UNIX Administration Course

Copyright 1999 by Ian Mapleson BSc.

Version 1.0

mapleson@gamers.org
Tel: (+44) (0)1772 893297
Fax: (+44) (0)1772 892913
WWW: http://www.futuretech.vuurwerk.nl/

Detailed Notes for Day 1 (Part 3)

UNIX Fundamentals: File Ownership

UNIX has the concept of file ’ownership’: every file has a unique owner, specified by a user ID number contained in /etc/passwd. When examining the ownership of a file with the ls command, one always sees the symbolic name for the owner, unless the corresponding ID number does not exist in the local /etc/passwd file and is not available by any system service such as NIS.

Every user belongs to a particular group; in the case of the SGI system I run, every user belongs to either the ’staff’ or ’students’ group (note that a user can belong to more than one group, eg. my network has an extra group called ’projects’). Group names correspond to unique group IDs and are listed in the /etc/group file. When listing details of a file, usually the symbolic group name is shown, as long as the group ID exists in the /etc/group file, or is available via NIS, etc.

For example, the command:

```
    ls -l /
```

shows the full details of all files in the root directory. Most of the files and directories are owned by the root user, and belong to the group called ’sys’ (for system). An exception is my home account directory /mapleson which is owned by me.

Another example command:

```
    ls -l /home/staff
```

shows that every staff member owns their particular home directory. The same applies to students, and to any user which has their own account. The root user owns the root account (ie. the root directory) by default.

The existence of user groups offers greater flexibility in how files are managed and the way in which users can share their files with other users. Groups also offer the administrator a logical way of managing distinct types of user, eg. a large company might have several groups:

```
    accounts
    clerical
    investors
    management
    security
```

The admin decides on the exact names. In reality though, a company might have several internal systems, perhaps in different buildings, each with their own admins and thus possibly different group names.

UNIX Fundamentals: Access Permissions

Every file also has a set of file ’permissions’; the file’s owner can set these permissions to alter who can read, write or execute the file concerned. The permissions for any file can be examined using the ls command with the -l option, eg.:

```
% ls -l /etc/passwd
-rw-r--r--    1 root     sys     1306 Jan 31 17:07  /etc/passwd

uuugggooo      owner   group     size  date   mod   name
```

Each file has three sets of file access permissions (uuu, ggg, ooo), relating to:

- the files owner, ie. the ’user’ field
- the group which the file’s owner belongs to
- the ’rest of the world’ (useful for systems with more than one group)

This discussion refers to the above three fields as ’user’, ’group’ and ’others’. In the above example, the three sets of permissions are represented by field shown as uuugggooo, ie. the main system password file can be read by any user that has access to the relevant host, but can only be modified by the root user. The first access permission is separate and is shown as a ’d’ if the file is a directory, or ’l’ if the file is a link to some other file or directory (many examples of this can be found in the root directory and in /etc).

Such a combination of options offers great flexibility, eg. one can have private email (user-only), or one can share documents only amongst one’s group (eg. staff could share exam documents, or students could share files concerning a Student Union petition), or one can have files that are accessible by anyone (eg. web pages). The same applies to directories, eg. since a user’s home directory is owned by that user, an easy way for a user to prevent anyone else from accessing their home directory is to remove all read and execute permissions for groups and others.

File ownership and file access permissions are a fundamental feature of every UNIX file, whether that file is an ordinary file, a directory, or some kind of special device file. As a result, UNIX as an OS has inherent built-in security for every file. This can lead to problems if the wrong permissions are set for a file by mistake, but assuming the correct permissions are in place, a file’s security is effectively secured.

Note that no non-UNIX operating system for PCs yet offers this fundamental concept of file-ownership at the very heart of the OS, a feature that is definitely required for proper security. This is largely why industrial-level companies, military, and government institutions do not use NT systems where security is important. In fact, only Cray’s Unicos (UNIX) operating system passes *all* of the US DoD’s security requirements.

Relevant Commands:

```
chown   - change file ownership
chgrp   - change group status of a file
```

```
chmod   - change access permissions for one or more files
```

For a user to alter the ownership and/or access permissions of a file, the user must own that file. Without the correct ownership, an error is given, eg. assuming I’m logged on using my ordinary ’mapleson’ account:

```
% chown mapleson var
var - Operation not permitted

% chmod go+w /var
chmod() failed on /var: Operation not permitted

% chgrp staff /var
/var - Operation not permitted
```

All of these operations are attempting to access files owned by root, so they all fail.

Note: the root user can access *any* file, no matter what ownership or access permissions have been set (unless a file owned by root has had its read permission removed). As a result, most hacking attempts on UNIX systems revolve around trying to gain root privileges.

Most ordinary users will rarely use the chown or chgrp commands, but administrators may often use them when creating accounts, installing custom software, writing scripts, etc.

For example, an admin might download some software for all users to use, installing it somewhere in /usr/local. The final steps might be to change the ownership of every newly installed file so ensure that it is owned by root, with the group set to sys, and then to use chmod to ensure any newly installed executable programs can be run by all users, and perhaps to restrict access to original source code.

