lab_intro Ineluctable Introduction

Due: Sunday, August 30 at 11:59 PM
Doxygen for lab_intro

Assignment Description

Today's lab will serve as your introduction to the EWS Linux environment and SVN.

Getting Started with Linux

Welcome to CS 225. In this class we will be using the Linux operating system for development and the terminal command line.

We expect that most students will not be familiar with the Linux command line. This skill is useful not only for CS 225 course but your future courses also. To learn it, you will need to regularly attend lab sections and to complete all of your MPs. In addition, we will now cover some basic commands you'll need.

First, you will need to open a terminal to enter the commands we give in this specification. The terminal can be opened by going to `Applications > System Tools > Terminal` located at the top of your screen. You can also drag this icon to the menu bar at the top of the screen for the future. You will also notice that if you navigate to a folder using Places and right-click inside it there is an option Open In Terminal which will open the terminal. The terminal is a text-based interface to a computer. You run programs by typing their names, and sometimes parameters such as files to open, instead of clicking on icons and buttons.

Type

```
TERMINAL
pwd
```

and press enter. You'll see the folder you are currently located in, called the working directory. Since you haven't moved anywhere yet, this is your home directory.

Let's make a directory (folder) for CS 225. Type the following in the terminal:

```
TERMINAL
mkdir cs225
```

Now let's look at the contents of the folder we're in (still your home directory):

```
TERMINAL
ls
```

You'll probably see several folders (in blue). One of these should be `cs225/`. Let's open `cs225/`:

```
cd cs225/
```

If you type `pwd` again, you will see you are now in `cs225/` inside of your home directory.

Useful Linux commands cheat sheet:

```
pwd               # Print Working Directory - shows the folder you are c

ls                # List - lists the files in your current folder<br>

cd FOLDER         # Change Directory to FOLDER - opens FOLDER<br>
cd ..             #  - opens the parent folder of your current location<
cd                #  - opens  your home folder<br>

gedit FILE &      # Opens (or creates) FILE in gedit, a simple text edit
vim FILE          # Opens (or creates) FILE in vim, a simple modal text 

rm FILE           # Remove - deletes FILE<br>
rm -r FOLDER      #        - deletes a folder and its contents<br>
mv FILE1 FILE2    # Move - moves/renames FILE1 to FILE2<br>
cp FILE1 FILE2    # Copy - copies FILE1 to FILE2<br>

make PROGRAM      # Compiles PROGRAM using the rules in the Makefile fil

xdg-open IMAGE    # Opens IMAGE in the default image viewer<br>

clear             # Clear the terminal screen (Ctrl-l also works)<br>
reset             # Reset and clear the terminal<br>
```

Keyboard Shortcuts for Bash:

```
Ctrl + a : Navigates to the beginning of the line<br>
Ctrl + e : Navigates to the end of the line<br>
Alt + b  : Back (left) one word<br>
Alt + f  : Forward (right) one word<br>
Ctrl + u : Clears the line from the cursor to the beginning<br>
Ctrl + c : Kills the current command being executed (useful if run int
tab      : Attempts to autocomplete your command
```

More commands can be found here. Note that some of these commands may be unavailable for Mac OS X.

Check out the list above for useful Linux commands, and try using them. You can learn more by looking at the tutorials posted on the Resources page.

Checking Out the Assignment

In this class we will be using a version control system called SVN (Subversion). It is a useful tool for managing changes to source code, especially when multiple people are changing the code at the same time. In this class we use it to distribute and hand in lab and MP assignments.

SVN works similarly to a website server. You can download (check out) files, upload (commit) files, and change which files will be uploaded/downloaded (add/remove).

When files are checked out (downloaded) from the SVN repository, they are collectively referred to as the “working copy”. Like your local copy of a website, changes made to the files in the working copy only change the working copy. In order to save these changes to the SVN server, they must be committed (uploaded) to the repository.

