

## Lab 03

# Automation, Libraries and Command Line Options

### Goals

In lab 3 we will be developing and building a complete (but small) software project using standard Unix software development tools but focusing on using the Gnu C compiler, Make and the Gnu debugger. The project we will be building is a simple example of a standard Unix filter program, that will read from standard input, transform the data, and write the results to standard output. The program will also accept a number of command line flags or options in order to modify its behavior.

Thus the goals of this project are twofold:

- 1) Learn a bit more about standard Unix commands and command line options from the inside out, especially the concept of a standard I/O filter that accepts data from standard input and writes filtered results to standard output.
- 2) Learn the basics of a simple project developed in a Unix software development environment, especially about using Makefiles to automate building and organizing a project.

### Instructions

For this and all future labs, all of your work will be done and submitted through your own student repository on the nisl class server. If you have not done already, you need to check out a working copy of your personal repository before you can begin this lab.

Use the in-class lab time not only to find the answers to the questions and complete the given tasks, but to practice your skills in using the gcc compiler, using standard Unix libraries, and using make and Makefiles. The first portion of the lab is to be completed in class so that you may ask the instructor for help and suggestions for any problems you may be experiencing.

### Exercise 3.0: Preliminaries

Check out a working copy of your students repository if you haven't already done so. Your student repository will have a url designation of the form:

<http://nisl.tamu-commerce.edu/repo/csci553/username>

Where you need to substitute your own nisl account username for the last portion above.

In your working copy, create and add to your repository a directory called *lab03* (make sure you name it exactly as shown, make sure you not only create the directory, but you run the appropriate command to add the directory to be under version control). All work for this lab is to be added and checked into the *lab03* directory of your repository.

Once you have created and added the lab03 directory to your working copy, extract the initial source files needed for this project from the classfiles directory:

```
[harry@nisl harryrepo]$ cd lab03
[harry@nisl lab03]$ tar xvfz /home/csci553/classfiles/lab03.tgz
filter.c
main.c
options.c
palindrome.c
reverse.c
filter.h
options.h
palindrome.h
reverse.h
```

```
[harry@nisl lab03]$ svn add *
A      filter.c
A      filter.h
A      main.c
A      options.c
A      options.h
A      palindrome.c
A      palindrome.h
A      reverse.c
A      reverse.h
```

Once you have performed these steps, go ahead and commit all of these newly added files as a single subversion operation (hint use the commit operation but don't specify any file). I will be looking for a single commit operation adding all of these files to your lab03 directory in your log to determine if you accomplished this step correctly. Don't forget to specify a log message stating that these additions were for lab 3, part 3.0.

### Exercise 3.1: Creating a Project Makefile

From part 3.0 you can see that your project consists of 5 source files (and 4 header files) in the C programming language. The source files implement a simple palindrome filter command, that reads lines from standard input, and echos the line to standard output if the line is a valid palindrome.

As we have seen in a previous lecture, the commands to compile these source files into an executable using the Gnu gcc compiler to create individual object files, then separately link the object files are:

```
[harry@nisl lab03]$ gcc -c filter.c
[harry@nisl lab03]$ gcc -c options.c
[harry@nisl lab03]$ gcc -c palindrome.c
[harry@nisl lab03]$ gcc -c reverse.c
[harry@nisl lab03]$ gcc -c main.c
[harry@nisl lab03]$ gcc -o palindrome main.o filter.o options.o palindrome.o reverse.o
```

The individual source files can be compiled into object files using the `-c` option, then the object files can be linked into an executable and the resulting executable given the name 'palindrome' using gcc as a linker with the `-o` option. However, this is a lot of typing just to build your simple project. And anytime you make a change, you would need to perform some or all of the above steps again. We will automate this build process by creating a Makefile for your simple project.

Create a Makefile in the lab03 directory. Don't forget that action lines need to be preceded by tab characters in your Makefile. You need to define the following targets for your makefile: `all`, `palindrome` and `clean`. The `clean` target is a standard cleaning task, the result of invoking `make clean` should be that all object and executable files are removed from the current directory.

