After completing this chapter, you’ll be able to meet the following Solaris Administration Exam objectives:

- Identify the following login procedures: log into a system, log out of a system, and change login passwords.
- State the command used to identify which users are currently logged into the system.
- State the steps required to create user accounts on the local system using the `admin-tool` utility.
- State the command syntax to add, modify, or delete user/group accounts on the local system with the `useradd`, `groupadd`, `usermod`, `groupmod`, `userdel`, or `groupdel` commands.
- Given a user’s login shell, list the shell initialization files used to set up a user’s work environment at login.

To fulfill these objectives, this chapter discusses:

- Account Configuration Files
- Admintool
- User Administration Shell Commands
- Shells
- User Account Commands
• User account configuration files;
• Adding, modifying, and deleting user accounts with Admintool;
• Adding, modifying, and deleting user accounts with shell commands;
• User shells; and
• User account commands.

5.1 Account Configuration Files

Three principal files define the identity of a user account: the password file, the shadow file, and the group file.

The Password File

The password file contains the basic identifying information for each user allowed to access a system. The system location of the password file is `/etc/passwd`. The format of the password file is:

```
```

These fields are:

• `username`—the name that identifies the user account.
• `password`—in Solaris 8 an “x” in this field signifies that the corresponding shadow file contains the encrypted password string.
• `uid`—the unique numerical ID assigned to the account. The maximum value for UID is 2147483647, but administrators are urged to use values less than 60,000 to ensure compatibility with all of the tools that are used to manage accounts or display information that includes usernames.
• `gid`—the primary (default) numerical group ID assigned to the account. Like the UID field, the maximum value for GID is 2147483647, but a maximum of 60,000 is preferable.
• `gcos-field`—this is the user’s real name (the term “gcos-field” is the historical term for this field).
• `home-dir`—the directory where the user is placed after logging in; this usually contains the user's own files and directories.
Account Configuration Files

- login-shell—the initial shell that is started on behalf of the user upon logging in. If this field is blank, then /usr/bin/sh is used.

The password file can be read by anyone on the system. There is no information that must be kept secret in this file. A sample password file looks like this:

```
root:x:0:1:Super-User:/sbin/sh
daemon:x:1:1::/
bin:x:2:2:/usr/bin:
sys:x:3:3::/
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
muucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
listen:x:37:4:Network Admin:/usr/net/nls:
pete:x:100:4:Peter Gregory:/export/home/pete:/bin/sh
nobody:x:60001:60001:Nobody::/
```

Fields in the password file are delimited by colons (“:”), and blank fields are signified by two adjacent colons (“::”). For instance, note that the account name field for daemon is blank—just two colons. Also, daemon has no shell entry, so the last character for daemon is the colon delimiter.

The colon delimiter is also used in the shadow and group files, which are discussed in a later section.

It is possible and permissible to create more than one username in the password file with the same UID. Each username will have its own unique password. However, tools such as ls and ps, when used with options displaying username, will display the first username found in the password file matching the UID.

When a new account is added, changed, or removed with the user-add, usermod, or userdel commands (which are discussed later in this chapter), the system creates a backup copy of the password file, called /etc/opasswd.

The Shadow File

The shadow file contains each user account’s encrypted password, as well as specific per-account parameters governing “password aging.” The system location of the shadow file is /etc/shadow. The format of the shadow file is:

These fields are:
- **username**—this is the same username found in the password file.
- **password**—a 13-character encrypted password. If this field contains a lock string (e.g., “locked” or “NP”), the account is inaccessible; if blank, the account has no password.
- **lastchg**—date of last password change (literally the number of days between January 1, 1970, and the date the password was last changed).
- **min**—minimum number of days allowed before the password can be changed.
- **max**—maximum number of days allowed before the password expires.
- **warn**—the number of days prior to expiration that the user is warned.
- **inactive**—the number of days of inactivity allowed for the account before the account is automatically locked.
- **expire**—the date when the user account is deactivated.
- **flag**—a field reserved for future use.

The shadow file is restricted so that only the system administrator can read it. This is because an intruder could perform a “dictionary attack,” using guessable passwords in an attempt to determine the passwords for one or more accounts. Programs that “crack” account passwords are available for this purpose; thus, the shadow file is not publicly readable.

A sample shadow file looks like this:

```
root:Pe0iQfp2LcAig:10528::::::
daemon:NP:6445:::::::
bin:NP:6445:::::::
sys:NP:6445:::::::
adm:nIP3GPx2FIZYQ:11053::::::
lp:NP:6445:::::::
uucp:NP:6445:::::::
nuucp:NP:6445:::::::
listen:*LK*:::::::
pete:GSSUYVrJ8ERyA:11055::::::
nobody:NP:6445:::::::
```
Note that some of the accounts in the example shadow file have “NP” (“no password”) or “*LK*” (locked) in them. These are just two ways of signifying that the accounts are locked against login. There is nothing really magic about “NP” or “*LK*”—they are just one way of easily showing that these accounts are locked and going to stay that way. You could put other text in the password field to suit your needs; for instance, a helpdesk ticket number or a date.

When a user changes his or her password, the system creates a backup copy of the shadow file, called /etc/oshadow. This also occurs if root changes a user’s password.

**EXAM NOTES**

**THINK ABOUT IT . . .**

Why are encrypted passwords found in /etc/shadow and not in /etc/passwd?

For many commands (ps, ls, etc.) to work properly, /etc/passwd must be world readable. Prior to the use of /etc/shadow, all users’ encrypted password strings were also publicly readable. This gave people with “password cracking” programs an opportunity to discover other users’ passwords. By moving the encrypted password strings to /etc/shadow (which can be read only by root), the ability to access encrypted passwords is eliminated.

First, a bit of history. Older versions of UNIX did not have a shadow file; instead, the encrypted password string was found in the password file, in the second field that is usually filled with an “x” in Solaris. Solaris still supports the encrypted password appearing in the password file, although this is not advisable, since the password file is publicly readable.

Older versions of UNIX with no shadow file also had no password-aging capability.

The Network Information System (NIS) subsystem was designed around the old shadow-less architecture; this explains why NIS has no shadow map, nor does it support password aging.
The Group File

The group file contains a listing of all of the groups on the system, along with each group's numeric groupid and a list of each of the usernames that are secondary members of each group. The location of the group file is /etc/group. The format of the group file is:

groupname:password:gid:user-list

These fields are:

- **groupname**—this is the name of the group.
- **password**—this is an optional password for the group.
- **gid**—the unique numerical ID assigned to the group. The maximum value for GID is 2147483647, but administrators are urged to use values less than 60,000 to ensure compatibility with all of the tools that are used to manage accounts or display information that includes group names.
- **user-list**—a comma-separated list of users allowed in the group. These groups are users’ secondary group IDs.