Each student in the class is given their own personal directory in the repository. If you haven’t already, follow the instructions on the MP 1 page for checking out your directory. Then, from there, run

TERMINAL
```
svn up
```

You should now have a new directory (folder) in your working directory called `lab_intro` containing the given files for `lab_intro`.

Ask your instructor

If you do not have access to SVN yet (you get a `403 Forbidden error`), let your lab instructor know, then find a nice person sitting near you and ask to follow along. You can download `lab_intro` here: `lab_intro.tar.gz`. Note you will not be able to check in these files since you did not get them from an SVN server. This is OK since `lab_intro` is not graded for credit.

To open this directory, use the `cd` (change directory) command:

TERMINAL
```
cd lab_intro/
```

To display the files in this directory, use the `ls` (list) command:

TERMINAL
```
ls
```

Common Coding Mistakes

Please read the specification carefully! The file names, executable names, program output, and public member functions must be named exactly as we ask. A single typo in these could result in a grade close to a zero on a MP submission. However, compiling the provided test cases will help detect these sort of typos.

Opening a Text Editor

In order to open and edit a text file, you need to open a text editor. `vim` is one of the most common text editors on Linux and the one that we recommend that you learn to use for this course. You will need to know at least the basics of `vim` (or another command line editor) if you use PuTTY in the future. But if you don't like `vim`, you can try `gedit` (below). Or if you already have a favorite editor, such as `emacs`, feel free to keep using it. We have some information and recommended tutorials on `vim` on the Resources page.

You only need to use one text editor, but we provide a variety of commands for the various options. We will be creating a file called `partners.txt`. You will create a `partners.txt` file for each MP in CS 225. (Note that opening a nonexistent file creates it in the following examples.)

The following command can be used to open `partners.txt` using `vim`:

```
TERMINAL
vim partners.txt
```

`vim` has two main modes: command mode and insertion mode. Pressing `Escape` will take you to command mode, which lets you type `vim` commands, and which `vim` starts in. Insertion mode lets you edit text. To get to insertion mode, press `i`. Use the arrow keys to position the cursor, and type to write. When you are done editing, press `Escape` to get back to command mode, and type `:wq` to write (save) and quit.

The following command can be used to open `partners.txt` using `gvim` (a graphical version of `vim`):

```
TERMINAL
gvim partners.txt &
```

More `vim` keybindings are available here. There's a great learning curve for `vim`, so don't feel expected to understand how to use it right away.

The following command can be used to open `partners.txt` using `gedit`:

```
TERMINAL
gedit partners.txt &
```

Writing the RGBAPixel class

Create a class called `RGBAPixel` whose functionality is described in the Doxygen documentation. Take care to note that all of the members of the class are public. Following convention, the class definition should be placed in a file called `rgbapixel.h`, and the member function implementations should be placed in a file called `rgbapixel.cpp`.

Common Coding Mistakes

You'll want to include standard library header file which contains the `uint8_t` type. You can include the header by doing the following in `rgbapixel.h`:

```cpp
#include <cstdint>
using std::uint8_t;
```

If you don't do this you will probably get errors about `uint8_t` not referring to a valid type.

Common Coding Mistakes

Do not forget to use double inclusion guards!!! A common error in the past has been a compilation failure due to doubly included header files.

Double inclusion guards look like the following (included in a `myfile.h` file):

```cpp
#ifndef MYFILE_H
#define MYFILE_H
// ... code goes here
#endif
```

This is general to all `*.h` files you create. For instance, you should have these in your `image.h` file (make sure to change `MYFILE` to something more relevant!).

Don't forget to `svn add` these two files!

Writing the PNG Manipulation Functions

For this lab, you'll find a partner and get a function from the list in `main.cpp` to implement. Feel free to look up any algorithms, or make up one that seems reasonable. You will not be graded on it if your image is the same as what we intended — be creative!

