THE UNIX PROGRAMMING ENVIRONMENT

Brian W. Kernighan
Rob Pike

PRENTICE-HALL SOFTWARE SERIES

Library of Congress Catalog Card Number 83-62851

Prentice-Hall Software Series
Brian W. Kernighan, Advisor

Editorial/production supervision: *Ros Herion*
Cover design: *Photo Plus Art, Celine Brandes*
Manufacturing buyer: *Gordon Osbourne*

This book was typeset in Times Roman and Courier by the authors, using a Mergenthaler Linotron 202 phototypesetter driven by a VAX-11/750 running the 8th Edition of the UNIX operating system.

10 9 8 7 6 5 4 3

ISBN 0-13-937699-2
ISBN 0-13-937681-X {PBK}

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
EDITORA PRENTICE-HALL DO BRASIL, LTDA., *Rio de Janeiro*
PRENTICE-HALL CANADA INC., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

Exercise 2-3. (Harder) How does the `pwd` command operate? □

Exercise 2-4. `du` was written to monitor disc usage. Using it to find files in a directory hierarchy is at best a strange idiom, and perhaps inappropriate. As an alternative, look at the manual page for `find(1)`, and compare the two commands. In particular, compare the command `du -a | grep ...` with the corresponding invocation of `find`. Which runs faster? Is it better to build a new tool or use a side effect of an old one? □

2.4 Permissions

Every file has a set of *permissions* associated with it, which determine who can do what with the file. If you're so organized that you keep your love letters on the system, perhaps hierarchically arranged in a directory, you probably don't want other people to be able to read them. You could therefore change the permissions on each letter to frustrate gossip (or only on some of the letters, to encourage it), or you might just change the permissions on the directory containing the letters, and thwart snoopers that way.

But we must warn you: there is a special user on *every* UNIX system, called the *super-user*, who can read or modify *any* file on the system. The special login name `root` carries super-user privileges; it is used by system administrators when they do system maintenance. There is also a command called `su` that grants super-user status if you know the `root` password. Thus anyone who knows the super-user password can read your love letters, so don't keep sensitive material in the file system.

If you need more privacy, you can change the data in a file so that even the super-user cannot read (or at least understand) it, using the `crypt` command (`crypt(1)`). Of course, even `crypt` isn't perfectly secure. A super-user can change the `crypt` command itself, and there are cryptographic attacks on the `crypt` algorithm. The former requires malfeasance and the latter takes hard work, however, so `crypt` is in practice fairly secure.

In real life, most security breaches are due to passwords that are given away or easily guessed. Occasionally, system administrative lapses make it possible for a malicious user to gain super-user permission. Security issues are discussed further in some of the papers cited in the bibliography at the end of this chapter.

When you log in, you type a name and then verify that you are that person by typing a password. The name is your login identification, or *login-id*. But the system actually recognizes you by a number, called your user-id, or *uid*. In fact different login-id's may have the same uid, making them indistinguishable to the system, although that is relatively rare and perhaps undesirable for security reasons. Besides a uid, you are assigned a group identification, or *group-id*, which places you in a class of users. On many systems, all ordinary users (as opposed to those with login-id's like `root`) are placed in a single group called `other`, but your system may be different. The file system, and therefore the UNIX system in general, determines what you can do by the

:? □

it to find files in a directory
ate. As an alternative, look
ımands. In particular, com-
›nding invocation of find.
side effect of an old one? □

ı it, which determine who
that you keep your love
. in a directory, you prob-
em. You could therefore
›ssip (or only on some of
ge the permissions on the
:hat way.
every UNIX system, called
ı the system. The special
ısed by system administra-
ılso a command called su
: password. Thus anyone
love letters, so don't keep

ta in a file so that even the
ısing the crypt command
secure. A super-user can
yptographic attacks on the
: and the latter takes hard
.

passwords that are given
ninistrative lapses make it
nission. Security issues are
bibliography at the end of

ify that you are that person
ıtification, or *login-id*. But
led your user-id, or *uid*. In
king them indistinguishable
:rhaps undesirable for secu-
up identification, or *group-*
systems, all ordinary users
e placed in a single group
The file system, and there-
what you can do by the

permissions granted to your uid and group-id.

