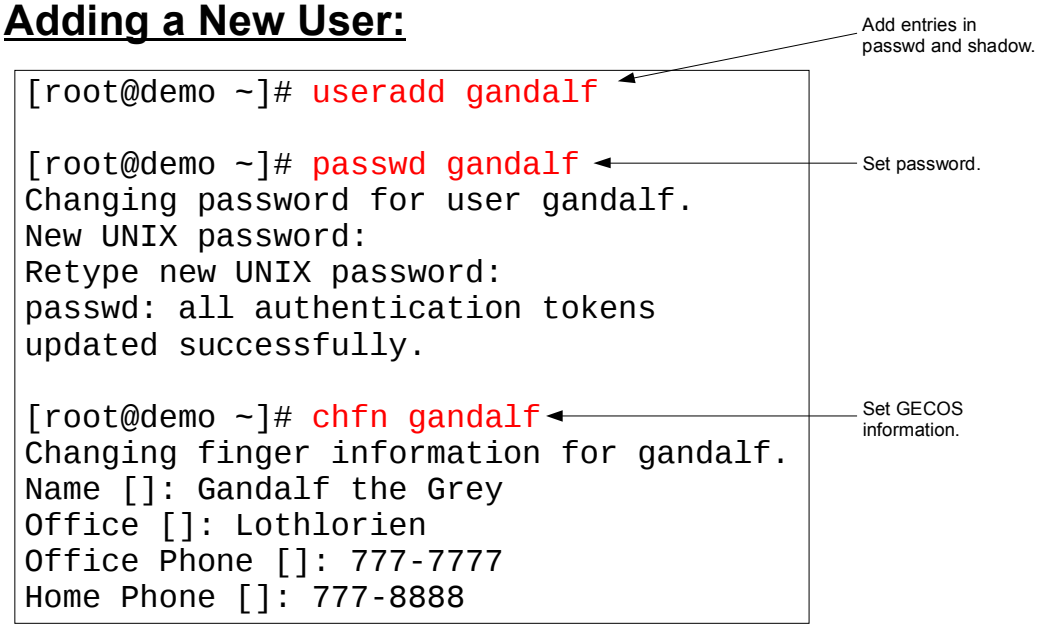
Finger can, in principle, be used to look at user information on other computer, across the network, but this isn't often done today, for security reasons. To enable this functionality, the remote computer must be running a server called a “finger daemon”.

The “whoami” and “id” Commands:

The “whoami” and “id” commands can be used to find out who the current user is:

```
~/demo> whoami
elvis

~/demo> id
uid=502(elvis) gid=503(elvis)
groups=503(elvis),505(demo)
```

Adding a New User:



At many sites, local administrators have created an “adduser” script, or something with a similar name, to automate this process, create accounts that conform to local standards, and perhaps perform additional tasks such as e-mailing an initial password to the new user.

Changing the Default Settings for New Accounts:

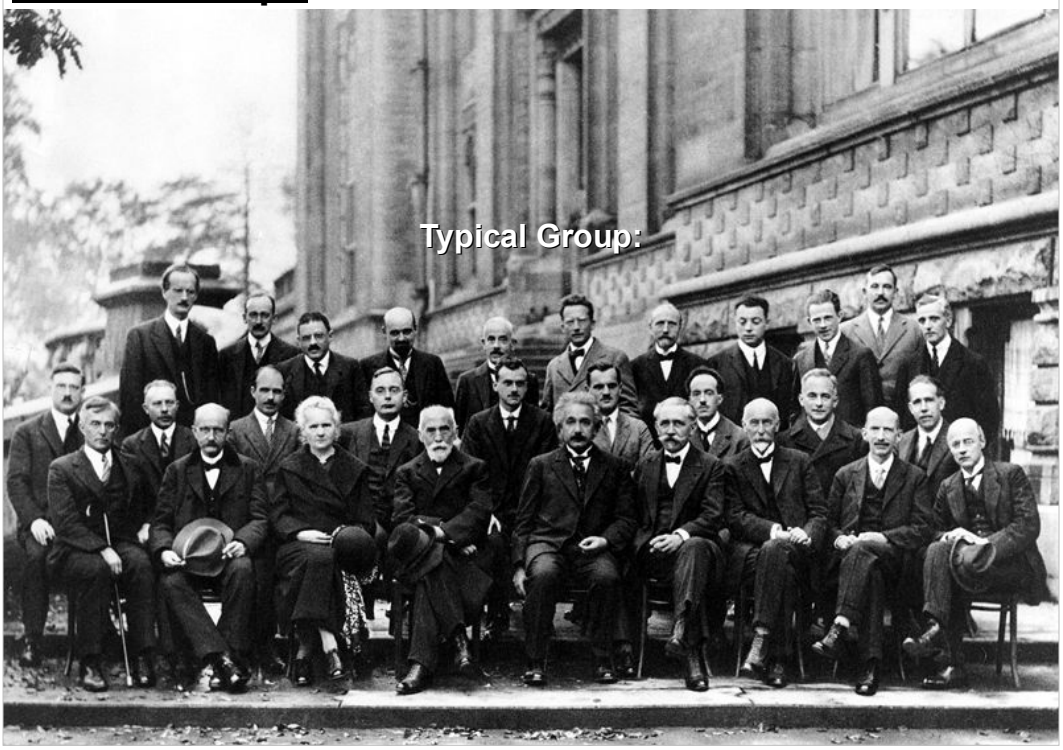
You can use the "-D" switch with useradd to change default settings used when creating new accounts:

Setting the default login shell to tcsh instead of bash:

```
[root@demo ~]# useradd -D -s /bin/tcsh
```

See "man useradd" for other options.

Part 2: Groups



It's often useful to define groups of users. The main reason this is done is so that these groups can be granted access permissions. We might say, for example, that all members of the “accountants” group have access to a particular file, but nobody else has access.

The /etc/group File:

```
~/demo> less /etc/group
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
.
.
.
bkw1a:x:501:
elvis:x:503:
```

The file has one group per line, with each group's information given as a colon-separated list of fields:

```
groupname:password:GID:userlist
```

Again, this is just a text file that can be edited with any text editor.

Fields in the /etc/group File:

```
demo:x:505:bkw1a,elvis
```
groupname password gid users

groupname The name of the group.

password	Either (1) the encrypted password, (2) a “*” to disable logins, or (3) an “X” to indicate that the password is stored elsewhere.
gid	A unique numerical identifier for the group.
userlist	A comma-separated list of users who belong to this group

The User's Primary Group:

Each user has a "primary group" specified in /etc/passwd. Note that the user doesn't need to be explicitly listed in the /etc/groups file as a member this group.

```
~/demo> getent passwd elvis
elvis:x:502:503:Elvis Aaron Presley:/home/elvis:/bin/tcsh

~/demo> getent group 503
elvis:x:503:
```

Most major Linux distributions, in their default configuration, are set up so that a new group will be created whenever a new user is added. This group will have the same name as the user, and will be the new user's primary group. This is part of what's called a "User Private Group" configuration. By making it safe to allow group read/write permission on files by default, UPG is intended to make it easier to create collaborative directories shared by several users. See the following web page for more details:
http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/ref-guide/s1-users-groups-private-groups.html

Traditionally, all users had a single group (called something like "users") as their primary group. Because of this, the default permissions on newly-created files couldn't allow the group to read or write the file, since that would allow all other users.

Some Commands for Working with Groups:

getent	Shows information from the /etc/group file.
groups	Shows user's group memberships.
groupadd	Create a new group.
groupdel	Delete a group.
groupmod	Change a group's name.
gpasswd	Set a group password, or manage the information in /etc/gshadow.
newgrp	Temporarily join a group.

Viewing Group Information:

Information about the "demo" group:

```
~/demo> getent group demo
demo:x:505:bkw1a,elvis
```

Which groups do I belong to?

```
~/demo/> groups
bkw1a demo
```

Which groups do I belong to? (Another way to find out)

```
~/demo> id
uid=500(bkw1a) gid=501(bkw1a)
groups=501(bkw1a),505(demo)
```

Or, if your group information is stored in /etc/group, just look at this file.

Group Passwords and the “newgrp” Command:

Administrators can set a password for a group,which will be stored in /etc/group.

```
[root@demo ~]# gpasswd demo
Changing the password for group demo
New Password:
Re-enter new password:
```

This has no effect on group members, but it allows non-members to temporarily join the group by using the “newgrp” command and supplying the password.

```
~/demo> newgrp demo
Password:
```

I've never seen this actually used, but it's interesting.