A sample group file follows.

root::0:root
other::1:
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
uucp::5:root,uucp
mail::6:root
tty::7:root,tty,adm
lp::8:root,lp,adm
nuucp::9:root,nuucp
staff:GSSUYVrJ8EKyA:10:pete
daemon::12:root,daemon
sysadmin::14:
nobody::60001:
nomax::60002:
nogroup::65534:

Note the password field in the group “staff.” Groups can be password protected by putting a password string into the password field.
Account Configuration Files

Note, though, that this is a completely manual process; there are no tools provided to put the group password in for you.

Hint: You can take a password string from the shadow file and use the vi editor to splice it into the group file. The group password will be the same as the account password from the shadow entry where you took the password string.

When a group is password protected, anyone who is not a member of the group will be challenged for the group’s password when they have entered the newgrp command.

**Primary and Secondary Groupids**

Each user account has one primary groupid—this is the group defined in the password file. When a user logs on, this is his or her associated groupid.

Each user account also has zero or more secondary groupids. A user’s secondary groups are those group entries that include the username in their lists of members.

There are some useful limitations of groupids. For example, each user account can be in no more than 16 groups, and a line in the group file cannot exceed 512 characters (including the newline character).

What happens if you need to add so many members to a group that you exceed the 512-character entry limit? You simply create another duplicate group entry—same name and group number—and list the additional members there.

The primary and secondary groupids directly impact file system access permissions, which is explored fully in Chapter 10.

**EXAM NOTES**

**THINK ABOUT IT . . .**

What would be the effect if a username were added to a group file entry when that group was the user’s primary group?

The additional entry would have no effect, since the user is already configured for the primary group membership in the /etc/passwd file.
The Root Account

The root account has special privileges on a system: Root is permitted to read and write every file on a system, regardless of the file’s ownership and permission settings. System administrators log in with the root account so that they can perform administrative tasks.

The root account gets its power and privilege from the value of its user number: Root is user number zero, defined in the password file.

EXAM NOTES

THINK ABOUT IT . . .

What would be the effects of changing an ordinary user’s user number to zero? What advantage might there be of putting root’s entry at the end of the passwd file (instead of the beginning)?

Changing an ordinary user’s user number to zero gives the user root privileges.

One advantage of putting root at the end of the password file is that if an intruder is able to modify the password file to give another user root privilege (see the first question here), then output from commands such as `ls` and `ps` would show that other user as the owner of root processes and files.

5.2 Admintool

Admintool is the primary user account maintenance program. It is used by system administrators to create, modify, and remove user accounts.

The system administrator must log in as root to use Admintool. To start Admintool, type `admintool&` at a shell prompt. The Admintool program appears as shown in Figure 5-1.

This initial view shows the system’s existing user accounts. Select the Browse menu to manage groups. Figure 5-2 shows a sample Browse menu.

Add User Account

To add a user, select the Edit menu, then Add. Fill in the userid in the User Name field, the user’s name in the Comment field, and the home directory in the Path field. If the user is to belong to any other groups, add the
group numbers in the Secondary Groups field. If you wish to impose password aging parameters, specify them in the Min Change, Max Change, Max Inactive, Expiration Date, and Warning fields. An example Add User screen is shown in Figure 5-3. Click OK or Apply to add the user.

**Modify User Account**

To modify a user account, select a user account in the main window by clicking on it. Then select the Edit menu, then Modify. An example Modify User screen appears in Figure 5-4.
Lock User Account

Admintool can be used to lock a user account. This might be a useful alternative to removing an account (or changing its password) if you need to temporarily block access to the account. To lock a user account, modify it as you normally would, then in the Password pull-down, select *Account is Locked*. An example is shown in Figure 5-5.

Delete User Account

Admintool is also used to delete user accounts. To delete a user account, select a user account in the main window. Then select the *Edit* menu, then *Delete*. See Figure 5-6 for an example.
Add Group

Adding groups with Admintool is as straightforward as adding users. To add groups using Admintool, select the Browse menu, then Groups. The list of groups on the system then appears. See Figure 5-7 for an example.

To add a group, select the Edit menu, then Add. Type in the number and name of the new group, then press OK. An example is shown in Figure 5-8.
Chapter 5 | User Administration

Figure 5-5  Admintool Lock User

Figure 5-6  Admintool Delete User
Modify Group

Use the Edit, Modify group menu items to change the name or members of a group. Group members are listed by name, separated by commas. An example is shown in Figure 5-9.

Delete Group

Admintool is also used to delete groups. See Figure 5-10 for an example. To delete a group, select a group in the main window by clicking on it. Then select the Edit menu, then Delete.
Removing a group destroys the record of its existence. The listed group name for any files or directories that were owned by the group account will reflect the numeric group of the prior owner. It is recommended that, instead of removing a group, you instead remove all users from its membership list and add the letters “LK” (short for “Locked”) in the group’s name field.

---

**Figure 5–9**  Admintool Modify Group

**Figure 5–10**  Admintool Delete Group
In addition to the Admintool GUI program, there are also several shell commands that can be used to administer user and group accounts. The commands are `useradd`, `usermod`, `groupadd`, and `groupmod`.

These commands let you administer user and group accounts. Let's explore each of these in greater detail.

### useradd Command

The `useradd` command is used to add a user account to the system. The syntax of the `useradd` command is `useradd [options] userid` where `options` is one or more `useradd` options, and `userid` is the name of an account to create. With no options specified, `useradd` will create an account with `useradd` defaults.

A second form of the command is `useradd –D [options]`. This form lets you view and, optionally, modify defaults.