The file /etc/passwd is the *password file*; it contains all the login information about each user. You can discover your uid and group-id, as does the system, by looking up your name in /etc/passwd:

```
$ grep you /etc/passwd
you:gkmbCTrJ04COM:604:1:Y.O.A.People:/usr/you:
$
```

The fields in the password file are separated by colons and are laid out like this (as seen in passwd(5)):

login-id:*encrypted-password*:*uid*:*group-id*:*miscellany*:*login-directory*:*shell*

The file is ordinary text, but the field definitions and separator are a convention agreed upon by the programs that use the information in the file.

The shell field is often empty, implying that you use the default shell, /bin/sh. The miscellany field may contain anything; often, it has your name and address or phone number.

Note that your password appears here in the second field, but only in an encrypted form. Anybody can read the password file (you just did), so if your password itself were there, anyone would be able to use it to masquerade as you. When you give your password to login, it encrypts it and compares the result against the encrypted password in /etc/passwd. If they agree, it lets you log in. The mechanism works because the encryption algorithm has the property that it's easy to go from the clear form to the encrypted form, but very hard to go backwards. For example, if your password is ka-boom, it might be encrypted as gkmbCTrJ04COM, but given the latter, there's no easy way to get back to the original.

The kernel decided that you should be allowed to read /etc/passwd by looking at the permissions associated with the file. There are three kinds of permissions for each file: read (i.e., examine its contents), write (i.e., change its contents), and execute (i.e., run it as a program). Furthermore, different permissions can apply to different people. As file owner, you have one set of read, write and execute permissions. Your "group" has a separate set. Everyone else has a third set.

The -l option of ls prints the permissions information, among other things:

```
$ ls -l /etc/passwd
-rw-r--r-- 1 root       5115 Aug 30 10:40 /etc/passwd
$ ls -lg /etc/passwd
-rw-r--r-- 1 adm        5115 Aug 30 10:40 /etc/passwd
$
```

These two lines may be collectively interpreted as: /etc/passwd is owned by login-id root, group adm, is 5115 bytes long, was last modified on August 30 at 10:40 AM, and has one link (one name in the file system; we'll discuss links in the next section). Some versions of `ls` give both owner and group in one invocation.

The string `-rw-r--r--` is how `ls` represents the permissions on the file. The first `-` indicates that it is an ordinary file. If it were a directory, there would be a `d` there. The next three characters encode the file owner's (based on uid) read, write and execute permissions. `rw-` means that `root` (the owner) may read or write, but not execute the file. An executable file would have an `x` instead of a dash.

The next three characters (`r--`) encode group permissions, in this case that people in group `adm`, presumably the system administrators, can read the file but not write or execute it. The next three (also `r--`) define the permissions for everyone else — the rest of the users on the system. On this machine, then, only `root` can change the login information for a user, but anybody may read the file to discover the information. A plausible alternative would be for group `adm` to also have write permission on `/etc/passwd`.

The file `/etc/group` encodes group names and group-id's, and defines which users are in which groups. `/etc/passwd` identifies only your login group; the `newgrp` command changes your group permissions to another group.

Anybody can say

```
$ ed /etc/passwd
```

and edit the password file, but only `root` can write back the changes. You might therefore wonder how you can change your password, since that involves editing the password file. The program to change passwords is called `passwd`; you will probably find it in `/bin`:

```
$ ls -l /bin/passwd
-rwsr-xr-x 1 root      8454 Jan  4  1983 /bin/passwd
$
```

(Note that `/etc/passwd` is the text file containing the login information, while `/bin/passwd`, in a different directory, is a file containing an executable program that lets you change the password information.) The permissions here state that anyone may execute the command, but only `root` can change the `passwd` command. But the `s` instead of an `x` in the execute field for the file owner states that, when the command is run, it is to be given the permissions corresponding to the file owner, in this case `root`. Because `/bin/passwd` is "set-uid" to `root`, any user can run the `passwd` command to edit the password file.

The set-uid bit is a simple but elegant idea† that solves a number of security problems. For example, the author of a game program can make the program set-uid to the owner, so that it can update a score file that is otherwise

† The set-uid bit is patented by Dennis Ritchie.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.