Adding Groups and Members:

Groups can be added with the “groupadd” command, and members can be added through “usermod”:

```
[root@demo ~]# groupadd whitecouncil

[root@demo ~]# usermod -a -G whitecouncil gandalf

[root@demo ~]# groups gandalf
gandalf : gandalf whitecouncil
```

The /etc/gshadow file:

For the same reasons that /etc/shadow was created as a supplement to /etc/passwd, the file /etc/gshadow was created as a supplement to /etc/group. The gshadow file is only readable by privileged users, and is primarily intended to contain the password hashes that originally lived in /etc/group.

The new file also provided an opportunity to add new functionality to the group system. This extra functionality is primarily accessed through the "gpasswd" command.

```
~/demo> ls -al /etc/group
-rw-r--r-- 1 root root 3565 Jan 25 00:12 /etc/group
```

Permissions

```
~/demo> ls -al /etc/gshadow
-r-------- 1 root root 2024 Jul  1  2008 /etc/gshadow
```

Permissions

The Format of the gshadow File:

```
demo:$1$2tbB2/gG$UXUzIujq/2GMxhF3xrhtB0::
```

The fields are:

- group name
- encrypted password
- comma-separated list of group administrators
- comma-separated list of group members

Note that the gshadow file allows for the existence of "group administrators". These are users (possibly otherwise unprivileged, and possibly not even members of the group) who are allowed to change the group's password or add/remove members from the group, using the gpasswd command:

```
~/demo> gpasswd -a elvis demo
```

The “setgid” Bit:

It is possible to set the permissions and ownership on a directory so that files created within the directory will inherit the group ownership of the directory. This is accomplished by setting the “setgid” bit in the directory's permissions:

```
drwxrwsr-x   2 bkw1a demo   4096 Jan 27 13:12 shared
```

Setgid bit

Files subsequently created in the “shared” directory will have their group ownership set to “demo”, making it easier to share them with other members of this group.

Part 3: Managing File Ownerships and Permissions:

The “chown” and “chgrp” Commands:

A file's user ownership and group ownership can be changed with “chown” (change ownership) command:

```
[root@demo ~]# ls -l junk.dat
-rw-r--r-- 1 bkw1a bkw1a 0 Jan 25 00:07 junk.dat

[root@demo ~]# chown elvis junk.dat
[root@demo ~]# ls -l junk.dat
-rw-r--r-- 1 elvis bkw1a 0 Jan 25 00:07 junk.dat

[root@demo ~]# chown elvis.demo junk.dat
[root@demo ~]# ls -l junk.dat
-rw-r--r-- 1 elvis demo 0 Jan 25 00:07 junk.dat
```

Group ownership can also be changed with the “chgrp” command:

```
[root@demo ~]# chgrp demo junk.dat
```

Ownership of all files in an entire directory tree can be changed by using the “-R” (for “recursive”) flag on chmod and chgrp. For example:

chown -R elvis.demo phase1

where “phase1” is a directory.

The “stat” Command:

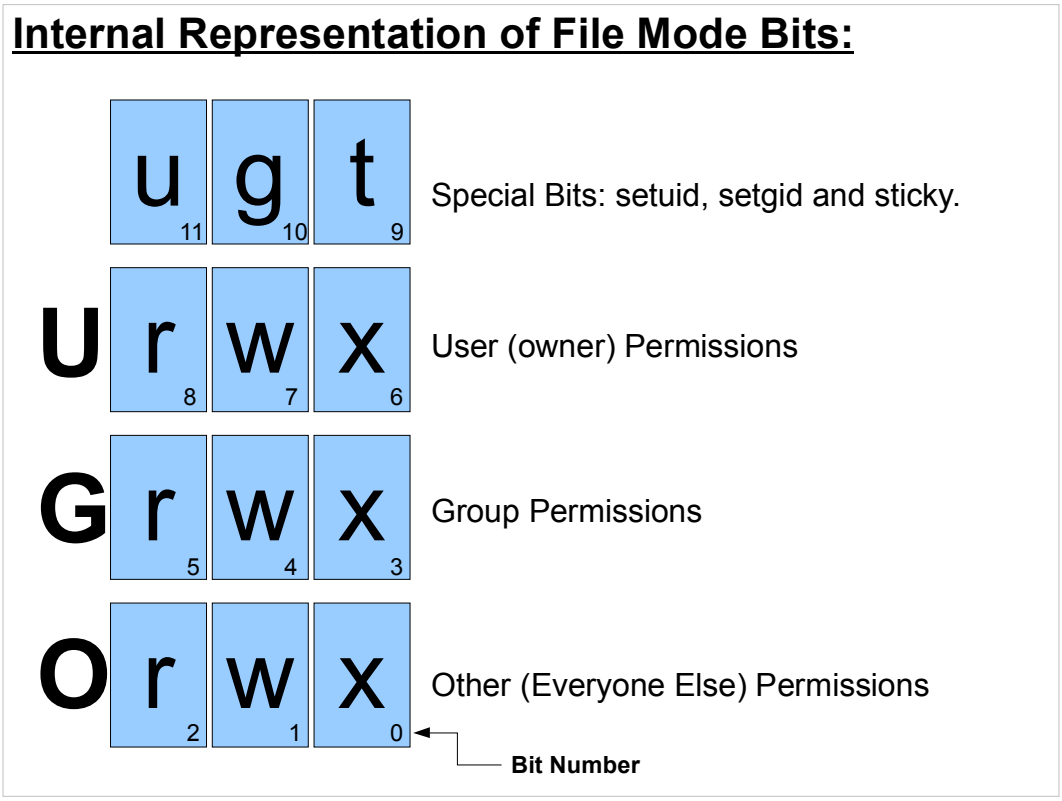
The set of permissions pertaining to a file is called the file's “mode”. The mode is displayed symbolically by commands like “ls”:

```
-rw-r-----  1 bkw1a demo     72 Jan 18 10:52 readme.txt
```

mode

Internally, though, the file's mode is represented by four sets of three bits (12 bits in all), which can collectively be written as a four-digit octal number. The “stat” command shows the mode in both formats:

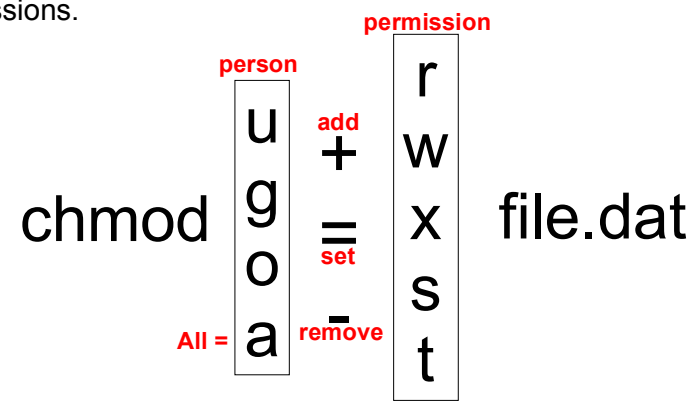
```
~/demo> stat readme.txt
  File: `readme.txt'
  Size: 72              Blocks: 8          IO Block: 4096
regular file
Device: fd00h/64768d    Inode: 17008595    Links: 1
Access: (0640/-rw-r-----)  Uid: (500/bkw1a)   Gid: (505/demo)
Access: 2009-01-19 10:58:02.000000000 -0500
Modify: 2009-01-18 10:52:29.000000000 -0500
Change: 2009-01-18 11:38:30.000000000 -0500
```



File permissions are actually stored as a set of 12 bits, shown above.

The “chmod” Command:

Permissions on files can be changed with the “chmod” (“change mode”) command. Permissions can either be specified symbolically or as an octal number. The symbolic form is most useful when modifying an existing set of permissions.



Give all users read permission:

```
~/demo> chmod a+r readme.dat
```

Give user and group read permission:

```
~/demo> chmod ug+r readme.dat
```

Alternatively, modes can be set directly as octal numbers:

Set the file's mode to rw-r-r-:

```
~/demo> chmod 0644 readme.dat
```

You can use “chmod g+s directory” to set the “setgid” bit on a directory. We'll see later that you can use “chmod u+s” to turn on a “setuid” bit.

Permissions on Directories:

- You need read permission on a directory in order to list its contents, even if all of the individual files within the directory are readable by you.

- If you have write permission on a directory, you can delete any file within the directory, regardless of whether you have ownership or write permission on the particular file.

- You need execute permission on a directory in order to traverse it. For example, to “cd” into a directory, you need execute permission.

The meaning of permissions on files if fairly clear, but the meaning of read, write and execute on a directory may need some explanation.

