Setting Up Your Work Environment

1 Overview

In this class, we use the ***Computer Science Department Network*** (***CSDN***) for all programming projects and for accessing certain programs I have written as demonstration programs for the class. The CSDN is the collection of computers named `cslabN (N=1,2,...,27)`, the gateway machine, `eniac`, and the file server, `biocs`, on which they depend. All of these machines "live" in the domain `cs.hunter.cuny.edu`, so their fully qualified names are names like `eniac.cs.hunter.cuny.edu`.

This class has its own directory in the file system, whose full pathname is

```
/data/biocs/b/student.accounts/cs493.65.
```

Within this directory are subdirectories of various kinds. Among them is a directory named `bin`, which contains executable programs that you can run. (In Unix, "bin" is short for "binaries", i.e., executables.) For example, there is a program in `bin` named `submithwk_cs49365`. To run this program from any directory in the file system, you would have to type

```
/data/biocs/b/student.accounts/cs340_sw/bin/submithwk_cs49365
```

which is tedious to type. In general, to run anay program in this directory you would have to type the full pathname of that program. This is inefficient. This document describes the changes you can make to the file that customizes your `bash` login shell so that you can just type the command (program) name and nothing else.

2 Customizing Your PATH Environment Variable

`bash` keeps track of your actions by saving various pieces of information in specially named variables called ***environment variables***. A ***variable*** is just like a mathematical variable – it is a symbol that can have a single value at any time. In UNIX, variables are named using ordinary words; environment variables are usually in all UPPERCASE. The collection of environment variables and their values is called your ***environment***.

One important variable is your `PATH` environment variable. The `PATH` environment variable tells `bash` where to look for commands that you type. It is a text string containing a colon-separated list of absolute paths to directories (or the special symbol "."). `bash` looks through this list of directories in left to right order when you type a program name to see if it exists in one of them. The first one it finds is the one it uses.

It would be slow if `bash` did this every time, so `bash` also caches in a hash table the commands that you have used recently to avoid the costly linear lookups.

To make it easier to run the programs in the class's `bin` directory, you can modify your `bash` configuration file so that you can type just the name of the program on the command line to execute it, instead of having to type that very long pathname for it. That file is your `.bashrc` file, located in your home directory.

Following are instructions to do this. In these instructions, the dollar sign '$' in the commands is the prompt character by the system. You do not type it.

1. Save a copy of your `.bashrc` file in case you make a mistake following these instructions:

   ```
   $ cp ~/.bashrc ~/.bashrc_old
   ```

2. Open the file `.bashrc` in the editor of your choice on the system.

3. Navigate to the bottom of the file and enter the following lines EXACTLY, without making any changes. Do not add white space unless it is there, and do not remove it:

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.

```
# Set up PATH
# Use the bash function pathmunge to avoid duplications and
# put paths in the best position
pathmunge () {
if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
    # the directory to be added is not already in PATH. Eureka!!!
    if [ "$2" = "after" ] ; then
        PATH=$PATH:$1
    else
        PATH=$1:$PATH
    fi
fi
}
# Modify PATH variable to include path to cs493.65/bin directory
if [ -e /data/biocs/b/student.accounts/cs493.65/bin ] ; then
    pathmunge /data/biocs/b/student.accounts/cs493.65/bin after
    export PATH
fi
```

4. The lines above define a `bash` function named `pathmunge`. In English, to *munge* is to transform or mix up data. This function expects a directory pathname as its first argument and the optional word "`after`" as its second argument. `pathmunge()` checks if the directory is already in the `PATH` variable and if it is, it does nothing. If it is not, then it either prepends the directory to the `PATH` or appends to the `PATH` depending on whether the word "`after`" is supplied. For example, if the current value of `PATH` is the string "`/bin:/usr/bin:/usr/local/bin:`", and I type

   ```
   pathmunge ~/bin  after
   ```

   then the new value of `PATH` will be the string "`/bin:/usr/bin:/usr/local/bin:~/bin`".

   After the definition of the `pathmunge()` function, it has lines that add a new directory to your `PATH` variable so that any executable in that directory can be run just by typing its name without a leading path. The directory that gets appended is the one that I use for class-specific commands.

5. Save the file and quit.

6. Once you have modified the `.bashrc` file, you must run the following command to tell the `bash` program to reread its contents:

   ```
   source ~/.bashrc
   ```

7. You can now type a commnd such as `submithwk_cs49365` without the full path to it.

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.