The options for `useradd` are:

- `-A authorization`. One or more comma-separated authorizations defined in the `/etc/security/auth_attrauth_attr` file. Only a user or role who has `grant` rights to the authorization can assign it to an account.
- `-b base_dir`. The default directory for the system if `-d dir` is not specified. `base_dir` is concatenated with the userid to define the user’s home directory. If the `-m` option is not used, `base_dir` must exist. For example, `base_dir` would typically be something like `/export/home`. If the account `jeffg` were being created, then the user’s home directory would be `/export/home/jeffg`.
- `-c comment`. This is generally a short description of the user account, and is currently used as the field for the user’s full name. This information is stored in the user’s `/etc/passwd` comment (name) field.
- `-d dir`. The home directory. The default is `base_dir/account_name`, where `base_dir` is the directory for new login home directories, and `account_name` is the new login name.
- `-D`. Display the default values for `group`, `base_dir`, `skel_dir`, `shell`, `inactive`, and `expire`. When used with the `-g`, `-b`, `-f`, `-e`, `-A`, `-P`, or `-R` options, the `-D` option sets the default values for the specified fields. When used with no other options, the `-D` option displays all current defaults. The defaults are:
Chapter 5 | User Administration

- `group=other (GID of 1)`
- `base_dir=/home`
- `skel_dir=/etc/skel`
- `shell=/bin/sh`
- `inactive=0`
- `expire=Null`
- `auths=Null`
- `profiles=Null`
- `roles=Null`

- `-e expire`. Set the expiration date for the userid. The user cannot log in after this date. The `expire` option argument is a date entered using one of the date formats included in the template file `/etc/datemsk`. Permissible formats include “10/5/2001” and “October 5, 2001” (dates including spaces must be quoted). A null value (“”) defeats the status of the expired date. The `-e` option is useful for creating temporary logins.

- `-f inactive`. The maximum number of days allowed between uses of a login ID before that ID is declared invalid. Normal values are positive integers. A value of 0 defeats the status.

- `-g group`. An existing group’s integer ID or character-string name. Without the `-D` option, it defines the new user’s primary group membership and defaults to the default group. You can reset this default value by invoking `useradd -D -g group`.

- `-G group`. An existing group’s numeric or character-string name. It defines the new user’s supplementary group membership. Duplicates between group with the `-g` and `-G` options are ignored. No more than NGROUPS_MAX (a value defined in `/usr/include/limits.h`, usually 16) groups can be specified. Multiple group names or numbers are separated by commas.

- `-k skel_dir`. A directory that contains skeleton information (such as `.profile`) that can be copied into a new user’s home directory. This directory must already exist. The system provides the `/etc/skel` directory that can be used for this purpose. Files such as `.profile` that are found in `skel_dir` will be copied into the user’s home directory when the account is created.

- `-m`. Create the new user’s home directory if it does not already exist. If the directory already exists, it must have read, write, and execute permissions by `group`, where `group` is the user’s primary group.
- `-o`. This option allows a UID to be duplicated (nonunique). This will create an additional account with the same numeric UID as one that already exists. Normally, an error message will be generated advising that the new account cannot be created because of the duplicate.

- `-P profile`. One or more comma-separated execution profiles defined in the `/etc/security/prof_attr` profiles configuration file.

- `-R role`. One or more comma-separated execution profiles defined in the `/etc/user_attr` profiles configuration file. Roles cannot be assigned to other roles.

- `-s shell`. Full pathname of the program used as the user’s shell. Its default is an empty field, causing the system to use `/bin/sh`. The value of shell must be a valid executable file. Setting the shell to `/bin/echo` will effectively prohibit the user from being able to log in to the account.

- `-u uid`. The numeric UID of the new user. This UID must be a non-negative decimal integer below MAXUID as defined in `<sys/param.h>`. The UID defaults to the next available (unique) number above the highest number currently assigned. For example, if UID’s 100, 105, and 200 are assigned, the next default UID number will be 201. (UIDs from 0–99 are reserved for possible use in future applications.)

Some examples of `useradd` are shown here.

```bash
# useradd -D
group=other,1  basedir=/home  skel=/etc/skel
shell=/bin/sh  inactive=0  expire=  auths=  profiles=  roles=
#
# useradd -D -b /d1/export/home
#
# useradd jeffg
#
# useradd -s /bin/csh -u 14002 -c "Paul Graham" paulg
#
```

In the first example, `useradd -D` is used to display defaults. In the second example, the default home directory base directory has been changed to `/d1/export/home`. In the third example, username `jeffg` has been created with all defaults. In the fourth example, the username `paulg`
has been created; this user has the nondefault shell /bin/sh, will be assigned UID 14002 (unless it exists already), and the string “Paul Graham" will appear in the account’s comment field in /etc/passwd.

**EXAM NOTES**

**THINK ABOUT IT . . .**

You import many files onto your system (perhaps from another system’s backup tape); these files are owned by a nonexistent user, and you decide that it is easier to create a new userid associated with the files than to change the ownership of the files. How would this be done?

This is solved by creating a userid with a specific UID. For example,

```
# useradd engrtools -u 10082 -c "Engineering Tools"
#
```

**EXAM NOTES**

**THINK ABOUT IT . . .**

Your site adds its user accounts with `useradd` as part of a larger automated process. Different classes of users require different environments and, consequently, different initialization files (i.e., `.profile`, `.login`, etc.). How could you handle this easily with `useradd`?

You could create different “skeleton” directories, similar to the default `/etc/skel`. Each of these skeleton directories would contain `.profile`, `.login`, `.cshrc` (and others as applicable) files, each tailored for the specific class of user. Then, the `-k` option in `useradd` would be chosen as appropriate for each user added.

**usermod Command**

The `usermod` command is used to modify the account settings for existing userids on the system. The syntax of the `usermod` command is `usermod [options] userid`, where `options` is one or more `usermod` options, and `userid` is the name of an account to modify.

The options for `usermod` are:
• `-A authorization`. One or more comma-separated authorizations as defined in the `/etc/security/auth_attr` file. Only a user or role who has grant rights to the authorization can assign it to an account. This replaces any existing authorization setting.

• `-c comment`. Specifies a comment string. `comment` can be any text string, generally a short description of the login, and is currently used as the field for the user’s full name. This information is stored in the user’s `/etc/passwd` entry.

• `-d dir`. The new home directory of the user. It defaults to `base_dir/login`, where `base_dir` is the base directory for new login home directories, and `login` is the account name.

• `-e expire`. Specifies the expiration date for a login. After this date, no user will be able to access this login. The `expire` option argument is a date entered using one of the date formats included in the template file `/etc/datemsk`. Permissible formats include “10/5/2001” and “October 5, 2001” (dates including spaces must be quoted). A null value (“”) defeats the status of the expired date. The `-e` option is useful for creating temporary logins.

