

# Programming Assignment 1: Shell

CSE3320

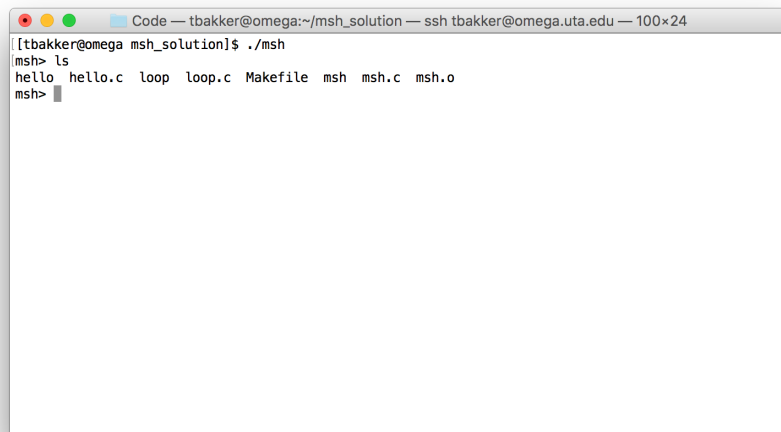
Due: Monday October 5th, 2020 5:30PM CDT

## Description

In this assignment you will write your own shell program, Mav shell (msh), similar to bourne shell (bash), c-shell (csh), or korn shell (ksh). It will accept commands, fork a child process and execute those commands. The shell, like csh or bash, will run and accept commands until the user exits the shell. Your file must be named msh.c

## Functional Requirements

**Requirement 1:** Your program will print out a prompt of msh> when it is ready to accept input. It must read a line of input and, if the command given is a supported shell command, it shall execute the command and display the output of the command.



```
Code — tbakker@omega:~/msh_solution — ssh tbakker@omega.uta.edu — 100x24
[tbakker@omega msh_solution]$ ./msh
msh> ls
hello  hello.c  loop  loop.c  Makefile  msh  msh.c  msh.o
msh> 
```

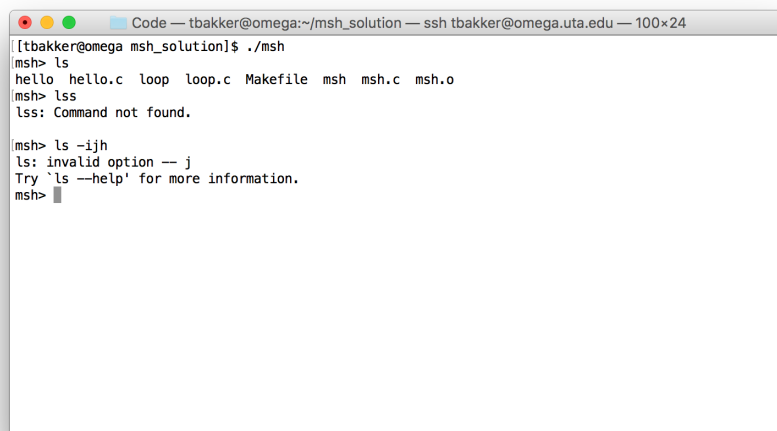
**Requirement 2:** If the command is not supported your shell shall print the invalid command followed by “: Command not found.”



```
Code — tbakker@omega:~/msh_solution — ssh tbakker@omega.uta.edu — 100x24
[[tbakker@omega msh_solution]$ ./msh
msh> ls
hello hello.c loop loop.c Makefile msh msh.c msh.o
msh> lss
lss: Command not found.

msh> 
```

**Requirement 3:** If the command option is an invalid option then your shell shall print the command followed by “: invalid option --” and the option that was invalid as well as a prompt to try —help. **exec()** outputs this automatically make sure you pass it on to your user. You have no work for this requirement. It’s free.



```
Code — tbakker@omega:~/msh_solution — ssh tbakker@omega.uta.edu — 100x24
[[tbakker@omega msh_solution]$ ./msh
msh> ls
hello hello.c loop loop.c Makefile msh msh.c msh.o
msh> lss
lss: Command not found.

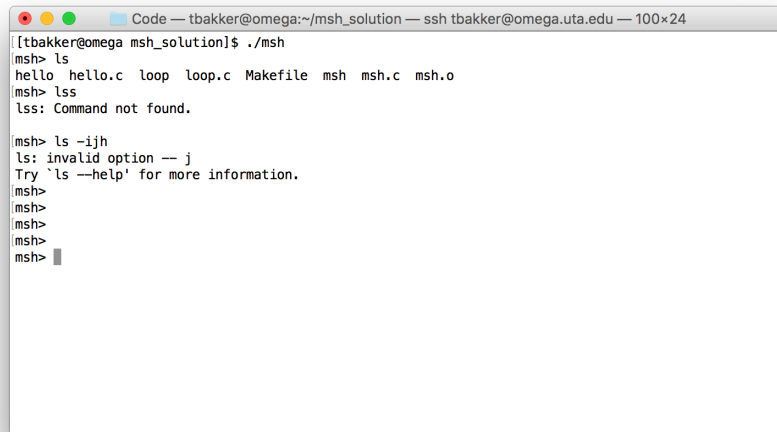
msh> ls -ijh
ls: invalid option -- j
Try 'ls --help' for more information.

msh> 
```

**Requirement 4:** After each command completes, your program shall print the `msh>` prompt and accept another line of input.

