Assignment 0: Linux Environment

Due: 11:59PM Thursday, September 15, 2022

Grader (TA): Ruxin Wang <rwang31@lsu.edu>

In this "assignment", you will become familiar with a few tools required to work in the Unix-like environment. Take your time and try to understand each step as fully as possible since its purpose is for you to learn how to use the programming environment used in the class. There is no need to finish it one sitting.

To do this work, you will need to log into the classes Linux machine **classes.csc.lsu.edu**. All common OSes have methods of getting remote terminal access:

- Linux/OS X: use ssh. Open up a terminal window, and type:
  - `ssh username@classes.csc.lsu.edu`
- Windows: install and use PuTTY

If you want to access the server from off campus, you need to connect to the LSU VPN first. You can find more information about how to use the server on http://csc.lsu.edu/~davidst/cs2700/classes.html.

Directory hierarchy operations:

1. Log in as outlined above.

2. After logging in, check what your current directory is by using the following command (i.e., print working directory).
   `pwd`
   After login, your working directory should be your home directory `/classes/cs3501/username`.

3. Create a subdirectory (AKA a folder) under your home directory with the following command.
   `mkdir prog0`

4. Read about the mkdir command with the following command.
   `man mkdir`

5. Look at all the files/directories in your current directory with the following command.
   `ls`

Do you see the `prog0` subdirectory that you just created? Execute `ls -l` to see more information on the files/directories in your home directory. Read a little of the man page to see what information this command is telling you.

6. Change directories to your `prog0` subdirectory with the following command.
   `cd prog0`

7. Make the directory `foo` under your `prog0` directory.

8. Change directories to your `foo` directory.

9. Make the directory `bar` under your `foo` directory.

10. Change directories to your `bar` directory.

11. Create a file containing the word "hello" using the echo command along with redirection.
    `echo "hello" > hello.txt`

12. Display the contents of the created text file one line at a time using the cat command.
    `cat hello.txt`

13. Read the man page on `cat`.

14. Change to your home directory using `cd`. Notice that a `cd` with no arguments changes to your home directory. The tilde (~) is shorthand for your home directory, so instead of referring to your `prog0` directory as `/classes/cs3501/username/prog0` you can use `~/prog0`.

15. Make an alias, `cdbar`, to change directories to your newly created bar directory with the command: `alias cdbar="cd ~/prog0/foo/bar"`

16. Change to your `bar` directory with `cdbar`, verify that you are there with `pwd`, then change back to your `prog0` directory.

17. Create a tar file (like a zip file) of all directories and files you have created using the following command.
    `tar cvf foo.tar foo`

18. If you are masochistic, horrify yourself by attempting to understand the man page on `tar`.

19. Scope the size of the files using `ls -l`

20. Compress the tar file using the following command.
    `bzip2 foo.tar`

21. Scope the `bzip2` man page.

22. Scope the size of the files now: `foo.tar` will now be smaller and be called `foo.tar.bz2`.

23. Delete all files in your bar directory using the following command.
    `rm foo/bar/*`

24. Scope the `rm` man page.

25. Delete the bar directory using the following command.
    `rmdir foo/bar`

26. Scope the `rmdir` man page.

27. Delete the `foo` directory.

28. Verify that these directories have been deleted.

29. **Important**: Create a file called **NAME.txt** containing your full name and student ID# so the graders can associate you with your login name.

30. Submit your tar file using LSU's auto-submit feature.
    `~cs3501_lee/bin/p_copy 0`

31. Verify files and dates submitted.
    `~cs3501_lee/bin/verify 0`

32. Untar all the stuff we just deleted.
    `bunzip2 -c foo.tar.bz2 | tar xvf -`

33. Verify your deleted directories/files are back.

34. Change to your home directory and make a subdirectory `tmp` where you can play with stuff that you will delete later.

Editing with vim:

1. Change to the `~/tmp` directory, execute `vimtutor` and follow the directions in order to get an introduction to the vi editor (vim is one of the fanciest modern variants of Unix's original vi editor).