The `all` target should be the default target. It will be a phony target whose only dependency will be the `palindrome` target. The `palindrome` target will be the target and name of the executable that your automated build will create. `palindrome`, as you see in the above example of building the project by hand, depends on the 5 object files that make up the project. The action for the `palindrome` target should be to link the object files together into the `palindrome` executable. You are required to use automatic variables to create this target's rule.

In addition to the targets/rules discussed, you will need rules that will create object files from the source `.c` files. You are required to do this with a single rule using patterns (`%` in a makefile) and automatic variables. This target will specify how to create object files from any generic `c` source file for your project.

If you create your Makefile correctly, you should be able to perform the following steps:

```
[harry@nisl lab03]$ make clean
rm -f -v *.o palindrome
[harry@nisl lab03]$ make
gcc -c main.c
gcc -c options.c
gcc -c filter.c
gcc -c palindrome.c
gcc -c reverse.c
gcc -o palindrome main.o options.o filter.o palindrome.o reverse.o
[harry@nisl lab03]$ make clean
rm -f -v *.o palindrome
removed `filter.o'
removed `main.o'
removed `options.o'
removed `palindrome.o'
removed `reverse.o'
removed `palindrome'
[harry@nisl lab03]$ make palindrome.o
gcc -c palindrome.c
[harry@nisl lab03]$ make main.o
gcc -c main.c
[harry@nisl lab03]$ make palindrome
gcc -c options.c
gcc -c filter.c
gcc -c reverse.c
gcc -o palindrome main.o options.o filter.o palindrome.o reverse.o\
```

**WARNING:** You are not yet done with this step until you add and commit your Makefile to your repository. When you are satisfied with your Makefile's operation, you should add and commit it to your repository before continuing to the next step.

### Exercise 3.2: Debugging Palindrome

After building your palindrome command, try running it. The palindrome command reads from standard input, and echos any palindromes it finds back to standard output. Thus if you run palindrome alone, nothing will happen unless you type something into the terminal as input:

```
[harry@nisl lab03]$ ./palindrome
reviver
Segmentation fault
[harry@nisl lab03]$
```

Whoops, that Segmentation fault means that the program crashed due to a memory access violation of some kind. At this point you could begin adding print statements to try and track down where the segmentation fault is occurring. However, if you use a debugger, you can run the program and inspect the stack after the fault occurs, to directly determine where the program is crashing.

Remember, in order to debug a program with a symbolic debugger, you need to compile the object files with debug information turned on using the `-g` option. For a small project like this, I normally define a make variable, something like `DBGFLAG = -g`. Using a variable at the top of your Makefile allows you to turn on and off debugging simply by modifying this variable. Add the `DBGFLAG` variable, and use the variable in the appropriate rule of your makefile so that debugging is turned on when you recompile your project. The result should look like this:

```
[harry@nisl lab03]$ make clean
rm -f -v *.o palindrome
removed `filter.o'
removed `main.o'
removed `options.o'
removed `palindrome.o'
removed `reverse.o'
removed `palindrome'
[harry@nisl lab03]$ make
gcc -g -c main.c
gcc -g -c options.c
gcc -g -c filter.c
gcc -g -c palindrome.c
gcc -g -c reverse.c
gcc -o palindrome main.o options.o filter.o palindrome.o reverse.o
```

Run the debugger of your choice. I would suggest that you use kdbg (the gui front end to the gnu debugger) if you have no preference. Run the program and cause it to crash (you will need to input a line of text and hit return). You should be able to examine the stack to determine the location where the memory access fault occurred. Fix the bug and recompile the program. Hint: the bug is due to an array access that is beyond the legal bounds of the array. If you successfully fix the bug, you should be able to run the program without it crashing:

```
[harry@nisl lab03]$ ./palindrome
reviver
reviver
yay
yay
I
I
cat
cattac
cattac
[harry@nisl lab03]$
```

The lines that were typed in by the user are italicized above. The program responds by echoing the line back to standard output if the line is a valid palindrome. You can terminate input by sending an EOF character to the program (Ctrl-d generates an EOF from the terminal).

