Supplementary Assignment: Processing Identical Files

1 Submission

Deadline: by 4.30 pm, Tuesday 27 July 2004. Late submissions are counted as failure.

Where:

- Paper submission at the office, C440, and
- Online at `http://nicku.org/perl2/submit.cgi`.

What: Submit:

- A printout of your program to the office
- A tarball or ZIP file submitted online, as described above. The tarball or ZIP file should contain two shell scripts.

Cheating: Your work *must* be original. Copying will be *severely* dealt with. I *will* use the plagiarism detection tools at `http://www.cs.berkeley.edu/~aiken/moss.html`, and rigorously compare your work with all previous assignment submissions. Of course, you are welcome to use the code I have provided in lectures and workshop sessions and emails I have sent you previously.

If your work fails the plagiarism detection, and if I find evidence of copying, then you will fail the CA component of this subject, and will need to repeating one year of study of the subject.

2 Assignment Requirements

This assignment requires you to write two shell scripts.

2.1 Delete Files in Current directory of which there are Copies Below

Write a shell script that:

1. Takes a list of filenames on the command line that exist in the current directory;
2. For each of these files, the program:
   (a) calculates an MD5 sum of the file contents;

(b) searches the subdirectories below the current directory for a file of the same name, and calculates the MD5 sum of the contents of each of the other files with the same name;

(c) The program prints the details of each pair of files, indicating whether their contents are the same or not;

(d) if the contents are the same, the program will delete the copy of the file in the *current* directory.

I wrote a script like this to check that the photos downloaded from my digital camera really were on the web site before deleting them.

2.2 Link All Duplicates

Write a shell script that:

1. Searches all files in the current directory and below that are on the same partition as the current directory
2. Calculates the MD5 sum of each file
3. Determines which files have the same file contents
4. For each set of files that have the same file contents and which are on the same partition
   (a) Replaces all copies of the file by a *hard link* from one of the copies of the file.

Note that the `-xdev` option to `find` may be helpful.

A script like this could be useful to save disk space when dealing with large amounts of data that are being distributed by HTTP or FTP, but where copies of the same files need to appear in different directories; for example, for source code bundled with binary software packages for different architectures will for the most part be identical for each architecture.

3 MD5 Sum

The MD5 sum is a *one way hash*, described in RFC 1321. It takes any size of input, and calculates a 128-bit value as output. This calculation is made in such a way that it is extremely unlikely that two different files have the same MD5 sum. Even if one bit in one byte of the data in a file changes, the MD5 sum will be totally different.

POSIX systems have a command, `md5sum`, that calculates the MD5 sum in hexadecimal of the contents of the input files. It displays the results so that the 32-character MD5 sum is displayed first, then the file name, one per line of output.

See

```
$ man md5sum
```

Here is example output from `md5sum`:

```
$ md5sum *
042e4581dcb7f30a256ab961e6643a2f  assignment-ca-supp-delete.tex
8b977b41c35c11ee0e3e06b951fa2e89  assignment-ca-supp-delete.tex~
91f357e9ce91baf7070b1be6e8744d93  assignment-ca-supp-delete.toc
c09bbef217efa9a3ebeef30f375b6577  Makefile
```

3.1 Other Useful POSIX Commands

Many other POSIX commands besides `md5sum` may be useful when working on this assignment. Here are a few: `find`, `xargs`, `sort`, `uniq`, `grep`, and the built-in commands `test` or `[...]`, `echo`. You will find that other programming structures are useful here, particularly pipes to connect these commands together.

4 Guide to Testing

If you want to fail this subject, the best thing to do is to never test your work, and blindly submit it and hope it works. It probably doesn't.

Test *incrementally* (as you write the program). As you build each part, test it, make sure it does what you expect. Don't just sit down, write your whole program, then start testing it.

You may use the `-x` option to the shell to enable tracing of your program as it executes.

Create a deeply nested directory structure, and copy lots of files into this. Make copies of various files into different locations in this directory structure. Keep a record of all the files you have created, and record what you expect your program to do. Record the MD5 sums of all the files.

Predict what your program should do, and write that down.

Run your program, and verify that it behaved as you expect.

Change the directory structure, change some of the files, and test again.

Test as much as you can.

5 Marking Scheme

Your submission will be marked as follows.

For the first program,

criteria	**mark**
Handle the list of filenames on the command line, and calculate the MD5 sum of each, ensuring that they exist	10
Search for matching files and calculate the MD5 sum of matches	30
displaying details of matching files	20
deleting the files if this is appropriate, and never deleting the wrong file	40

For the second program,

criteria	**mark**
Find all matching files that have the same file contents	50
ensure that they are on the same partition (filesystem)	10
replacing each copy with a link to one of the copies	40

The marks for each program are equal, and the marks listed above for meeting requirements constitute 70% of the marks for this assignment. The remaining 30% shall be determined by the quality of the submission according to the following scheme:

criteria	mark
Elegant design that is as simple as possible	40
Use of pipes to connect the commands together, avoiding temporary files	30
Robust design, never making any system call unchecked	15
Good structure of program: code divided into simple functions that each do one simple, easily specified action, identifiers have meaningful names,...	10
Additional flexibility (i.e., behaviour can be changed using options)	5

6 Getting Help

6.1 Documentation

All these commands are documented with `man` pages, as well as (in more detail) in `info` format. I usually use Emacs to read `info` documentation, as I described in section 6.9 on page 192 of my workshop notes (see below for the link to the workshop notes). The documentation for the Bash shell itself is in one big `man` page, or alternatively, available in `info` format.

There are at least two very useful books available online at `http://tldp.org/guides.html`. There is the *Bash Guide for Beginners* available in HTML at `http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html` and in other formats too. There is also the *Advanced Bash-Scripting Guide* available in HTML at `http://tldp.org/LDP/abs/html/index.html`. Both contain plenty of examples.

Of course, my workshop notes may be helpful here: `http://nicku.org/ossi/lab/workshop-notes.pdf` as well as my lecture notes on shell programming: `http://nicku.org/ossi/lectures/shell/shell-slides.pdf`.

See Module 5 of my workshop notes for details about hard links. Also see the notes at `http://nicku.org/ossi/lab/sym-link/sym-link.pdf` for more about hard and soft links.

6.2 Asking Questions

If you have any questions about the assignment, please ask them in person, by phone to the office (2436 8576) or by email. I will send the reply to all students if that seems helpful, but I will conceal the identity of the person who asked the question. I welcome any questions.

6.3 I am Not Available in the Office After 16 July 2004

Please note that I will not be available in the office after 16 July 2004. However, I will read your email and answer it with care.