• `-f inactive`. Specifies the maximum number of days allowed between uses of a login ID before that login ID is declared invalid. Normal values are positive integers. A value of 0 defeats the status.

• `-g group`. Assigns an existing group’s numeric or character name. It redefines the user’s `primary` group membership.

• `-G grouplist`. Specifies an existing group’s integer number or character string name. This redefines the user’s `supplementary` group membership. Duplicates between group with the `-g` and `-G` options are ignored. No more than `NGROUPS_U MAX` groups may be specified as defined in `/usr/include/param.h`. Multiple group names or numbers are separated by commas.

• `-l new_logname`. Specifies the new login name for the user. The `new_logname` argument must be a string no more than eight characters consisting of characters from the set of alphabetic characters, numeric characters, period (.), underline (_), and hyphen (-). The first character should be alphabetic and the field should contain at least one lowercase alphabetic character. A warning message will be written if these rules are not met. The `new_logname` argument must contain at least one character and must not contain a colon (:) or newline (\n).
-m. Moves the user's home directory to the new directory specified with the -d option. If the directory already exists, it must have permission to read/write/execute by group, where group is the user's primary group.

-0. This option allows a UID to be duplicated (nonunique). This will create an additional account with the same numeric UID as one that already exists. Normally, an error message will be generated advising that the account cannot be accessed because of the duplicate.

-P profile. One or more comma-separated execution profiles defined in /etc/security/auth_attr. This replaces any existing profile setting.

-R role. One or more comma-separated execution profiles defined in the /etc/user_attr profiles configuration file. Roles cannot be assigned to other roles.

-s shell. Full pathname of the program used as the user's shell. It defaults to an empty field causing the system to use /bin/sh as the default. The value of shell must be a valid executable file.

-u uid. Specifies a new UID for the user. It must be a non-negative decimal integer less than MAXUID as defined in /usr/include/param.h. The UID associated with the user's home directory is not modified with this option; a user may not have access to his or her home directory until the UID is manually reassigned using the chown command.

Some examples of usermod are shown here.

```bash
# usermod -s /bin/tcsh hamidf  #
# usermod -l shammer sgilbery  #
# usermod -G admins,backups jeffg  #
```

In the first example, the shell for userid `hamidf` is changed to `/bin/tcsh`. In the second example, the userid is changed from `sgilbery` to `shammer`. In the last example, the user `jeffg` is added to the secondary groups `admins` and `backups`. 
**EXAM NOTES**

THINK ABOUT IT . . .

What command would you use to change a user’s home directory? Would any follow-up tasks be necessary?

Use the `usermod -m` command to change a user’s home directory. You will also have to move the contents of the user’s home directory to the new location and check to make sure that the permissions of the new home directory are correct.

---

**EXAM NOTES**

THINK ABOUT IT . . .

A group of people shares the same UID and account (they have different passwords since their usernames are different), and they work on files in the same home directory. One person in the group is given an additional role of occasionally working on confidential materials that the others in the group are not allowed to access. What change could be made to allow this person to continue working on the pool of files, but at the same time prevent others in the group from viewing the confidential files?

One way to solve this is to change the one person’s UID to a new number, and change his or her groupid, but then add those IDs to the group shared by the others. The command is:

```
# usermod -u 102 -g conf -G pool pbrown
#
```

In this example, the user’s name is pbrown. Pbrown is given a new UID (102), a new group name (conf), and added as a secondary member to the group pool.

---

**userdel Command**

The `userdel` command is used to remove a userid from the system. The syntax for `userdel` is `userdel [-r] userid`, where `userid` is the name of an account to be removed. The `-r` option, if specified, will remove the user’s home directory from the system.


**groupadd Command**

The `groupadd` command is used to add groups to the system. The syntax for the `groupadd` command is `groupadd [options] groupid`, where `options` is one or more `groupadd` options, and `groupid` is the name of a group to add.

The options for `groupadd` are:

- `-g gid`. Assigns the group number for the new group. This groupid must be a non-negative decimal integer below `MAXUID` as defined in `/usr/include/sys/param.h`. The groupid defaults to the next available (unique) number above the highest number currently assigned. For example, if groups 100, 105, and 200 are assigned as groups, the next default group number will be 201. Groupids from 0–99 are reserved by Solaris for future applications.
  - `-o`. Permits duplicate group number.

Some examples of `groupadd` are:

```bash
# groupadd admins

# groupadd -g 300 backupadmins
```

In the first example, the group `admins` is created. In the second example, the group `backupadmins` is created as group number 300.

**Exam Notes**

**Think About It . . .**

After a disaster, you recover files from tape. Many files, however, are recovered with their old groupids. People cannot access these files because the groupid’s have changed. You decide that it is easier to create a new group corresponding to the groupid on the recovered files than it would be to change the groupid of all of the recovered files. How would this be done?

Create a new group with `groupadd`, specifying the desired groupid. Then, add the required users to the new group with `usermod -G`. For example:

```bash
# groupadd -g 108 dbadmins
# usermod -G dbadmins paul
```
# usermod –G dbadmins mbowman
# usermod –G dbadmins sghani
#

**groupmod Command**

The `groupmod` command is used to modify existing groups. The syntax for `groupmod` is `groupmod [options] groupid`, where `options` is one or more `groupmod` options, and `groupid` is the name of a group to modify.

The options for `groupmod` are:

- `-ggid`. Specify the new numeric groupid for the group. This groupid must be a non-negative decimal integer less than MAX-UID, as defined in `/usr/include/sys/param.h`.
- `-o`. Allow the GID to be duplicated (nonunique).
- `-nname`. Specify the new name for the group. The name argument is a string of no more than eight bytes consisting of characters from the set of lowercase alphabetic characters and numeric characters. A warning message will be written if these rules are not met. The `name` argument must contain at least one character and must not include a colon (`:`) or newline (`\n`).

Some `groupmod` examples are shown here.

# groupmod –nsysadmins admins
#
# groupmod –g201 admins
#

In the first example, the group `admins` is renamed to `sysadmins`. In the second example, the group number for the group `admins` is changed to 201.

**Exam Notes**

**Think About It . . .**

You have upgraded an application that now expects its primary groupid to be a different value. The impact of changing the groupids on all of the associated files would be too great; instead, you want to change the number of the existing group. How would you do this?
Use the *groupmod* –g command to change the group number of an existing group. For example:

```
# groupmod –g420 finance
```

**groupdel** Command

The *groupdel* command is used to remove a group from the system. The syntax for *groupdel* is *groupdel group*, where *group* is the existing group name to be deleted. An example is shown here.

```
# groupdel admins
```

In this example, the group admins is removed from the system.