**WARNING:** When you are satisfied that you have fixed the bug, you should commit your changes to the repository. You should have fixed the bug in one of the source files, plus you should have made changes to your Makefile. Before committing your Makefile back to the repository you should turn off debugging. If you used a make variable as required, this means you can turn off debugging simply by setting the variable to empty: `DBGFLAG =`.

### Exercise 3.3: Parsing command line options using `c`

The palindrome command supports a number of command line options. You can see a list of the supported options by invoking the command with the `-h` flag:

```

[harry@nisl lab03]$ ./palindrome -h
Usage: ./palindrome [options]
Filter input looking for palindromes

-h          show this help message and exit
-i          ignore case when testing for palindrome
-s          ignore single character palindromes like 'a'
            or 'I', equivalent to -l 1
-f <file>  process input from <file> rather than standard
            input

```

As you can see from the above output, the palindrome supports a number of options in addition to the -h flag for requesting help. The -i and -s flags accept no parameters and allow you to ignore case when testing for a palindrome, and to ignore single letter palindromes respectively. The -f option allows you to specify a file that should be read from for input, rather than reading from standard input.

Try the following

```

[harry@nisl lab03]$ cat /usr/share/dict/words | ./palindrome
[harry@nisl lab03]$ cat /usr/share/dict/words | ./palindrome -i
[harry@nisl lab03]$ cat /usr/share/dict/words | ./palindrome -s
[harry@nisl lab03]$ cat /usr/share/dict/words | ./palindrome -i -s
[harry@nisl lab03]$ cat /usr/share/dict/words | ./palindrome
[harry@nisl lab03]$ ./palindrome -f /usr/share/dict/words

```

These commands show how you can use the various options to search the shared dictionary for palindromes. Question: If you wanted to count the number of palindromes found in the above examples, how would you do this?

There are several conventions governing the use of command line options for standard commands that we have not explicitly mentioned yet in class but they you may have intuitively noticed. In general command line options come in one of two flavors, short options and long options. Short options use a '-' followed by a single character. For example, for the grep command the short option to cause each line of output to be prefixed by the line number the match was found on is '-n'. The grep command has a corresponding long option that does the same thing, namely '--line-number'. Notice that this option works as a flag, by default line numbers or not shown. But if you specify the -n/--line-number flag to grep, then the default behavior is overridden and line numbers will be shown. Besides simple flags, some command line options accept a parameter. An example with the grep command again, the '-C NUM' option is used to tell grep to display a NUM of lines of context around lines that have a match of the search pattern. The corresponding long option is '--context=NUM'. Notice that for a long option that requires a parameter, an equal (=) sign is used, however for a short option with a parameter only a space is present (don't ask why, this is just historically how the convention evolved). For our palindrome command, -s and -i are short options and they are simple flags, while -f <filename> is a short option that accepts a parameter.

In this portion of the lab you will be adding two command line options to the palindrome program. The palindrome.c and reverse.c source files you have seen before in class, these simply implement the test for a string to determine if it is a palindrome. The new code resides in the options.c and filter.c source files. First take a look at the main.c. You will notice that the main function simply performs two actions, first it calls a function in order to parse command the command line options, then it calls another function to do the actual work of filter the input to test for palindromes.

### *Exercise 3.3.1: Adding an option with a parameter*

Examine the option.c and option.h source file. These files contain code for parsing command line options and passing the results. Instead of defining global variables, all options and flags are kept in a struct called Options, the definition of which can be found in options.h

We will first add an option that accepts a parameter. If you examine the option parsing code, the `-i` flag is implemented by setting the `options.minPalindromeLength = 2`. This integer parameter controls the filtering process, and causes any string that is smaller than the indicated length to be ignored. We are going to add a `-l <length>` option to allow the user to specify any arbitrary minimum length cutoff for palindromes they wish to identify.