**Requirement 5:** Your shell will exit with status zero if the command is “quit” or “exit”.

**Requirement 6:** If the user types a blank line, your shell will, quietly and with no other output, print another prompt and accept a new line of input.



```
Code — tbakker@omega:~/msh_solution — ssh tbakker@omega.uta.edu — 100x24
[tbakker@omega msh_solution]$ ./msh
msh> ls
hello hello.c loop loop.c Makefile msh msh.c msh.o
msh> lss
lss: Command not found.

msh> ls -ijh
ls: invalid option -- j
Try 'ls --help' for more information.
msh>
msh>
msh>
msh>
```

**Requirement 7:** Your version of Mav shell shall support up to 10 command line parameters in addition to the command.

**Requirement 8:** Your shell shall support and execute any command entered. Any command in `/bin`, `/usr/bin/`, `/usr/local/bin/` and the current working directory is to be considered valid for testing.

Your shell shall search in the following PATH order:

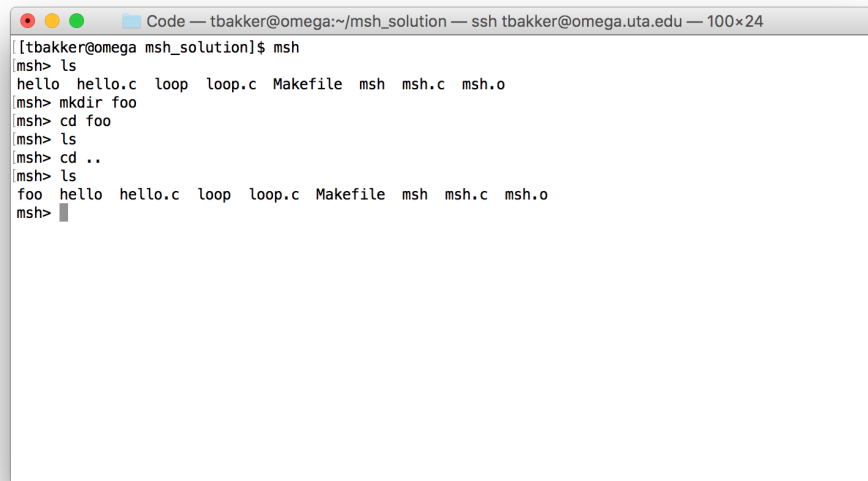
1. Current working directory,
2. `/usr/local/bin`
3. `/usr/bin`
4. `/bin`

Parameters may also be combined. For example, `ps` may be executed as: `ps -aef` or `ps -a -e -f`

**Requirement 9:** Mav shell shall be implemented using `fork()`, `wait()` and one of the `exec` family of functions.

Your Mav shell shall not use `system()`. Use of `system()` will result in a grade of 0.

**Requirement 10:** Your shell shall support the `cd` command to change directories. Your shell must handle `cd ..`.

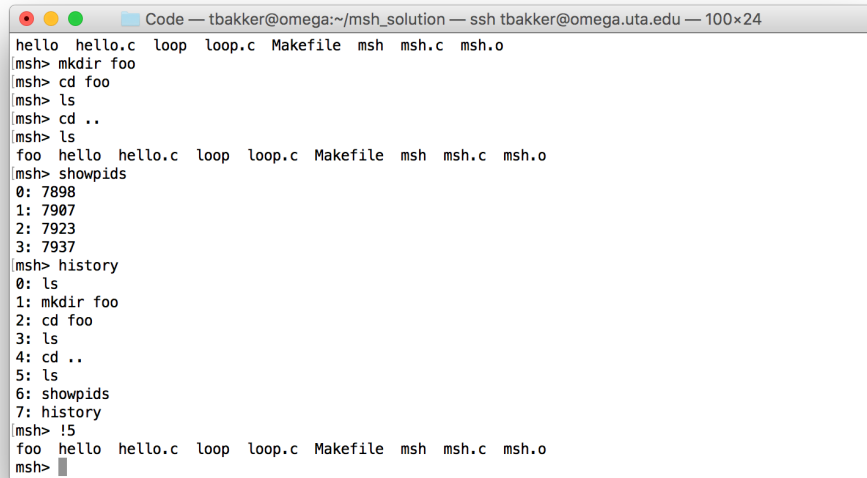


```
[tbakker@omega msh_solution]$ msh
msh> ls
hello  hello.c  loop  loop.c  Makefile  msh  msh.c  msh.o
msh> mkdir foo
msh> cd foo
msh> ls
msh> cd ..
msh> ls
foo  hello  hello.c  loop  loop.c  Makefile  msh  msh.c  msh.o
msh>
```

