Using Regular Expressions to Create User Accounts from Registration Data

For each exercise where you write a program, keep the original program from each exercise and modify a copy for the next exercise.

Contents

1 Background

1.1 Character Classes

We have seen how character classes can match a set of characters. For example, the character class `/[0-9]/` (or `/\d/`) matches any one digit, and `/[0-9][0-9]/` (or, if you like, `/\d\d/`) matches any two digits, one after the other.

1.2 Capturing the Match in Parentheses

This next topic is like gold mining: extracting useful information from among other less useful material.

You can use parentheses in a regular expression, and if there is a match, then the variable `$1` is set to the contents of the first set of parentheses in the regular expression. For example, this code:

```
my $line = "STUDENT REGISTER           2001/02 2nd Term  MODE : PTE";
if ( $line =~ /MODE : ([Pp][Tt][Ee])/ )
{
    print "The mode of study is $1\n";
}
```

prints:

```
The mode of study is PTE
```

1.3 Creating User Accounts

There are many ways of creating accounts on a Linux system; your program could select the next available user and group IDs, manually edit the `/etc/passwd`, `/etc/group` and `/etc/shadow` files, create the user directories, copy the login scripts and other basic account files from `/etc/skel`, and change the ownership to the new account, but a simpler and more portable way is to use the standard `useradd` program that you learned about last year. We can call that from our own Perl programs using the built-in `system` function.

Today we will write a program that will create a user account for each student listed in the artificial student registration data.

1.4 The `system` Builtin Function

Perl provides a number of ways of calling external programs. We will use the `system` builtin function to call `/usr/sbin/useradd` to create user accounts from our student registration data. Refer to slides 82–86 in the Perl lecture slides, and section 1.6 on page 7 of my Perl summary, available at
`http://nicku.org/snm/lectures/perl/perl.pdf`.

2 Procedure

2.1 Installing Linux using Kickstart

1. Install Red Hat Linux version 9 using the `Kickstart` disk that you are given. To do this, simply:

   (a) Ensure that your removable hard disk is properly installed and firmly pushed into its socket

   (b) Turn on your computer

   (c) Insert your kickstart installation disk before the computer boots.

   (d) At the `boot:` prompt, type:

   ```
   boot: linux ks=floppy
   ```

   (e) When prompted, insert the network card driver disk.

   (f) If any prompts appear that say partition tables on `/dev/hdc` are inconsistent, ignore them, as this inconsistency is a result of cloning the disks using Ghost.

   (g) The installation takes place using Kickstart. You may see the chapter from the *Red Hat Linux Customization Guide* for all details about Kickstart. The installation instructions are in a text file on the floppy disk, called `ks.cfg`; you can look at it if you like.

   (h) While the installation takes place, work on the following written tutorial exercises.

```
24-SEP-01 09:36                                       A COLLEGE IN HONG KONG                            GRADE REPORTING   PAGE :   1
VTC GCP1309L-1                                           STUDENT REGISTER                               2001/02 2nd Term  MODE : PTE

A College in Hong Kong
A computing department
2241/2 Higher Certificate in Software Engineering

NO.   NAME                                SEX STUD.NO.  HKID        HOME TEL.  COMPANY                             COMPANY ADDRESS
----- ----------------------------------- --- --------- ----------- ---------- ----------------------------------- ------------------------------ -------------------
 1    LI, Sze Wai                          M  914981001  L700339(4) 24118453
 2    LAW, Kar Hang                        M  943430710  X984028(4) 28543261   Wing Fat Hong International Corp Ltd
 3    YEUNG, Hoi Man                       M  915367894  K383949(9) 21943771   David Hot Blocking Press Ltd
 4    NG, Sze Wing                         M  914925086  W216913(3) 23291992
 5    LAW, Wing Yee                        F  907466937  C977399(7) 21234895   Lucky Industrial (Holdings) Ltd
 6    YAU, KA KEI                          M  973517175  D896832(2) 22818446   Union Plastic Factory Ltd
 7    YIM, Man Wai                         M  981309634  G422563(7) 29105262   Jardine Matheson Ltd
 8    WONG, Kam Lun                        M  946874929  H187711(5) 28376578   System Engineering Ltd
```

Figure 1: The first few lines from the file of artificial student data.

2.2 Tutorial Exercises to Work on While Linux is Installing

Figure 1 shows a few lines from the student information file.

1. For the data shown in figure 1, write a regular expression (with your match enclosed in parentheses) that will select the:

   (a) student number

   (b) Hong Kong ID

   (c) The student's name

   (d) the course code

   The course and year are shown in this case on the sixth line: `2241/2`. The course is 2241; this is the second year of study.

   (e) the year of study

   (f) The company the student works for

   (g) The home telephone number

   (h) The gender of the student

2.3 Finishing the Setup of Your Linux Installation

1. Log into your system as your student account (your student number is your user name; the password you should already know)

2. Set up `sudo` with

```
$ su -c /usr/sbin/visudo
```

3. At the prompt, you should enter the root password, which was set in the Kickstart file `ks.cfg` to the nine-character string "`)3SnhGxv9`".

You may refer to the document I wrote about `sudo`, available here:
`http://nicku.org/ossi/lab/sudo/sudo.pdf`

4. You may wish to change the resolution of the screen using the program `redhat-config-xfree86`.

2.4 Exercises for You After Installation has Finished

1. Download the artificial student data from

`http://nicku.org/snm/lab/regular-expressions/`
`artificial-student-data.txt`. There is also a link to this file from the subject web site. This file is in the old format of the student registration system, but contains no real data about any student. We will work toward generating system accounts from this file during this class.

