

homework 3

Homework 3: Write your own shell

[Assignment checkout.](#)

Background

The command line is a flexible, powerful tool that takes advantage of Unix-style OS's "everything is a file" and "compose lots of little programs to do complicated tasks" strategies.

In this assignment, you will build a shell that can compose multiple programs, manage jobs, and best of all, it will use the modern `posix_spawn` interface rather than the sad old `fork` interface.

The programming part

For this assignment you are to write a simple program that will act as a shell. The program should:

- Display a command prompt and read in a command line from the user (the prompt must be `CS361 >` with exactly one space between `CS361` and `>`, otherwise it cannot be detected by the autograder)
- Your shell must support basic piping like the unix shell.
 - i. `$ command arg1 arg2 ...` Run command, with stdin and stdout connected to their usual files. We already implemented this requirement during week 6 lab. Use this as a starting point to complete the rest of the requirements
 - ii. `$ command args < input_redirection` Run the command, but connect `stdin` to the contents of the file `input_redirection`.
 - iii. `$ command args > output_redirection` Run the command, but connect `stdout` to the contents of the file `output_redirection`.
 - iv. `$ command args < input_redirection > output_redirection` Do both of the previous tasks at the same time.
 - v. `$ command1 | command2` Run `command1` as in #1, but redirect the output of `command1` as input to `command2`.

- vi. `$ command1 ; command2` Run `command1` as in #1, wait for it to finish, and then run `command 2` as in #1.
- vii. When any non-background `command` finishes, the parent should output the prompt again as normal. However, if the user executes the special command `?`, the shell should output `\npid:%d status:%d\n` (with the proper relevant values inserted into the format string), and then print the prompt again and wait for more input. For composed commands, the output should only be shown for the final command in the sequence.
- viii. Your shell should handle the following signals:
 - `SIGINT` - Generated by `Ctrl-C`. This signal allows a user to terminate a running program. Your shell should **NOT** exit when user presses `Ctrl-C` or the process receives `SIGINT` but simply report that the `SIGINT` signal has been received by the shell. If the process receives `SIGINT`, you must print the string **"caught sigint"** on a line by itself, and then show the prompt again.
 - `SIGTSTP` - Generated by `Ctrl-Z`. This signal allows a user to pause a running program. Your shell should not exit when user presses `Ctrl-Z` or the process receives `SIGTSTP` but simply report that the `SIGTSTP` signal has been received by the shell. If your shell receives `SIGTSTP`, you must print the string **"caught sigtstp"** on a line by itself, and then show the prompt again.

Non-goals:

- The shell does **NOT** need to support restoring stopped background processes or handling multiple chained pipes (e.g. `command1 | command2 | command3`). Essentially, if anything not matching the use cases/requirements above is in question, you probably do not have to do it. Due to the breadth of this assignment, it is difficult to clarify every case in writing. Please ask in office hours or on Piazza if you have questions or need clarification.

Note: Throughout the skeleton code, you will find useful `TODOs` and `Hints`. Use them as a guide to complete the tasks.

A word on academic integrity

Previous solutions, or other students' work, are off limits and any cheating will be prosecuted to the fullest extent of university rules. *There are many pre-written shells on the Internet.* If we find you using them, we will give you an F in the class. Don't look at them, don't use them. It is not worth it. Remember that the assignments in this course are meant to provide structure for your own learning, not just a seal of approval that will get you your next job.

Requirements

It is **highly** recommended you develop and test your code on systems1 or the Ubuntu 18.04 devcontainer we've provided.

When you turn in the assignment, the TA should be able to compile your program by running' `gcc -Wall -Werror -o spawnshell spawnshell.c` in the `hw3` directory (case sensitive). You are free to include any files e.g. `hw3.h` , it just has to compile using the above command.

Testing

One very important component of all software engineering is testing. Rather than doing our testing for this assignment via the autograder right away, we will be showing you how you can test the full functionality of your shell. You're encouraged to write your own test cases to determine whether your code is working correctly.

Using input redirection to test a shell

One important concept we'll go over this week is file redirection. In short, file redirection allows you to read your program's input from a file, rather than from the keyboard.

For instance, if you have a file `input` that has the lines:

```
pwd
ls
date +%Y%m%d
```

You can feed it to a new instance of a shell program by redirecting the input from that file by adding the `<` character as an argument, then the name of the file:

```
ckanich@home-desktop:~/repos/shell-skel$ sh < input
```

In addition to using `sh` , you can also run the skeleton code for the assignment in the same way, and see that it works properly. If you would like to automate the process of comparing the output of a real shell with your shell, you can use the `diff` program. Running `man diff` at the

command line or googling `diff` command `linux` can give you a good introduction to how it works.


Grading

Grading will happen using an autograder on Gradescope. Note that we will not be releasing the autograder early on during the assignment: learning how to test your own code is an important aspect of this assignment.

Due Date

This assignment is due 3/1/2021, 4:59:59 PM in time zone America/Chicago.

See the [syllabus](#) for the late turn-in policy. This assignment is worth just as much as every other homework, so getting as much credit on it as possible is important (don't turn it in late!).

 [Edit this page](#)