Compiling and executing C programs:

1. Type in the following C program into `show-bytes.c`. Comments (started with /* and ended with */) are optional and can be changed or omitted without changing the program.

show-bytes.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef unsigned char *byte_pointer;

void show_bytes(byte_pointer start, int len) {
        int i;
        for (i = 0; i < len; i++)
                printf(" %.2x", start[i]);
        printf("\n");
}

void show_int(int x) {
        show_bytes((byte_pointer) &x, sizeof(int));
}

void show_float(float x) {
        show_bytes((byte_pointer) &x, sizeof(float));
}

void show_pointer(void *x) {
        show_bytes((byte_pointer) &x, sizeof(void *));
}


void test_show_bytes(int val) {
        int ival = val;
        float fval = (float) ival;
        int *pval = &ival;
        show_int(ival);
        show_float(fval);
        show_pointer(pval);
}
```

show-bytes.c

```
void simple_show_a() {
        int val = 0x87654321;
        byte_pointer valp = (byte_pointer) &val;
        show_bytes(valp, 1); /* A. */
        show_bytes(valp, 2); /* B. */
        show_bytes(valp, 3); /* C. */
}

void simple_show_b() {
        int val = 0x12345678;
        byte_pointer valp = (byte_pointer) &val;
        show_bytes(valp, 1); /* A. */
        show_bytes(valp, 2); /* B. */
        show_bytes(valp, 3); /* C. */
}

void float_eg() {
        int x = 3490593;
        float f = (float) x;
        printf("For x = %d\n", x);
        show_int(x);
        show_float(f);

        x = 3510593;
        f = (float) x;
        printf("For x = %d\n", x);
        show_int(x);
        show_float(f);
}

void string_ueg() {
        const char *s = "ABCDEF";
        show_bytes((byte_pointer) s, strlen(s));
}

void string_leg() {
        const char *s = "abcdef";
        show_bytes((byte_pointer) s, strlen(s));
}

void show_twocomp() {
        short x = 12345;
        short mx = -x;

        show_bytes((byte_pointer) &x, sizeof(short));
```

show-bytes.c

```
        show_bytes((byte_pointer) &mx, sizeof(short));
}

int main(int argc, char *argv[]) {
        int val = 12345;

        if (argc > 1) {
                if (argc > 1) {
                        val = strtol(argv[1], NULL, 0);
                }
                printf("calling test_show_bytes\n");
                test_show_bytes(val);
        } else { /* without argument */
                printf("calling show_twocomp\n");
                show_twocomp();
                printf("Calling simple_show_a\n");
                simple_show_a();
                printf("Calling simple_show_b\n");
                simple_show_b();
                printf("Calling float_eg\n");
                float_eg();
                printf("Calling string_ueg\n");
                string_ueg();
                printf("Calling string_leg\n");
                string_leg();
        }
        return 0;
}
```

2. Compile the program in `show-bytes.c` calling the executable `show-bytes`, using the command:

   ```
   gcc -Wall -ansi -g -o show-bytes show-bytes.c
   ```

   This will compile the C file, create an object file and then link this object code to the C libraries all in one step to create the executable. The object code file will not be written to the user's directory in this case. Note that we could separate the compile and link steps by:

   (a) `gcc -Wall -ansi -g -c show-bytes.c` (compile `show-bytes.c` to object file, stored as `show-bytes.o`).

   (b) `gcc -ansi -g -o show-bytes show-bytes.o` (link object file with system libs to create executable).

   In this case, `show-bytes.o` will persist in user's directory until deleted manually. If our program is spread over multiple files, just repeat step (a) for each file, and then list all files during step (b).

3. Run the program without any argument:
   `./show-bytes`
4. Run the program with an argument:
   `./show-bytes 1`
5. There is **nothing to submit** for this phase.

Compiling and executing C programs (Another example):

1. Type in the following C program into `twos_complementary.c`. This program aims to show the sign bit of two's-complement integers.

twos_complementary.c

```
#include <stdio.h>

int main()
{
    int i = 100;
    int i_sign = i >> 31;
    i_sign = i_sign & 0x1;
    printf("Sign bit of %d: %d\n", i, i_sign);

    int j = -1000;
    int j_sign = j >> 31;
    j_sign = j_sign & 0x1;
    printf("Sign bit of %d: %d\n", j, j_sign);

    return 0;
}
```

2. Go to the saved c file's directory by using the `cd` command:

3. Compile the program in `twos_complementary.c` calling the executable `twos_complementary`, using the command:

```
gcc twos_complementary.c -o twos_complementary
```

This will compile the C file, create an object file, and then link this object code to the C libraries all in one step to create the executable. The object code file will not be written to the user's directory in this case.

4. After generating the executable named `twos_complementary`, run it using the command:

```
./twos_complementary
```

Important notes:

- You should **NOT** send any assignment-related emails to the instructor. The TA has full control over the assignment grading process. The instructor will NOT reply to assignment-related emails and will NOT answer to assignment-related questions during his office hours. You can still ask high-level concepts.

Only for Honors Option:

No additional question in this assignment.