Here's a sample function, which makes the image more green.

```cpp
PNG getGreenComponent(PNG original)
{
   for (int x = 0; x < original.width(); x++)
   {
      for (int y = 0; y < original.height(); y++)
      {
         original(x, y)->red = 0;
```

```
            original(x, y)->blue = 0;
        }
    }
    return original;
}
```

Helpful Functions

```C++
image.resize(1024, 768);    // Changes the size of the PNG named i
image.height();             // Returns the height of the PNG named
image.width();              // Returns the width of the PNG named
image(i, j)->red = 255;     // Sets the red channel at coordinates
int red = image(i, j)->red; // Gets the value stored in the red ch
                            //   of the PNG named image and stores
```

Compiling the Code

To compile your code, copy and paste the following command into your terminal:

```
make
```

You will learn more about how to compile your projects in MP 1.

Testing the Code

After compiling your code, an executable named `lab_intro` should be located in your working directory. To test your code, run `lab_intro`:

```
./lab_intro
```

This will make the png image `output.png` in your current directory. You can view it with:

```
xdg-open output.png
```

Submitting Your Work

We'll be grading most of the labs this semester, and all of the MPs. This part of the lab is critical. If you do not submit your lab or MP code correctly, it could cost you most or all of the points. However, this lab in particular is not worth credit, so if you do not have an EWS or SVN account, follow along with a neighbor. This lab will be graded to help introduce you to our grading procedures (however it will still not be worth credit in your course grade).

To facilitate anonymous grading, do not include any personally-identifiable information (like your name, your UIN, or your NetID) in any of your source files.

For MPs, we sometimes allow you to collaborate with one other currently enrolled student, but we require each student to submit their work separately. In addition we require that you submit a text file called `partners.txt` that contains a list of NetIDs of the student that you collaborated with on the MP and yourself. This file should have one NetID per line and no additional content or whitespace. If you did not collaborate with anyone, the file should contain only your NetID. If we have to manually correct the format of your `partners.txt` file on a real MP submission, you may lose points. Failure to cite collaborators violates our academic integrity policy; we will be aggressively pursuing such violators.

At the beginning of class, you already checked out a working copy of the code from the class repository. To submit your work, you will need to commit the changes that you have made. First be sure that your working directory is the same that you had checked out from the repository (`lab_intro/`). You can view the status of the working copy by typing:

```
TERMINAL
svn status
```

This command asks for the SVN status of all files and directories within the working directory. It returns a list of modified(`M`), unknown(`?`) , and added(`A`) files. If no output appears, then your repository is already up-to-date with your working copy. Any new files that you created and were not previously in the working copy are labeled unknown and may need to be added to it. `partners.txt` falls into this category and should be added by using the following command:

```
TERMINAL
svn add partners.txt
svn add rgbapixel.cpp
svn add rgbapixel.h
```

You can confirm changes made to the working copy by again checking the status. Now that all the important files are either added, modified, or up-to-date, the changes are committed to the repository using the following command:

```
TERMINAL
svn commit -m "lab_intro submission"
```

The `-m` flag tells SVN that the quoted message following it should be used as a description of the changes you made. If you forget this flag, you will see a strange message `"Editor not set"`. The string may be empty (`""`), but it is a good practice to add comments to each revision of your code. In the event of needing to revert to previous revision, the comments will help you differentiate between the revisions.

Your repository directory can be viewed from a web browser from the following URL: https://subversion.ews.illinois.edu/svn/fa15-cs225/NETID/, where `NETID` is your university NetID. It is important to check that the files you expect to be graded are present and up-to-date in your SVN copy.

Testing with Monad

We will be using an autograder called Monad this semester. When we release test cases for you, you can run them yourself with your copy of Monad. When we run the autograder, it will also be using Monad to test your code. We encourage you to read more about Monad on the Testing with Monad page.

To check out Monad for this lab, go to your `cs225` directory (not your `lab_intro` directory) and run:

TERMINAL

```
svn co https://subversion.ews.illinois.edu/svn/fa15-cs225/_shared/mona
```

To run Monad, first go into the `monad` folder, then `svn update`, then run Monad as follows:

TERMINAL

```
cd monad
./monad lab_intro
```

This will run our test cases on your code. For more information on running Monad and interpreting the results, see Testing with Monad.