If you examine the `parseOptions` function, you will see that it is basically a large switch statement. The switch statement is based on the results of a function call to the system `getopt()` function. You may want to read a bit about the `getopt()` function, it is in section 3 of the man pages (where most of the c system library function calls can be found):

```
[harry@nisl lab03]$ man 3 getopt
```

Basically `getopt()` provides basic parsing capabilities for command line options. Notice that the way `getopt` works is by passing the `argc` and `argv` variables, which hold the raw command line information, to the function. The important part to notice, if you want to parse a new option, is the 3<sup>rd</sup> string parameter to `getopt`, which looks like "hisf:". Do you see what this string does? Recall that the palindrome command accepts the options `-h`, `-i`, `-s`, and `-f <filename>` (note that `-f` is the only option that accepts a parameter). To have `getopt` parse a new short option, you simply need to modify this string to add the option, and you need to add an appropriate case statement to the switch below.

Add the `-l` option. This option should accept an integer as a parameter, e.g. `-l <length>`. The result of specifying `-l` by the user should be that the `minPalindromeLength` is set to the value indicated on the command line. Look at how the `-f` option is handled in the switch statement for some ideas. You will need to use the `atoi()` function to convert the string into an integer value.

Also, make sure you update the usage message (in the `usage()` function) to give help about your new `-l` option. When you are done, you should be able to invoke the `palindrome` command with your new option like this:

```
[harry@nisl lab03]$ make
gcc -c options.c
gcc -o palindrome main.o options.o filter.o palindrome.o reverse.o
[harry@nisl lab03]$ ./palindrome -h
Usage: ./palindrome [options]
Filter input looking for palindromes

    -h          show this help message and exit
    -i          ignore case when testing for palindrome
    -s          ignore single character palindromes like 'a'
                or 'I', equivalent to -l 1
    -l <length> only find palindromes of length size or greater
    -f <file>  process input from <file> rather than standard
                input

[harry@nisl lab03]$ ./palindrome -f /usr/share/dict/words -l 10
kinnikinnik
tat-tat-tat
```

**WARNING:** When you are satisfied that your `-l` option is working, you should go ahead and commit your changes to the repository. I will expect to see a revision at this point where you have only implemented the `-l` option in your project.

### *Exercise 3.3.2: Adding a flag option*

Adding a flag option that does not require a parameter should in some ways be easier. However, in order to implement this command line option, you will need to add a new member to the `Options` struct, and add a bit of code to the `filterInput()` function in the `filter.c` file.

You will be adding a flag '-v' that will invert the sense of palindrome matching. This is similar to the '-v' flag for grep. The basic idea is that if -v is specified, the palindrome command should find and echo lines that are NOT palindromes.

Besides adding the v flag to getopt, and adding a case statement to your parseOptions() switch, you will need to do the following. You will need to add a new member to the Options struct in options.h. This should be a bool member variable. Call it something like invertMatch. The default value for invertMatch should be false (make sure you set the default at the top of the parseOptions() function. If the -v flag is given, however, then invertMatch should have a value of true.

Besides adding the flag, and the Option struct member to hold its value (and again don't forget to update the usage message), you will also need to change the logic of the filterInput() function in the filter.c file. The change is fairly simple. The line should be printed to standard output in either of two cases. If the line is a palindrome, and invertMatch is false, then this is the normal operation and it should be printed out. Likewise, if the line is NOT a palindrome, but invertMatch is true, then this is the inversion case, and the non-palindrome line should be displayed in this case.

If you successfully add this option, you should be able to use it in a manner similar to this:

```
[harry@nisl lab03]$ ./palindrome -h
Usage: ./palindrome [options]
Filter input looking for palindromes

  -h          show this help message and exit
  -i          ignore case when testing for palindrome
  -v          invert sense of matching, e.g. find and
             display non-palindromes
  -s          ignore single character palindromes like 'a'
             or 'I', equivalent to -l 1
  -l <length> only find palindromes of length size or greater
  -f <file>   process input from <file> rather than standard
             input

[harry@nisl lab03]$ ./palindrome -f /usr/share/dict/words -v -l 30
dichlorodiphenyltrichloroethane
half-embracinghalf-embracingly
pneumonoultramicroscopicsilicovolcanoconiosis
```

**WARNING:** Once again, you are not finished with 3.3.2 until you commit your changes to add the -v option. When you are satisfied that your new -v flag is working correctly, commit your changes to the repository.

