05 – Wildcards, loops, and variables

CS 2043: Unix Tools and Scripting, Spring 2019 [1]

Matthew Milano

February 4, 2019

Cornell University

Table of Contents

Chaining Commands

Your Environment and Variables

- There are various *environment* variables defined for your shell.
- They are almost always all capital letters.
- You obtain their value by dereferencing them with a `$`.

```
$ echo $PWD    # present working directory
$ echo $OLDPWD # print previous working directory
$ printenv     # print all environment variables
```

- There are also *local* variables you can use / set.
- Primary difference:
  - *Environment* variables are available in your shell, *and* in scripts.
  - *Local* variables are *only* available in your shell.
    - "Shell" here just means "current terminal session."

What is Defined?

- The environment:
  - `env`: displays all environment variables.
  - `unsetenv <var_name>`: remove an environment variable.
  - Create an environment variable*:
    1. `env ENV_VAR_NAME="value"`
    2. `export ENV_VAR_NAME="value"`
  - `export` is the most common. Exceptional explanation here.
- The local variables:
  - `set`: displays all shell / local variables.
  - `unset <var_name>`: remove a local shell variable.
  - Create a local variable*:
    1. `set local_var="value"`
    2. `local_var="value"`

* These only last for the current shell session; we will learn how to make them “permanent” soon.

Brief Example: Environment Variable Manipulation

```
# MY_ENV_VAR is not set yet, so nothing prints
$ echo "My env var is: $MY_ENV_VAR"
My env var is:

# Set the environment variable (can also use `export` in bash)
$ env MY_ENV_VAR="Lemming King"

# Now that we have set it, print it
$ echo "My env var is: $MY_ENV_VAR"
My env var is: Lemming King

# "Delete" with `unsetenv`. Print again, confirming it's gone
# Emphasis: there *is* an `env` after `unset`
$ unsetenv MY_ENV_VAR
$ echo "My env var is: $MY_ENV_VAR"
My env var is:
```

Brief Example: Local Variable Manipulation

```
# my_local_var is not set yet, so nothing prints
$ echo "My local var is: $my_local_var"
My local var is:

# Just declare it (can also use the `set` command)
$ my_local_var="King of the Lemmings"

# Now that we have set it, print it
$ echo "My local var is: $my_local_var"
My local var is: King of the Lemmings

# "Delete" with `unset`. Print again, confirming it's gone
# Emphasis: there is *not* an `env` after `unset`
$ unset my_local_var
$ echo "My local var is: $my_local_var"
My local var is:
```

Exit Codes

- When you execute commands, they have an “exit code”.
  - This how you “signal” to others in the shell: through exit codes.
- The exit code of the *last command executed* is stored in `$?`
- There are various exit codes, here are a few examples:

```
$ super_awesome_command
bash: super_awesome_command: command not found...
$ echo $?
127
$ echo "What is the exit code we want?"
What is the exit code we want?
$ echo $?
0
```

- The success code we want is actually `0`. Refer to [2].
- Remember `cat` with no args? You will have to `ctrl+c` to kill it, what would the exit code be?

Executing Multiple Commands in a Row

- With exit codes, we can define some simple rules to chain commands together:
- Always execute:

```
$ cmd1; cmd2   # exec cmd1 first, then cmd2
```

- Execute conditioned upon exit code of `cmd1`:

```
$ cmd1 && cmd2 # exec cmd2 only if cmd1 returned 0
$ cmd1 || cmd2 # exec cmd2 only if cmd1 returned NOT 0
```

    - Kind of backwards, in terms of what means continue for *and*, but that was likely easier to implement since there is only one `0` and many *not* `0`'s.

References

[1] Stephen McDowell, Bruno Abrahao, Hussam Abu-Libdeh, Nicolas Savva, David Slater, and others over the years. “Previous Cornell CS 2043 Course Slides”.

[2] The Linux Documentation Project. *Exit Codes with Special Meanings*. 2017. URL: http://tldp.org/LDP/abs/html/exitcodes.html.