Although chown is normally used to change the user ID of a file, and chgrp the group ID, chown can actually do both at once. For example, while acting as root:

```
yoda 1# echo hello > file
yoda 2# ls -l file
-rw-r--r--    1 root     sys           6 May  2 21:50 file
yoda 3# chgrp staff file
yoda 4# chown mapleson file
yoda 5# ls -l file
-rw-r--r--    1 mapleson staff         6 May  2 21:50 file
yoda 6# /bin/rm file
yoda 7# echo hello > file
yoda 8# ls -l file
-rw-r--r--    1 root     sys           6 May  2 21:51 file
yoda 9# chown mapleson.staff file
yoda 10# ls -l file
-rw-r--r--    1 mapleson staff         6 May  2 21:51 file
```

Figure 18. Using chown to change both user ID and group ID.

Changing File Permissions: Examples.

The general syntax of the chmod command is:

```
chmod [-R] <mode> <filename(s)>
```

Where <mode> defines the new set of access permissions. The -R option is optional (denoted by square brackets []) and can be used to recursively change the permissions for the contents of a directory.

<mode> can be defined in two ways: using Octal (base-8) numbers or by using a sequence of meaningful symbolic letters. This discussion covers the symbolic method since the numeric method (described in the man page for chmod) is less intuitive to use. I wouldn’t recommend an admin use Octal notation until greater familiarity with how chmod works is attained.

<mode> can be summarised as containing three parts:

```
U operator P
```

where U is one or more characters corresponding to user, group, or other; operator is +, -, or =, signifying assignment of permissions; and P is one or more characters corresponding to the permission mode.

Some typical examples would be:

```
chmod go-r file       - remove read permission for groups and others
chmod ugo+rx file     - add read/execute permission for all
chmod ugo=r file      - set permission to read-only for all users
```

A useful abbreviation in place of ’ugo’ is ’a’ (for all), eg.:

```
chmod a+rx file       - give read and execute permission for all
chmod a=r file        - set to read-only for all
```

For convenience, if the U part is missing, the command automatically acts for all, eg.:

```
chmod -x file         - remove executable access from everyone
chmod =r file         - set to read-only for everyone
```

though if a change in write permission is included, said change only affects user, presumably for better security:

```
chmod +w file         - add write access only for user
chmod +rwx file       - add read/execute for all, add write only for user
chmod -rw file        - remove read from all, remove write from user
```

Note the difference between the +/- operators and the = operator: + and - add or take away from existing permissions, while = sets all the permissions to a particular state, eg. consider a file which has the following permissions as shown by ls -l:

```
-rw-------
```

The command ’chmod +rx’ would change the permissions to:

```
-rwxr-xr-x
```

while the command ’chmod =rx’ would change the permissions to:

```
-r-xr-xr-x
```

ie. the latter command has removed the write permission from the user field because the rx permissions were set for everyone rather than just added to an existing state. Further examples of possible permissions states can be found in the man page for ls.

A clever use of file ownership and groups can be employed by anyone to ’hand over’ ownership of a file to another user, or even to root. For example, suppose user alex arranges with user sam to leave a new version of a project file (eg. a C program called project.c) in the /var/tmp directory of a particular system at a certain time. User alex not only wants sam to be able to read the file, but also to remove it afterwards, eg. move the file to sam’s home directory with mv. Thus, alex could perform the following sequence of commands:

```
cp project.c /var/tmp           - copy the file
cd /var/tmp                     - change directory
chmod go-rwx project.c          - remove all access for everyone else
chown sam project.c             - change ownership to sam
```

Figure 19. Handing over file ownership using chown.

Fig 19 assumes alex and sam are members of the same group, though an extra chgrp command could be used before the chown if this wasn’t the case, or a combinational chown command used to perform both changes at once.

After the above commands, alex will not be able to read the project.c file, or remove it. Only sam has any kind of access to the file.

I once used this technique to show students how they could ’hand-in’ project documents to a lecturer in a way which would not allow students to read each others’ submitted work.

Note: it can be easy for a user to ’forget’ about the existence of hidden files and their associated permissions. For example, someone doing some confidential movie editing might forget or not even know that temporary hidden files are often created for intermediate processing. Thus, confidential tasks should always be performed by users inside a sub-directory in their home directory, rather than just in their home directory on its own.

Experienced users make good use of file access permissions to control exactly who can access their files, and even who can change them.

Experienced administrators develop a keen eye and can spot when a file has unusual or perhaps unintended permissions, eg.:

```
-rwxrwxrwx
```

if a user’s home directory has permissions like this, it means anybody can read, write and execute files in that directory: this is insecure and was probably not intended by the user concerned.

A typical example of setting appropriate access permissions is shown by my home directory:

```
ls -l /mapleson
```

Only those directories and files that I wish to be readable by anyone have the group and others permissions set to read and execute.