**EXAM NOTES**

**THINK ABOUT IT . . .**

What are some of the advantages of using the *user* and *group* commands rather than *Admintool*?

You have more control with the *user* and *group* commands than with *Admintool*. Further, the *user* and *group* commands can be made part of shell scripts or custom programs, which can help to automate the process of adding, changing, and removing users. Also, repetitive changes can be put into a script (e.g., if the path to everyone’s home directory has changed, then it would be easy to write a shell script to modify everyone’s account very quickly; this would be very time consuming using *Admintool*).

**5.4 Shells**

A shell is a system-supplied, character-oriented program that accepts commands and displays the results of those commands.

The commonly used shells included with Solaris are the Bourne Shell (called “sh”), the C-Shell (called “csh”), and the Korn Shell (called “ksh”). Table 5.1 compares the features of each shell.
Bourne Shell

The Bourne Shell is the default shell in the Solaris environment. Like the other shells, the Bourne Shell has a scripting language capability; its scripting language is probably the most popular in the UNIX world. It has no command history or command-line completion capabilities.

Initialization Files

The global initialization file, /etc/profile, is used to set environment variables for all Bourne Shell users. When a Bourne Shell user logs in, the user's shell executes /etc/profile first. Then, the user's own initialization file, .profile, is executed. The .profile file is located in the user's home directory.

The order of execution of Bourne Shell initialization files is:

1. .dtprofile in home dir (if using CDE)
2. /etc/profile
3. .profile in home dir
4. .login in home dir
5. .kshrc in home dir
6. .logout

---

**Table 5.1 Shell Features Comparison**

<table>
<thead>
<tr>
<th>FEATURE</th>
<th>Bourne Shell</th>
<th>C-Shell</th>
<th>Korn Shell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Command-line completion</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Command history</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Environment variables</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Job control</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Systemwide login initialization file</td>
<td>/etc/profile</td>
<td>/etc/login</td>
<td>/etc/profile</td>
</tr>
<tr>
<td>User login initialization file</td>
<td>.profile</td>
<td>.login</td>
<td>.kshrc</td>
</tr>
<tr>
<td>Shell execution initialization file</td>
<td>.profile</td>
<td>.cshrc</td>
<td>.profile</td>
</tr>
<tr>
<td>Initialization file order at login</td>
<td>1. .dtprofile in home dir (if using CDE)</td>
<td>1. .dtprofile in home dir (if using CDE)</td>
<td>1. .dtprofile in home dir (if using CDE)</td>
</tr>
<tr>
<td></td>
<td>2. /etc/profile</td>
<td>2. /etc/.login</td>
<td>2. /etc/profile</td>
</tr>
<tr>
<td></td>
<td>3. .profile in home dir</td>
<td>3. .cshrc in home dir</td>
<td>3. .profile in home dir</td>
</tr>
<tr>
<td></td>
<td>4. .login in home dir</td>
<td>4. .kshrc in home dir</td>
<td>4. .logout</td>
</tr>
<tr>
<td>User logout file</td>
<td>(none)</td>
<td>.logout</td>
<td>(none)</td>
</tr>
</tbody>
</table>
Environment Variables

Environment variables in Bourne Shell are defined using the `VARIABLE=value; export VARIABLE` syntax; an example follows:

```
$ TERM=xterm; export TERM
$  
```

Aliases

The Bourne Shell user can predefine a command alias. An example alias follows.

```
rm () { /usr/bin/rm -i $* }
```

In this example, the `rm` command is aliased so that the `-i` option (manually verify the removal of each file) is used.

C-Shell

The C-Shell has a somewhat richer feature set than the Bourne Shell, specifically in that it has command history and command-line editing capabilities. Although C-Shell has its own scripting language, its syntax is quite different from the Bourne and Korn Shells, and is consequently less popular.

Initialization Files

The global C-Shell initialization file, `/etc/.login`, defines environment variables for all C-Shell users on a system.

There are two user initialization files, `.cshrc` and `.login`. When a user logs in, both `.cshrc` and `.login` are executed. If, after logging in, the user starts another C-Shell, then just `.cshrc` is executed.

The order of execution of C Shell initialization files is:

1. `.dtprofile` in home directory (if user logs into a console using CDE)
2. `/etc/profile`
3. `.profile` in home directory
When the C-Shell user logs out, the \texttt{.logout} file in the user’s home directory is executed.

The user files \texttt{.csbrc}, \texttt{.login}, and \texttt{.logout} are all located in the user’s home directory.

\textbf{Environment Variables}

C-Shell environment variables are set using the \texttt{setenv VARIABLE value} syntax; an example follows.

\begin{verbatim}
% setenv TERM xterm
%
\end{verbatim}

\textbf{Aliases}

C-Shell aliases are defined using the \texttt{alias} command. An example follows.

\begin{verbatim}
% alias rm "/usr/bin/rm -i \!\!*"
%
\end{verbatim}

In this example, the \texttt{–i} option is always added whenever using the \texttt{rm} command.

\textbf{Command History}

The interactive C-Shell has a command history capability. The \texttt{history} command can be used to later recall these commands, and even to re-execute them without having to type them in again.

The history capability is deactivated by default; to permanently enable command history, put the \texttt{set history} command in your \texttt{.login} file.

\textbf{Filename Completion}

To use filename completion, the C-Shell variable \texttt{filec} must be set. To permanently enable filename completion, put the command \texttt{set filec} in your \texttt{.login} file.

\textbf{Korn Shell}

The Korn Shell is a superset of the Bourne Shell; that is, the Korn Shell has all of the Bourne Shell’s features plus some of the built-in functions that the C-Shell is known for, including command-line completion and command history.
Initialization Files

The global initialization file, /etc/profile, is used to set environment variables for all Korn Shell users. When a Korn Shell user logs in, the user's shell executes /etc/profile. Next, the user's initialization file, .profile, is executed.

Many Korn Shell users also use a .kshrc file for Korn Shell–specific initialization commands. This capability exists so that Korn Shell users can use the Bourne Shell as needed. Login initialization for either Korn Shell or Bourne Shell can be placed in .profile, and Korn Shell–only commands placed in .kshrc.