The “Sticky Bit”:

One of the bits in a file's mode is called the “sticky” bit. If this bit is set on a directory, only a file's owner (or root) is allowed to delete or rename files in this directory, no matter what would otherwise be allowed. The sticky bit is most often used on temporary directories, like /tmp, where everyone needs to have write access, but it's desirable to prevent users from deleting one another's files.

```
drwxrwxrwt  34 root root 36864 Jan 27 15:49 tmp
```

The sticky bit shows up in the symbolic representation of the permissions as a “t” in the last position if the “x” bit is set for “others”, and as a “T” in this position otherwise.

Attributes, and Immutable Files:

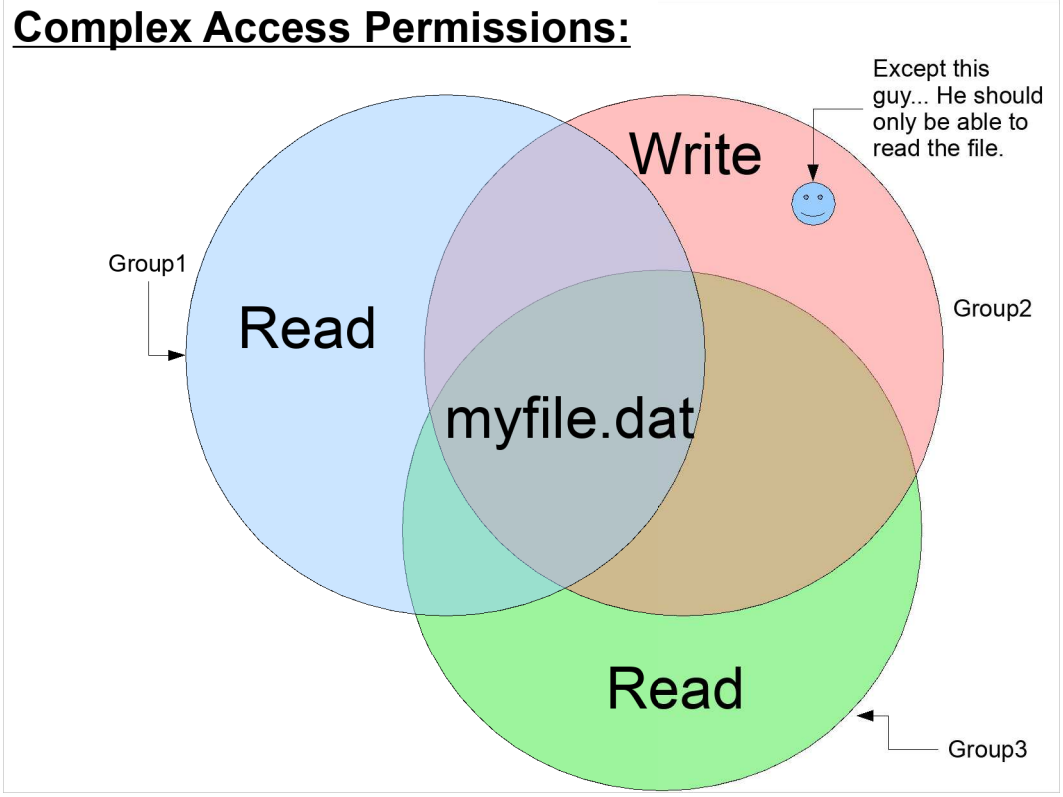
In addition to the file permissions available on all Unix filesystems, the common filesystems under Linux also support a set of extended file attributes. Some of these are quite esoteric, but one, at least, is widely useful. This is the "immutable" attribute.

Files marked as immutable cannot be changed or deleted, even by the root user (although the root user has the power to remove the immutable attribute). This is useful for preventing accidental or malicious modification of files that are normally unchanging.

Attributes can be listed with "lsattr" and changed with "chattr":

```
[root@demo ~]# lsattr junk.dat
------------- junk.dat

[root@demo ~]# chattr +i junk.dat

[root@demo ~]# lsattr junk.dat
----i-------- junk.dat
```

The attribute can be removed with the "-i" flag.



But what if we want to have a really complex system of access permissions for a file? We can't do this with just user, group and other.

Access Control Lists (ACLs):

In addition to the read/write/execute permissions for user/group/other, the most common Linux filesystems also offer a mechanism to deal with more complex access restrictions. This mechanism is called Access Control Lists (ACLs).