**Requirement 11:** Your shell shall support the `showpids` command to list the PIDs of the last 15 processes spawned by your shell. If there have been less than 15 processes spawned then it shall print only those process PIDs

**Requirement 12:** Your shell shall support the `history` command which will list the last 15 commands entered by the user. Typing `!n`, where `n` is a number between 1 and 15 will result in your shell re-running the `nth` command. If the `nth` command does not exist then your shell will state "Command not in history.". The output shall be a list of numbers 1 through `n` and their commands, each on a separate line, single spaced.

If there are less than 15 commands in the history only list the commands the user has entered up to that point.



```
Code — tbakker@omega:~/msh_solution — ssh tbakker@omega.uta.edu — 100x24
hello hello.c loop loop.c Makefile msh msh.c msh.o
[msh> mkdir foo
[msh> cd foo
[msh> ls
[msh> cd ..
[msh> ls
foo hello hello.c loop loop.c Makefile msh msh.c msh.o
[msh> showpids
0: 7898
1: 7907
2: 7923
3: 7937
[msh> history
0: ls
1: mkdir foo
2: cd foo
3: ls
4: cd ..
5: ls
6: showpids
7: history
[msh> !5
foo hello hello.c loop loop.c Makefile msh msh.c msh.o
[msh>
```

**Requirement 13:** [This requirement is removed.]

**Requirement 14:** Tabs or spaces shall be used to indent the code. Your code must use one or the other. All indentation must be consistent.

**Requirement 15:** No line of code shall exceed 100 characters.

**Requirement 16:** Each source code file shall have the following header filled out:

```
/*
Name: Student Name
ID: 10000001
*/
```

**Requirement 17:** All code must be well commented. This means descriptive comments that tell the intent of the code, not just what the code is executing.

The following are poor comments.

```
// Set working_str equal to strdup return
char *working_str = strdup( cmd_str );

// Set working_root equal to working_str
char *working_root = working_str;
```

The following explains the intent:

```
// Save a copy of the command line since strsep
// will end up moving the pointer head
char *working_str = strdup( cmd_str );

// we are going to move the working_str pointer so
// keep track of its original value so we can deallocate
// the correct amount at the end
char *working_root = working_str;
```

When in doubt over comment your code.

**Requirement 18:** Keep your curly brace placement consistent. If you place curly braces on a new line, always place curly braces on a new end. Don't mix end line brace placement with new line brace placement.

**Requirement 19:** Each function should have a header that describes its name, any parameters expected, any return values, as well as a description of what the function does. For example

**Requirement 20:** Remove all extraneous debug output before submission. The only output shall be the output of the commands entered or the shell prompt.

## Administrative

This assignment must be coded in C. Any other language will result in 0 points. Your programs will be compiled and graded on a CSE 3320 VM. Please make sure they compile and run on the VM before submitting them. Code that does not compile on the VM with:

```
gcc -Wall msh.c -o msh -std=c99
```

will result in a 0.

Your program, `msh.c` is to be turned in via Canvas. Submission time is determined by the Canvas system time. You may submit your programs as often as you wish. Only your last submission will be graded.

There are coding resources and working code you may use on Canvas and in the course github repository at: <https://github.com/CSE3320/Shell-Assignment> . You are free to use any of that code in your program if needed. You may use no other outside code.

## Academic Integrity

This assignment must be 100% your own work. No code may be copied from friends, previous students, books, web pages, etc. All code submitted is automatically checked against a database of previous semester's graded assignments, current student's code and common web sources. By submitting your code on Canvas you are attesting that you have neither given nor received unauthorized assistance on this work. **Code that is copied from an external source or used as inspiration, excluding the course github or Canvas, will result in a 0 for the assignment and referral to the Office of Student Conduct.**

## Hints

Read the man pages for the following: `fork`, `exec`, `exit`, `print`, `fgets`, `strtok`, `strsep`, `strcmp`, `wait`, and `pause`.

Use `fork` and one of the `exec` family as discussed in class to execute the command and call `wait` to wait for the child to complete. If the command is “`cd`” then use `chdir()` instead of `exec`. Note, `chdir()` must be called from the parent.

If you see garbage in any of your commands or parameters, try using the functions `memset()` or `bzero()` to clear out your input string and token array before and/or after you are done using them. Also, verify you are NULL terminating your strings.

There are examples on the course GitHub repository for this assignment that show how to use `execl` and `execvp`.