To use .kshrc, the environment variable ENV must be set in the user's .profile (or in /etc/profile) as follows:

```
ENV=$HOME/.kshrc;export ENV
```

The user files .profile and .kshrc are located in the user's home directory.

The order of execution of Korn Shell initialization files is:

1. .dtprofile in home directory (if user logs into a console using CDE)
2. /etc/profile
3. .profile in home directory
4. .kshrc in home directory

Environment Variables

Korn Shell environment variables are set using the same syntax as that of the Bourne Shell: VARIABLE=value; export VARIABLE; an example follows.

```
$ TERM=xterm; export TERM
```

Aliases

Aliases are created using the alias name=command syntax. Our earlier example rm alias seen in the Bourne and C-Shells would be defined as follows:

```
alias rm="/usr/bin/rm -i "$"
```

The Korn Shell also supports Bourne Shell's function syntax.
Command-Line Editing

The Korn Shell uses a vi-like editing syntax that allows you to edit your command line while entering it. When you start entering a command, you are automatically placed in the vi “input mode.” To enter command mode while on the command line, press ESC. You can then move about the command line using vi-like movement commands (left/right arrow, or the H or L keys), and enter input mode again by pressing ESC. If in command mode you press “v,” the Korn Shell will start an actual vi session and give you full vi editing capabilities over your command line.

Command History

The Korn Shell keeps its command history in the file .sh_history, or in a different file as defined by the environment variable $HISTFILE. Up to 128 commands are stored there, unless the environment variable $HISTSIZE is set to a different value. The file .sh_history is assumed to be in the user’s home directory.

Restricted Shells

A restricted shell is a shell with very limited capabilities that generally includes these features:

• the cd (change directory) command is disallowed;
• the $PATH environment variable cannot be changed;
• commands containing pathnames beginning with “/” are disallowed; and
• output cannot be redirected to a file (the use of > and >> in a command line is disallowed).

Restricted shells are available for circumstances that require a user to have a shell, but where the user’s actions need to be more tightly controlled.

Two restricted shells are available: the restricted Bourne Shell and the restricted Korn Shell. The pathnames for these are /usr/bin/rksh and /usr/lib/rsh, respectively.

Warning

The restricted shell (/usr/lib/rsh) and the remote shell (/usr/bin/rsh) are easily confused. Because both /usr/lib/rsh and /usr/bin/rsh are shells, substituting one for the other may give you functional results, although not the results you intended.
Let's explore some common account-oriented commands and functions that users can perform.

### Finding Files by Username or Group

This section describes a few ways to find files on a system belonging to a specific user or group.

The `-l` option of the `ls` command shows the username and group name associated with files and directories. When `ls` shows the numeric value for a username or group, this is an indication that the username or group has been removed from the system. The `-n` option of the `ls` command shows the numeric userid and groupid.

The `find` command can be used to locate files on a system belonging to a particular user or group. The syntax of the `find` command is `find directory expression`, where `directory` is the place where `find` should start searching, and `expression` is one or more of the following:

- `-user username`. Search for files owned by `username`; `username` can be the actual username or the numeric userid.
- `-group groupname`. Search for files owned by `groupname`; `groupname` can be the actual group name or the numeric groupid.
- `-name filename`. Search for files named `filename`.
- `-print`. Find will list all files found matching criteria.
- `-exec command {} \;`. Execute `command` with the located file or files as arguments to `command`.
- `-type x`. Find files of type `x`, where `x` is one of: f (ordinary file), d (directory), b (block special file), c (character special file), D (door), l (symbolic link), p (named pipe), or s (socket).

Some examples of the `find` command follow:

```
# find /export/home -user mark -print
# find /tmp -user 1002 -exec ls -la {} \;
# find /export/tools -group toolmgr -name "*tmp" -exec rm {} \;
```

In the first example, `find` will list all files owned by username `mark` in or below the directory `/export/home`. In the second example, `find` will locate all “orphaned” files with userid 1002 in the directory `/tmp` and
perform an `ls -la` on each located file. In the last example, all files whose names end in “tmp” with groupid `toolmgr` in or below the directory `/export/tools` are removed.

**EXAM NOTES**

**THINK ABOUT IT . . .**

What command syntax would be used to change the ownership of “orphaned” files to that of another user? The `find` and `chown` commands could be chained together. For example:

```bash
find / -user 1002 -print | xargs chown terry
```

**What Users Are Logged In?**

**who Command**

The `who` command is used to display which users are currently logged into the system. The format of the `who` command is `who [options]`. The options for `who` are:

- `-a`. Specify all (-bdlpru) options.
- `-b`. Show the system’s boot time.
- `-d`. Show dead processes.
- `-H`. Print a header above output.
- `-l`. Show login processes waiting for someone to log in.
- `-n#`. Specify the number of users per line for the -q option.
- `-p`. Show active processes spawned by `init`.
- `-q`. Show just-logged-in usernames.
- `-r`. Show the system run level.
- `-s`. Short form of `who` (no time since last output or pid).
- `-u`. Show logged-in users.
- `-m`. Show information only about current terminal in -u format.
- `am i`. Show information about current terminal (same as -m).

Some examples follow.

```
# who
root           console   Sep 28 06:13    (:0)
```
pete pts/2 Nov 6 05:23 (java)
mark pts/6 Sep 28 06:13 (acorn)

# who –H
NAME LINE TIME
root console Sep 28 06:13 (:0)
pete pts/2 Nov 6 05:23 (java)
mark pts/6 Sep 28 06:13 (acorn)

# who –q
pete root mark
# users=3
# who am i
pete pts/2 Jan 31 05:24 (java)

In the first example, who shows all logged-in users. The second example includes a heading. The third example shows all logged-in users in quick format.

The last example shows information about the current user. Note that root is running the who am i command, but that the output shows the user pete. The who command displays which users are logged in, but does not reflect whether any users have su’d to other accounts (including root).

The command who am i is frequently confused with a different command, whoami. The answers given by these two commands are essentially the same unless you have su’d from one account to another. In this situation, the who am i command will display the original logged-in username, while whoami will display the effective (su’d) username.

rwho Command

In environments with the in.rwhod feature enabled, the rwho command shows which users are logged in to all systems on the local network. The syntax of the rwho command is rwho [-a], where the -a option will show only active users (those who have typed in commands in the past hour). An example follows.

