The Small Shell:

This archive has a little bit of code to illustrate what a modern Linux shell architecture is requires. The beauty of the original Unix operating system can be seen from the flexibility and power of the software architecture. Very little source is required to provide a very powerful shell. Even the portions provided are fairly small if you implement the primary functionality.

This C source for the shell implements:

1. Command-line loop with history and prompt (parts using the GNU readline library)
2. Complex command-line parsing (all homegrown)
3. Some signal management (catches ^C using standard API)
4. Redirection for < and > (homegrown using file API)
5. Pipes between commands (using standard API)

What this shell lacks is:

1. Globbing of command arguments (future additions)
2. Job control (more future additions)

In any case, there are some things that need your attention.

1. (10 points)

The first problem is to extend the command line to recognize the path when typing in a command. At the moment, you must the full path for commands, such as:

```
> /bin/ls -l|/usr/bin/sort -r|/usr/bin/wc
```

Find a way to alter this behavior so that the program will search for arg[0] in the exec function using the PATH environment variable. When you make the change, leave comments in the file to say what you did to change it. After all the change is really only very small.

2. (20 points)

The < and > redirection is already implemented. The parseline function already recognizes the >> to do append redirection and places a flag in the command so that you can be use it when found (look in the API for command -- command.h -- to find the function). To finish this problem, implement the >> redirection in the program.

3. (30 points)

There is no build-in commands to manage environment variables. In this, you will add the following commands:

a. print <VAR> - prints value of environment variable <VAR>

```
ex: > print PATH
    /bin:/usr/bin:/usr/local/bin
    >
```

b. set <VAR> <VALUE> - set value of environment variable <VAR> to <VALUE>

```
ex: > set NAME "Linus Torvalds"
    >
```

NOTE: <VALUE> can contain special characters (i.e. ' ') but they must be escaped or placed in single or double quotes. (This actually makes your job easier)

c. printenv - print the environment (all variables and values)

```
ex: > printenv
    PATH=/bin:/usr/bin:/usr/local/bin
```

```
        NAME=Linus Torvalds
        ...
        >
```

Don't forget to add the built-in help for each new command!

4. (30 points)
   Next is to extend the built-in commands to replicate some of the commands already available on the command-line:

   a. md <PATH> - create directory given by <PATH>

```
      ex: > md ./ssh/src
          >
```

   b. rd <PATH> - remove directory given by <PATH>

```
      ex: > rd /home/linus.torvalds/ssh
          ERROR: directory '/home/linus.torvalds/ssh' not empty
          >
```

      NOTE: the directory must be empty

   c. del <FILE> - delete <FILE>, if directory, recursively deletes

```
      ex: > del a.out
          Are you sure you want to delete 'a.out'? (y/n) y
          >
```

      NOTE: This command will always ask you if you want to delete the file.

      Make sure you catch all errors for these commands and report them.
      And don't forget to add the built-in help for each new command!

5. (5 points)
   Looking at the source code, what would you need to do to add redirection to stderr. Explain your answer in detailed steps telling what files you would change and what steps were required to accomplish this.

6. (5 points)
   Looking at the source code, what would you need to do to add an "alias" command. That is you want to be able to do:

```
     > alias change_directory cd
     > change_directory /etc/init.d
     >
```

   Explain your answer in detailed steps telling what files you would change and what steps were required to accomplish this.