Bash Scripting Primer

CS14 - S26

Scripting

Generally, a “script” is a short, custom program meant to complete a single repetitive task.

As opposed to more complex programs, scripts are typically written in a single file and are minimally interactive.

Some problems you could typically solve with scripts are:

- Copying sets of files from one directory to another and configuring their permissions.
- Compiling and installing programs in specific locations.
- Converting one data format to another

Bash Programming Language

Bash is a full, scripting oriented programming language.

Throughout the semester, we have been using the bash shell, which interprets Bash language commands.

There exist many alternative shells with their own languages. E.g., sh, zsh, fish

bash is the default shell of many Linux distros. (zsh for Mac)

Bash scripts are great for operating at the granularity of files and directories and for allowing you to incorporate command line utilities easily.

However, Bash is infamous for its confusing and bug prone syntax.

Making a bash script

Bash script files conventionally end with the `.sh` extension.

To make sure a script is interpreted as bash code, you should add a “shebang” directive to the top, typically:

```
#!/bin/bash
```

Bash scripts need to be given the executable permission (with `chmod`) in order to run them using `./`

* You can also make python programs runnable with `./` by adding a shebang `#!/bin/python3` and giving the program the executable permission!

Bash “Hello World”

```
#!/bin/bash


echo “Hello world!”
```

Command Line Arguments

Any space-separated tokens* that come after a bash script on the command line are interpreted as command line arguments.

These can be accessed in the script using a `$` followed by the position of the argument.

Similar to other languages, `$0` is the name of the script, `$1` is the first argument to the script, and so on.

* If you put single or double quotes around tokens separated with a space, it will be stored in a single command line argument. This is one of the many edge cases that makes bash scripting error prone.

Bash “Hello user”

```
#!/bin/bash


echo “Hello $1!”
```