Shell in C: Part 3

Goal

Deployment: Make our shell usable by other people and ourselves.

Technical Goals

Learn about…

- Configuration of programs
- Documentation
- Logging
- Deployment of tools

Introduction

This week we will focus on adding additional features common to shells, and adding features you would do just before deploying.

Must have pre-deployment features

Configurability / Documentation
Adding configurability and documentation, while nice for a user, usually makes a developers job more difficult due to coupling within your system. Slight changes within your code can require changes within many locations. This is why we wait until right before deploying to implement those features (Apologies if you plan to continue working on your shell).

Configuration is typically done through argument variables, config files, and sometimes environment variables. See more at wiki (config files) or Shell Part 1

(From wikipedia) Software documentation is written text or illustration that accompanies computer software or is embedded in the source code. The documentation either explains how the software operates or how to use it, and may mean different things to people in different roles.

Documentation solutions depend on the medium. For shell programs, we usually create manpages, while for network API's we might use Swagger docs. At the end of the day, documentation is a nicely formatted document

Logs
(From wikipedia) Logging is the act of keeping a log of events that occur in a computer system, such as problems, errors or just information on current operations. These events may occur in

the operating system or in other software. A message or log entry is recorded for each such event. These log messages can then be used to monitor and understand the operation of the system, to debug problems.

In the simplest case, messages are written to a file, although depending on the use case, additional features may be necessary.

Logs are an essential part of any program, for the user to help debug when things go wrong. In fact, one of the biggest skills when debugging is being able to read logs. **Pro Tip**: When joining a company, find out where the logs are.

Logging is a part of system observability, an important idea we will touch on later.

Check out the Syslog specification for a popular logging standard.

Example Logs

System Log:

```
2024-03-04 12:15:32 [INFO] Application started successfully.
2024-03-04 12:20:45 [WARNING] Disk space is running low (10% remaining).
2024-03-04 12:30:10 [ERROR] Database connection failed. Retrying...
2024-03-04 12:32:05 [ERROR] Database connection failed. Retrying...
2024-03-04 12:35:20 [ERROR] Unable to establish connection with database.
Terminating application.
```

Security Log:

```
2024-03-04 14:20:35 [WARNING] Multiple failed login attempts detected from
IP address 203.0.113.12.
2024-03-04 14:21:10 [WARNING] Account 'user123' has been locked due to
suspicious activity.
2024-03-04 14:25:45 [INFO] Successful login from IP address 203.0.113.15.
```

Program logs are usually written to 2 locations depending on the type of program.

1. If the program is a service/daemon with running as root or in the syslog group, log to /var/log
2. Otherwise, log to a file in the user's home directory

Deployment

Deliverable Summary

Deployment Features

- logging
- configurability

Shell in C: Part 3
Written by Archer Heffern

- manual pages
- Deploying: Adding to PATH and/or make users default shell
- Deploying: Making globally accessible (Optional - Not recommended)

Logging

Implement logging, a log should be written to a file called “tamid_sh.log” located at “$HOME/.tamidsh.log”. /var/log is where many program logs are located, so I would suggest poking around there. The format of the logs can be whatever you want, although a timestamp and the command executed are a good place to start. Be pragmatic, log what you think is useful.

(Optional) Create a builtin or custom command which displays logs (eg. bash’s “history” builtin), bonus points for adding parsing (eg. Filter by time range or log level)

Configurability / Startup Scripts

Config File Locations

/etc/tamidsh
$HOME/.tamidsh_rc

This should be in increasing precedence. if an option is specified in /etc/tamidsh and in $HOME/.tamidsh_rc, the $HOME/.tamidsh_rc option should override the /etc/tamidsh option.

$HOME is the environment variable representing the home directory of a user. Eg. /Users/archerheffern
You can access this using the `getenv(3)` function as following:

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

int main() {
        printf("%s\n", getenv("HOME"));
}
```

Config File Format

You can format your config files however you want, just make sure you have a commenting system (eg. config file parser will ignore lines starting with #),

I suggest keeping it simple and doing basic key value pairs as such:

```
Option1=value
Option2=value
```

```
# This is a comment
Option3=value
```

Hint: You will probably want a function that opens a file, reads it line by line, updating a Config struct each time. If there is a syntax error it should notify the user.

Options

Create at least 3 config file options, you have full creative freedom. Just make sure to list everything in the config file documentation (Explained below)

Here are some ideas:

- Custom prompt: User can make prompt something like [Hello] instead of >
- Set environment variables
- Importing other files as config files - Config parser will parse those files next
- Specify commands to run before running the shell

Documentation

Create 2 manpage documentations, one for your Shell and one for the Shell config file, its syntax, and options.

Follow this guide. As with anything, don't read this start to end (skimming section titles is fine). Know what you want, and extrapolate the information you actually need. Submit the groff file with your deliverable. Since this is a small project, the documentation will not be that large - do what is necessary, nothing more and nothing less.

See man-pages(7) for man page conventions and pipe(2) as a reference of what a good man page looks like.

Deployment: Your Machine

- Move documentation to /usr/share/man/man1 and /usr/share/man/man5 respectively
    - Ensure it works by running man <command name>, you may need to reload the mandb database using mandb(8)
- Make executable anywhere on your machine
    - Move the binary to a folder specified in path - /usr/local/bin is conventional, that way it is accessible anywhere on your machine.
    - OR add to path the current location of your binary
    - OR create an symlink/hard link in a folder specified in path to your binary (`ln` command)
- (Optional - Strongly not recommended): Make your shell the default shell for your user.

- On Linux the default shell for each user is determined by the /etc/passwd configuration file. See passwd(5) for more info. You can either edit this file directly to make your own shell the default, or be responsible and use the `chsh` command to more safely edit this file for you

(Optional) Deployment: Making your program globally accessible

I would **not** suggest doing any of these, but for completeness sake, it's good to be aware of this section.

The idea of deployment is, how do we deliver our software to users. Users are going to have to install it, so there are a few options:

1. Share it via hard drive
2. Transmit via internet
    a. Personal website
    b. Code sharing websites (github, gitlab, etc)
    c. Package manager

Code sharing website:

- You upload your code and other people can download it
- eg: github, personal sites, non website servers (this one uses FTP)

Package managers:
Package managers are programs which make it super easy to install software online right from the terminal! They tend to have a config file with a list of URLs to query for your package (see the man page FILES section), and will install if there is a name match. To make your program accessible, either create your own server (abiding by the communication specification of the package manager so the package manager and your site can properly interact) or upload to someone else's server.
eg.

brew (brew install <package>) # MacOS only
apt (apt install <package>) # Debian based Linux Distributions
git (git clone <url>) # Not a package manager, but you can still install software from online

Continuing with the project

Should you want to continue with this project, you can!

If you are interested in adding other features to your shell, check out documentation of shells like bash(3) or zsh(3), or think of something fun you would like to implement!

Shell in C: Part 3
Written by Archer Heffern

The greatest opportunity you have is to **learn how to make a programming language**. Modern day shells have “Scripting language”, to make it easier to execute many programs in a structured manner.

A great book to get started is Crafting Interpreters. It has a pragmatic and hands on approach, and teaches everything you need for a modern language.