2. Write a Perl program that can read all the lines of this file when it is given as a command line parameter, and display it on standard output. For example, if your program is called `printit`, then the following command will display the content of the big file of student data:

```
$ printit artificial-student-data.txt
```

This will be a very short program: between 1 and three lines of code should do it.

3. Make a copy of your program, and modify it so that it prints only the lines that contain a number with eight or more digits.

4. Modify the last program so that it prints all lines that contain a Hong Kong ID.

5. Modify this last program further so that it prints only the Hong Kong ID, and nothing else for each line. Each Hong Kong ID should be printed one to each line. There should be no other output from your program.

6. Write a Perl program that can read all the lines of this file when it is given as a command line parameter, and print the student numbers only, one to each line.

7. Make a copy of your program, and modify it so that it prints only the names of the students, one to each line, with no extra spaces either at the beginning or end of each name.

8. Using the manual page for the `useradd` program as a guide, modify your previous programs so that your program *prints* one `useradd` command for each student, using the student ID as the login ID, the Hong Kong ID as the password, and the name of the student as a comment.

Note that our Linux systems require that the user/group names must start with a letter, and may not contain colons, commas, newlines or any non-printable characters. The maximum length of the user ID is 32 characters; that of the group ID is 16. To meet this requirement, make the user ID be the first letter (made lower case) of the family name, followed by the student ID.

9. Modify this last program further so that it uses the built-in Perl function `system` to execute the `useradd` command as well as print the command. Type `perldoc -f system` to read about this important function. You will probably need the function `hash_md5_password` described in the appendix on page 5.

> I suggest that you take one or two student record lines from the student information file, and run your program on that file, generating accounts for them at first.
>
> Only after you have tested your program on a subset of the data, and have demonstrated that it works, then run it on all the data in the data file.

10. Modify your last program so that it reports an error message if the execution of any `useradd` command is unsuccessful.

11. Modify your program further so that it creates a group (see `man groupadd` and `man gpasswd`) for each year found in the data file, and for each course, and makes each student a member of these groups as their secondary group. For a student in year 1, create a group `year1` if it does not already exist, and make the student a member of that group. For a student in the course `2241`, create a group with the name `ict2241` and add the user to that group.

A Appendix: `hash_md5_password`, Provided for Reference Only

Figure 2 on the next page shows a function that I have adapted, plus a stub program to test it. You can download a copy of it from `http://nicku.org/snm/lab/regular-expressions-make-accounts/hash_md5_password.txt`. There is a link on the subject page that you can download it from.

You do not need to study this function in detail, but I explain it here for completeness.

You will probably want to copy and paste lines 14–23 into your progam that calls `useradd`. `useradd` provides the option `-p` *hashed_password* to allow creation of accounts with passwords. But you need to *hash* the password yourself. This function does that. Try running the program, typing in some text, and seeing the output as a hash of the text, suitable for use in the `/etc/shadow` file that holds hashed passwords.

As you can see from the call to the function in the stub program on line 27, the function `hash_md5_password` takes one scalar parameter. This is put into the variable `$clear_text_password` on line 16.

The lines 17–19 need explaining.

To attack a set of hashed passwords, a technique that works well with Windows NT passwords is to build a dictionary of hashed words. With such a dictionary, it would take a short time to find all the trivial passwords in a system.

Linux avoids this danger by appending a 48-bit random number called a `salt` to the plain text before it is hashed. To build an already hashed dictionary would require hashing each word $2^{48} = 281{,}474{,}976{,}710{,}656$ times, making such an attack impractical. Note that a salt does not decrease the time an attacker needs to search for a single user's password.

```
1   #! /usr/bin/perl -w
2
3   # Example program to generate MD5 hashed passwords suitable for use in
4   # /etc/shadow on a Linux system.
5   # You could pass the output of this function to useradd -p xxxxx,
6   # where xxxxx is the output of hash_md5_password().
7   # Based on /usr/share/doc/samba-2.2.1a/examples/LDAP/ldapsync.pl,
8   # distributed with samba.
9   # A portable alternative is the module Crypt::PasswdMD5, available
10  # through the cpan program.
11
12  use strict;
13
14  sub hash_md5_password($)
15  {
16      my $clear_text_password = shift;
17      my $salt = join ’’,
18      (’.’, ’/’, 0..9, ’A’..’Z’, ’a’..’z’)[rand 64, rand 64, rand 64, rand 64,
19                                           rand 64, rand 64, rand 64, rand 64];
20      $salt = ’$1$’.$salt.’$’;
21      my $hashed_password = crypt( $clear_text_password, $salt );
22      return $hashed_password;
23  }
24
25  # This code is a stub just to test the function:
26  our $clear_password = shift;
27  our $hashed_password = hash_md5_password( $clear_password );
28
29  print "MD5 Hash of ’$clear_password’ is ’$hashed_password’\n";
```

Figure 2: A function to generate MD5 Hashes of passwords, and a stub program to test it.

So lines 17–19 build a list of 64 characters (used for *mime*, or *base 64* encoding), and then build an array of eight of these characters, using the builtin `rand` function as an index.

Line 20 encloses the salt between a literal ‘`$1$`’ and a dollar sign. This is the way the `crypt` standard library function determines that the password has been hashed with MD5 rather than with the weaker DES hash.

We then pass the plain text and the salt to the `crypt` standard library function.