When ACLs are available, each file or directory can have a complex set of access permissions associated with it. These permissions consist of an arbitrarily long list of access control rules. A rule can be created, for example, to give a particular user read-only access to a file, or to allow read-write access to a particular group.

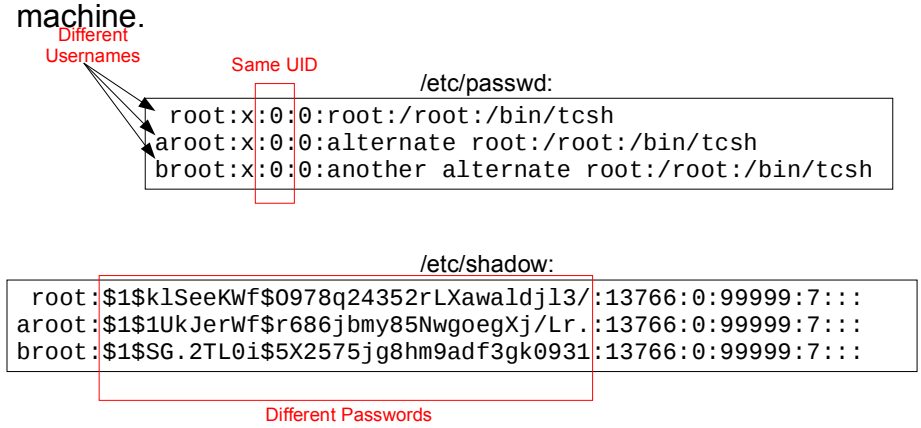
ACLs can be modified with the "setfacl" command, and viewed with the "getfacl" command.

```
[root@demo ~]# getfacl myfile.dat
# file: myfile.dat
# owner: elvis
# group: demo
user::rw-
group::r--
other::---

[root@demo ~]# setfacl -m user:priscilla:rw

[root@demo ~]# getfacl myfile.dat
# file: myfile.dat
# owner: elvis
# group: demo
user::rw-
user:priscilla:rw
group::r--
other::---
```

Part 4: Allowing Users to Act as Root

Multiple “Root” Accounts:

Since there can be more than one username associated with a given UID, it's possible to have multiple root accounts with different passwords. This is often how things are arranged when two or more people need to have administrative access to a machine.



The "setuid" Bit: (CAUTION!)

It is possible to set the permissions and ownership on an executable file so that the program will always automatically run as though it had been invoked by a specified user. This is most often used to allow unprivileged users to do something that only the root user could normally do. Changing a password, for example:

```
~/demo> ls -al /usr/bin/passwd
-rwsr-xr-x 1 root root 22984 Jan  6  2007
/usr/bin/passwd
```

Setuid bit

When the setuid bit is set, the file will run as though it had been invoked by the file's owner (root, in this case).

```
[root@ayesha ~]# chmod u+s /usr/bin/passwd
```

Setting the setuid bit.

When invoked by a non-root user, passwd runs as root and allows the user to change his or her own password, but no others. But if it's running as though it was invoked as root, how does passwd know that it was really invoked by a non-root user? The answer is that each process actually maintains two UID records, called an "effective" UID and a "real" UID. Because of this, passwd can look at the real UID to decide how it wants to behave.

It's important to note that it's up to setuid programs to decide on their own what they will and won't do. Mistakes often lead to security problems.

What happens if the setuid bit is set, but the file isn't executable? In that case, the output of "ls" would look like:
"-rwSr-xr-x" (with a capitol S), indicating that the file would execute as root if it were exectuable.

The /etc/sudoers File:

The "sudo" command and its associated configuration file, /etc/sudoers, allow an administrator to selectively dole out administrative privileges to certain users, under certain circumstances. For example, the sudoers file might be configured to grant root access to all members of the "wheel" group through the sudo command.

```
~/demo> groups
elvis demo wheel

~/demo> sudo useradd priscilla
Password:
```

To execute privileged commands, the user types "sudo" followed by the command. The user is then prompted for his or her own password, to verify the user's identity. The sudoers file can be configured to allow the user to execute all commands, or only selected ones.

Some distributions come configured this way by default. (Ubuntu Linux, for example.) This is also the way Mac OS X is configured.

Note that the name "wheel" has been used since the beginnings of Unix. It refers to people who are "big wheels".

Thanks!