Note: to aid security, in order for a user to access a particular directory, the execute permission must be set on for that directory as well as read permission at the appropriate level (user, group, others). Also, only the owner of a file can change the permissions or ownership state for that file (this is why a chown/chgrp sequence must have the chgrp done first, or both at once via a combinational chown).

The Set-UID Flag.

This special flag appears as an ’s’ instead of ’x’ in either the user or group fields of a file’s permissions, eg.:

```
% ls -l /sbin/su
-rwsr-xr-x    1 root     sys        40180 Apr 10 22:12 /sbin/su*
```

The online book, "IRIX Admin: Backup, Security, and Accounting", states:

```
"When a user runs an executable file that has either of these
 permissions, the system gives the user the permissions of the
 owner of the executable file."
```

An admin might use su to temporarily become root or another user without logging off. Ordinary users may decide to use it to enable colleagues to access their account, but this should be discouraged since using the normal read/write/execute permissions should be sufficient.

Mandatory File Locking.

If the ’l’ flag is set in a file’s group permissions field, then the file will be locked while another user from the same group is accessing the file. For example, file locking allows a user to gather data from multiple users in their own group via a group-writable file (eg. petition, questionnaire, etc.), but blocks simultaneous file-write access by multiple users - this prevents data loss which might otherwise occur via two users writing to a file at the same time with different versions of the file.

UNIX Fundamentals: Online Help

From the very early days of UNIX, online help information was available in the form of manual pages, or ’man’ pages. These contain an extensive amount of information on system commands, program subroutines, system calls and various general references pages on topics such as file systems, CPU hardware issues, etc.

The ’man’ command allows one to search the man page database using keywords, but this text-based interface is still somewhat restrictive in that it does not allow one to ’browse’ through pages at will and does not offer any kind of direct hyperlinked reference system, although each man pages always includes a ’SEE ALSO’ section so that one will know what other man pages are worth consulting.

Thus, most modern UNIX systems include the ’xman’ command: a GUI interface using X Window displays that allows one to browse through man pages at will and search them via keywords. System man pages are actually divided into sections, a fact which is not at all obvious to a novice user of the man command. By contrast, xman reveals immediately the existence of these different sections, making it much easier to browse through commands.

Since xman uses the various X Windows fonts to display information, the displayed text can incorporate special font styling such as italics and bold text to aid clarity. A man page shown in a shell can use bright characters and inverted text, but data shown using xman is much easier to read, except where font spacing is important, eg. enter 'man ascii' in a shell and compare it to the output given by xman (use xman's search option to bring up the man page for ascii).

xman doesn't include a genuine hypertext system, but the easy-to-access search option makes it much more convenient to move from one page to another based on the contents of a particular 'SEE ALSO' section.

Most UNIX systems also have some form of online book archive. SGIs use the 'Insight' library system which includes a great number of books in electronic form, all written using hypertext techniques. An ordinary user would be expected to begin their learning process by using the online books rather than the man pages since the key introductory books guide the user through the basics of using the system via the GUI interface rather than the shell interface.

SGIs also have online release notes for each installed software product. These can be accessed via the command 'grelnotes' which gives a GUI interface to the release notes archive, or one can use relnotes in a shell or terminal window. Other UNIX variants probably also have a similar information resource. Many newer software products also install local web pages as a means of providing online information, as do 3rd-party software distributions. Such web pages are usually installed somewhere in /usr/local, eg. /usr/local/doc. The URL format 'file:/file-path' is used to access such pages, though an admin can install file links with the ln command so that online pages outside of the normal file system web area (/var/www/htdocs on SGIs) are still accessible using a normal http format URL.

In recent years, there have been moves to incorporate web technologies into UNIX GUI systems. SGI began their changes in 1996 (a year before anyone else) with the release of the O2 workstation. IRIX 6.3 (used only with O2) included various GUI features to allow easy integration between the existing GUI and various web features, eg. direct iconic links to web sites, and using Netscape browser window interface technologies for system administration, online information access, etc. Most UNIX variants will likely have similar features; on SGIs with the latest OS version (IRIX 6.5), the relevant system service is called InfoSearch - for the first time, users have a single entry point to the entire online information structure, covering man pages, online books and release notes.

Also, extra GUI information tools are available for consulting "Quick Answers" and "Hints and Shortcuts". These changes are all part of a general drive on UNIX systems to make them easier to use.

Unlike the xman resource, viewing man pages using InfoSearch does indeed hyperlink references to other commands and resources throughout each man page. This again enhances the ability of an administrator, user or application developer to locate relevant information.

Summary: UNIX systems have a great deal of online information. As the numerous UNIX variants have developed, vendors have attempted to improve the way in which users can access that information, ultimately resulting in highly evolved GUI-based tools that employ standard windowing technologies such as those offered by Netscape (so that references may include direct links to web sites, ftp sites, etc.), along with hypertext techniques and search mechanisms. Knowing how to make the best use of available documentation tools can often be the key to effective administration, ie. locating answers quickly as and when required.