# rwho
pete grommit:pts/0 Nov 7 05:48
mark wallace:pts/2 Nov 6 05:23

In this example, pete is logged into the system grommit, and mark is logged into the system wallace.
At large sites, **rwho** can create significant amounts of network traffic. This is because all systems running the **in.rwhod** daemon are frequently broadcasting their status to all other systems on the network. You can see why this feature is disabled by default.

### **rusers** Command

In environments with the **rusersd** feature enabled, the **rusers** command shows which users are logged in to another system on the local network. The syntax of the **rusers** command is **rusers [-ahilu] hostlist**, where **hostlist** is a list of one or more host names. The options are:

- **-a**. Give a report for a machine even if no users are logged on.
- **-h**. Sort alphabetically by host name.
- **-i**. Sort by idle time.
- **-l**. Give a longer listing in the style of **who**(1).
- **-u**. Sort by number of users.

When the **-l** option is used, the **rusers** output resembles that of the **who** command; the output format is:

```
userid    hostname:terminal    login date    login time   idle time
login host
```

Some examples follow.

```
# rusers -l
 Sending broadcast for rusersd protocol version 3...
pete         wallace:pts/2             Jan 31 05:24       7 (java)
gsmith       wallace:pts/3             Jan 31 05:29         (java)
mark         grommit:pts/0             Feb  1 04:04       7 (acorn)
 Sending broadcast for rusersd protocol version 2...
```

```
# rusers
 Sending broadcast for rusersd protocol version 3...
wallace       pete gsmith
 grommit       mark
 Sending broadcast for rusersd protocol version 2...
```

In the first example, the **rusers -l** command lists three users (**pete**, **gsmith**, and **mark**) on two systems (**wallace** and **grommit**), in the format
resembling that of `who`. In the second example, the abbreviated `rusers` output lists the usernames who are logged into each system on the network.

**finger Command**

The `finger` command can be used to get information about other users on the system. The syntax of the `finger` command takes three forms:

```
finger [-bfhilmpqsw] [username]...
finger [-l] [ username@hostname 1 [ @hostname 2 .. @hostname n]...]
finger [-l] [ @hostname 1 [ @hostname 2 .. @hostname n]...]  
```

In the first form, the available options are:

- `-b`. Suppress printing the user’s home directory and shell in a long format printout.
- `-f`. Suppress printing the header that is normally printed in a short format printout.
- `-h`. Suppress printing of the `.project` file in a long format printout.
- `-i`. Force “idle” output format, which is similar to short format except that only the login name, terminal, login time, and idle time are printed.
- `-l`. Force long output format.
- `-m`. Match arguments only on username (not first or last name).
- `-p`. Suppress printing of the `.plan` file in a long format printout.
- `-q`. Force quick output format, which is similar to short format except that only the login name, terminal, and login time are printed.
- `-s`. Force short output format.
- `-w`. Suppress printing the full name in a short format printout.

In the second and third forms, the `-l` option forces long output format. Some examples follow.

```
# finger gsmith  
Login name: gsmith                      In real life: Greg Smith  
Directory: /export/home/gsmith          Shell: /bin/sh  
On since Nov 7 06:18:59 on pts/17 from myclient  
No unread mail  
No Plan.  
#  

# finger -q gsmith  
```
User Account Commands

Login TTY When
gsmith pts/3 Wed Jan 31 05:29
#

# finger mark@grommit
[grommit]
Login Name TTY Idle When Where
pete Mark Foster pts/0 Thu 04:04 wallace
#

# finger @wallace
[wallace]
Login Name TTY Idle When Where
pete Peter Gregory pts/2 Wed 05:24 myclient
gsmith Greg Smith pts/3 1 Wed 05:29 myclient
#

# finger -l @wallace
[wallace]
Login name: pete In real life: Peter Gregory
Directory: /export/home/pete Shell: /bin/sh
On since Jan 31 05:24:14 on pts/2 from myclient
Mail last read Thu Jun 29 04:23:10 2000
No Plan.

Login name: gsmith In real life: Greg Smith
Directory: /export/home/gsmith Shell: /bin/sh
On since Jan 31 05:29:31 on pts/3 from myclient
2 minutes 11 seconds Idle Time
No unread mail
No Plan.
#

In the first example (finger gsmith), the username gsmith is logged in on the same system that executed the finger command. In the next example (finger –q gsmith), we see information about username gsmith but in quick format. In the third example (finger mark@grommit), we are querying specifically about the username mark on the system grommit. In the fourth example (finger @wallace), we are looking for short-form information about all users logged into the system wallace. In the last example (finger –l @wallace), we see long-form information about users logged into wallace.

In recent years many sites have disabled finger for security reasons. Check with your site administrator to see if this is the case in your organization. If finger is disabled on a system you’re trying to reach, you’ll get
an error message that reads, “[system-name] connect: Connection refused”.

**Change Password**

Root can change any user’s account with the `passwd` command. The format of the `passwd` command is `passwd username options`. Allowable options include:

- `-d`. Deletes the password. Subsequent logins will not prompt for a password.
- `-f`. Forces the user to change passwords at the next login.
- `-l`. Locks the account, preventing further logins.
- `-n min`. Sets the minimum number of days between password changes.
- `-w warn`. Sets the number of days before password expiration that the user is warned about it.
- `-x max`. Sets the maximum number of days between password changes.
- `-s username`. Displays password attributes for username.
- `-a`. Displays password attributes for all users. Use only with the `-s` option.
- `-h username`. Changes the home directory for username.
- `-e username`. Changes the login shell for username.
- `-g username`. Changes the gcos-field (full name) for username.