### Exercise 3.G.1: Tagging a Project Revision

This task is only required for graduate students. As with lab 1, undergraduates may also do this section for extra credit, if desired. For undergraduates you can receive up to 5 points of extra credit for completing 3.G.1-3.G.4

In this portion of the lab, we will attempt to learn more about using libraries or archives, and how they are commonly used in standard Unix software development projects. Currently all of our source, header and build files for the palindrome project should be in the lab03 directory. We are going to split the project into a library, consisting of the palindrome string functions, and the filter portion of the palindrome command.

At this point in time, your work on your project constitutes a significant milestone. Before we begin the major changes of restructuring the project layout, it would be a good idea to tag this revision with a label. A tag is just a “snapshot” of a project in time. In the example below, make sure you replace the username 'harry' with your own username.

First change to the root of your working copy, something similar to:

```
[harry@nisl lab03]$ cd
[harry@nisl ~]$ cd harryrepo/
```

Create a new directory to hold your tagged releases

```
[harry@nisl harryrepo]$ svn mkdir tags
A      tags
[harry@nisl harryrepo]$ svn ci
```

Use `svn log` to determine the last revision of your last commit for lab section 3.3.2. To be completely certain what your last revision was, you need to determine the revision number assigned to the commit of your last change. This should have been the commit of the `options.h`, `options.c` and `filter.c` file to implement the `-v` option for lab part 3.3.2. You can do an `svn log` of any of the files to see which revision number was assigned:

```
[harry@nisl harryrepo]$ svn log -v lab03/options.c
-----
r920 | harry | 2008-02-06 11:53:31 -0600 (Wed, 06 Feb 2008) | 3 lines
Changed paths:
  M /csci553/harry/lab03/filter.c
  M /csci553/harry/lab03/options.c
  M /csci553/harry/lab03/options.h
```

Lab 3.3.2, added the `-v` option to invert sense of matching a palindrome.

... <snip> ...

In this case, revision 920 was where harry committed his changes for lab 3.3.2 where he added the `-v` option (make sure you find the most recent revision number, it should be the first one that is printed out by the `log` command). Your revision number will be different.

Now use the `copy` command to create a tag/branch of this version of your lab03. This is long to type in so be careful (I have added some newlines to make more readable, but this is all one line. Of course you can provide the log message using an editor rather than the `-m` option, if you like):

```
[harry@nisl harryrepo]$ svn copy -r 920
http://nisl.tamu-commerce.edu/repo/csci553/harry/lab03
http://nisl.tamu-commerce.edu/repo/csci553/harry/tags/lab03-release-1.0
-m "Tagging the 1.0 release of my lab03 work, includes
    implementation of the -v and -l flags, and a project Makefile"
```

Committed revision 922.

Now, if you were successful, if you do an update you should see you `lab03-release-1.0` tag populated with the snapshot of our palindrome project:

```
[harry@nisl harryrepo]$ svn update
A      tags/lab03-release-1.0
A      tags/lab03-release-1.0/palindrome.h
A      tags/lab03-release-1.0/options.h
... <snip> ...
```

### Exercise 3.G.2: Reorganize Repository

In this portion of the assignment, you will be reorganizing your project layout, and splitting the code into a library and an executable. This should give you a better feel for how actual, more complex projects are organized and built.

Change back to your lab03 directory in your working copy. We will first add several directories under lab03, which will act as sub-modules of the project, and allow us to develop a build and installation

procedure. Create and add directories named lib, include, libpalindrome and palindromefilter to your repository working copy. The lib and include directories will be used for your build and install procedures, and are really only working directories. The libpalindrome will hold the code to create an archive/library of the palindrome and reverse string functions. Finally, the palindromefilter will hold the option parsing and input filtering code along with the main function to actually build the palindrome command.