Some examples follow.

```bash
# passwd –f mark
#
# passwd –s gsmith
# gsmith PS
#
#
# passwd –h gsmith
Default values are printed inside of ‘[]’. To accept the default, type <return>. To have a blank entry, type the word ‘none’.

Home Directory [/export/home/gsmith]: /d2/home/gsmith
#```
In the first example, the account mark is forced to change passwords at the next login. In the second example, the password status for account gsmith is listed. PS means that the account has a password; NP means no password; and LK means the account is locked. In the third example, the home directory for account gsmith is changed to /d2/home/gsmith. In the fourth example, the default shell for jhamid is changed from /bin/sh to /bin/csh. In the fifth example, a user attempts to change his password but is denied because the minimum number of days between password changes has not elapsed. In the last example, a user successfully changes his password.

**Password Complexity**

The passwd command also checks the “complexity” of the password that the user entered. By complexity we mean that all of the following rules must be met:

- The length of the password must be at least PASSLENGTH characters, where PASSLENGTH is a variable that is defined in the /etc/default/passwd file. The default value is set to 6. Only the first eight characters are significant.
- The password must contain at least two lower- or uppercase letters and at least one numeric or special character.
• The password must be different from the userid and any reverse or circular shift of the userid. Upper- and lowercase letters are considered equal for this test. For example, user gsmith cannot change his password to GSMITH.

• The new and old passwords must differ by at least three characters. Upper- and lowercase letters are considered equal for this test.

Of the characteristics listed, only PASSLENGTH can be changed. All of these characteristics can be bypassed if root is changing the password. This is discussed in the next section.

**Root and Password Changes**

The root user is permitted to change the password for any user. When root changes a user’s password, the `passwd` command does not prompt for the old password. This permits root to “reset” a user’s password in circumstances when the user has forgotten his or her password. This also permits root to lock a user’s account by changing the user’s password to a value that the user is unlikely to guess.

---

**Warning**

The password aging and complexity requirements are bypassed when the root user changes a user’s password (or when root changes its own password). The danger of this feature is that a system administrator can set poor-quality passwords. Poor passwords are passwords that are short and/or easily guessed.

---

**EXAM NOTES**

**THINK ABOUT IT . . .**

Why should users change their own passwords?

First, the user will have a greater chance of remembering the password. This will lead to fewer sticky notes with passwords written down. Next, all of the quality rules about passwords (length, complexity, etc.) are enforced when users change their own passwords (no quality checking is performed when root changes a user’s password). Finally, over time the user will have a better understanding of the password quality rules.
Each person who uses a Solaris system is given a *user account*. The user account uniquely identifies the person; this allows the system administrator to tailor each user’s working environment.

Account information is stored in the password file (`/etc/passwd`). Encrypted passwords and password aging information are stored in the shadow file (`/etc/shadow`). Group memberships are defined in the group file (`/etc/group`).

The user known as *root* has all system privileges. The attribute that gives root its power is its user number 0.

The system administrator uses the Admintool program to create, modify, and remove user accounts and groups. The `useradd`, `usermod`, `userdel`, `groupadd`, `groupmod`, and `groupdel` commands are command-line tools used to manage users and groups.

The system administrator has a choice of shells to choose from when setting up each new user. The various shells employ systemwide and per-user initialization files in order to customize each user’s environment. The shells used in Solaris are the Bourne Shell, the C-Shell, and the Korn Shell. Restricted shells are available in instances where users need restricted capabilities.

There are a variety of commands available to see which other users are logged into the system or other systems on the network. They are `who`, `rwho`, `rusers`, and `finger`.

The user account is the sole means for associating an individual to a system. A system administrator creates a *username* for an individual. This username is a name that he uses to identify himself to the system. The administrator also assigns a secret password to the user account.

To use a system, the user must first *log in* with his or her userid and, later, the associated password as requested by the system. When the system is satisfied that the user has furnished the correct userid and password, the system admits the user.

The `-l` and `-n` options to the `ls` command list user/group names or user/group numbers, respectively. The `find` command can be used to find files owned by specific users or groups.

Users can change their password with the `passwd` command. The system will enforce password complexity policies, such as the length of the password and the number and type of characters in the password. These policies are not checked when root changes a user’s password.
Chapter 5 | User Administration

Test Yourself

Multiple Choice

1. Given the output:

root       console      Sep 28 06:13    (:0)
pete       pts/2        Nov  6 05:23    (java)
mark       pts/6        Sep 28 06:13    (acorn)

Which statement correctly describes the command used to generate it?

A. who -u
B. rusers
C. rwhod
D. ps -u

2. Given:

# useradd -D skel_dir=/etc/skel

What is the result?

A. Existing users’ .profile files are replaced by /etc/skel/.profile.
B. The .profile files used for all subsequently created accounts will be copied from /etc/skel/.
C. The default .profile files are deleted.
D. Nothing. The command is invalid.

3. What command is used to search for files?

A. seek
B. search
C. find
D. fs
4. What is the purpose of the `rusers` command?
   A. List which local and remote users are logged on.
   B. Interactively remove user accounts.
   C. Bulk remove user accounts.
   D. List dormant user accounts.

5. Which three are valid expressions of the `find` command? (Choose three)
   A. `-exec`
   B. `-run`
   C. `-path`
   D. `-user`
   E. `-owner`
   F. `-groupid`
   G. `-list`
   H. `-print`

6. Given the passwd file entry:

   `pete:x:100:4:Peter Gregory:/export/home/pete:/bin/sh`

   What is this user’s primary groupid?
   A. 100
   B. 4
   C. x
   D. It is NOT possible to tell.

7. Given the password file entry:

   `pete:x:100:4:Peter Gregory:/export/home/pete:/bin/sh`

   What is this user’s first secondary groupid?
   A. 100
   B. 4
   C. x
   D. It is not possible to tell.
8. Given:

root::0:root
other::1:
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
uucp::5:root,uucp
mail::6:root
tty::7:root,tty,adm
lp::8:root,lp,adm
nuucp::9:root,nuucp
staff::GSSUYVrJ8EKyA:10:pete
daemon::12:root,daemon
sysadmin::14:
nobody::60001:
noauth::60002:
nogroup::65534:

What is the effect of the user mark executing the `newgrp staff` command?

A. The `newgrp` command will create a new group called `staff`, overwriting the existing entry.
B. The `newgrp` command will attempt to create a new group called `staff`, but will be unable to because the group `staff` is password protected.
C. The user will be prompted for the password for the group `staff`.
D. The attempt will fail, because the group `staff` is locked by root.

9. Given the shadow file entry:

pete:NP:11055::::::

What is the meaning of the string `NP`?

A. The user selected a short password which encrypts to the string “NP”.
B. The user's primary groupid is NP.
C. No password is required to log into the account.
D. The account is locked.
10. You need several people to log in to the same account to run an application, and you want files created by one person to be owned by all of the others. However, you do not wish to have the same password shared between all of the users. Which is the correct method to achieve this?
   A. Make all of the users members of the same group.
   B. Create all of the user accounts with the same user ID.
   C. Assign the same home directory for all of the users.
   D. Create one password file entry for all users, and one shadow file entry for each user.

FREE RESPONSE

11. What command is used to add a user account to the system?

12. You removed a user account from the system, but you did not check first to see if the user owned any files. What command and option should you use to locate files owned by that user?