We want the source code for reverse.[ch] and palindrome.[ch] to be moved into the libpalindrome directory, as they will form the core of the static library. Likewise, the main.c and the options.[ch] and filter.[ch] should be moved into the palindromefilter directory. Use the subversion rename command to move these source files to the appropriate directories. You should also rename the Makefile you created and place it into the palindromefilter directory along with main, options and filter.

Once you have moved the source and Makefile there should be nothing remaining in the lab03 directory (except possible some old object files and/or executables which you may go ahead and delete). Once you have moved/reorganized the files as indicated, you should go ahead and commit these changes to the repository.

### Exercise 3.G.3: Create libpalindrome.a Library Subproject

Change into the libpalindrome subdirectory. If you completed the previous step correctly, you should have the c source files and headers for palindrome and reverse in this directory. Your task is to create a new Makefile for this subproject. The Makefile should have a default all target which depends on the libpalindrome.a target. The libpalindrome.a target is the actual rule to build the static library. As we saw in a previous lecture, the action for this rule should use the ar command to build an archive from the reverse and palindrome object files. You will need pattern rules, as we did for part 3.1, in order to create the object files from the reverse.c and palindrome.c source files.

In addition to the build, your subproject Makefile should support a standard clean target rule. You will also create a install rule. The install rule should simply copy the libpalindrome.a static library to the lib directory, and all of the header files to the include directory. You should use make variables to specify the target locations for the include and lib directories.

If your Makefile is constructed correctly, you should now be able to build and install your libpalindrome static library subproject using a sequence of commands like this:

```
[harry@nisl libpalindrome]$ make clean
rm -f -v *.o libpalindrome.a
removed `palindrome.o'
removed `reverse.o'
removed `libpalindrome.a'
[harry@nisl libpalindrome]$ make
gcc -c palindrome.c
gcc -c reverse.c
ar -crv libpalindrome.a palindrome.o reverse.o
a - palindrome.o
a - reverse.o
[harry@nisl libpalindrome]$ make install
cp -a libpalindrome.a ../lib
cp -a *.h ../include
```

**WARNING:** As before, at this point you should commit your changes to the repository once you have your libpalindrome subproject building and installing correction. Also remember, in addition to making a few modifications, you are adding a new Makefile to this subproject. If you haven't done so already, you should add the Makefile to your subversion repository, before committing all of your changes.

### Exercise 3.G.4: Modify filter subproject to use libpalindrome static library

Notice that in the last step of installing the static library, the libpalindrome.a archive was copied to the lib directory, and the header files were copied to the include directory. This is the standard way that C libraries are shared and installed in a Unix system. Any project should be able to reuse the libpalindrome static library by linking against it and by including the appropriate location where the headers for the library are found.

We now need to modify slightly the Makefile in the palindromefilter subproject to instead link against the libpalindrome library. Your Makefile will have the same targets as it did from step 3.3, however you will need to modify slightly the commands for building the object files and for linking the executable together. In particular, when you compile the object files, you need to include a `-I../include` option, so that the header files may be found. Likewise for the linking phase, you need to have the flags `-L../lib -lpalindrome` in order to link your files with the libpalindrome archive.

If you modify your Makefile correctly, you should be able to build the palindromefilter subproject in this manner:

```
[harry@nisl palindromefilter]$ make clean
rm -f -v *.o palindrome
removed `filter.o'
removed `main.o'
removed `options.o'
removed `palindrome'
[harry@nisl palindromefilter]$
[harry@nisl palindromefilter]$ make
gcc -I ../include -c main.c
gcc -I ../include -c options.c
gcc -I ../include -c filter.c
gcc -o palindrome main.o options.o filter.o -L ../lib -lpalindrome
[harry@nisl palindromefilter]$
[harry@nisl palindromefilter]$ ./palindrome -f /usr/share/dict/words -l 10
kinnikinnik
tat-tat-tat
```

**WARNING:** At this point you have finished the third lab. As before, you need to commit your changes at this point before you are finished with the lab.