Software Development on Linux Systems

4002-XXX-XX

By

Cody Van De Mark

Today

- Command Line
- Scripting
- Configuration Files

Command Line

- For many tasks, the command line is the fastest and most efficient way to completion
- The command line allows you to string commands together quickly and script tasks for automatic completion
- The Linux command line is incredibly powerful and offers support for most applications and commands
- This makes it unbelievably powerful and versatile for programming and scripting

Command Line

- The Linux command line (terminal) offers the Bash environment which supports thousands of powerful commands

- Terminal also offers a stack trace system for any language and invocation of almost all applications with any supported flags

- Through the use of Linux's cron system, you can have terminal automate tasks at set times;

  With the use of xdotool, you can even automate mouse and keyboard input

Bash

- Bash is the default terminal environment on Ubuntu and Fedora ;

  There are others installed you that may switch to on the fly

- Bash is the Unix-like system you have probably seen before with commands like:
  - ls
  - chmod
  - echo
  - ping
  - cd

Bash

- Bash also has support for a plethora of administrative commands

  Examples:

  - adduser - adds a user to the system
  - chgrp - changes the group a file is associated with
  - chown - changes the owner of a file
  - fakeroot - run commands as a fake root user with no permissions
  - ln - creates links to files, either shortcuts or duplicates
  - passwd - allows you to change the password of a specified user
  - tar - allows you to compress and uncompress files
  - uname - gives operating system details such as kernel version, etc
  - whoami - lets you know who the current process is running as

Bash

- Bash also supports variables and logical statements, such as if statements and loops

- Variables are referenced with a dollar sign and created with an equals sign with no spaces

```
myVariable=hello;
echo $myVariable;
```

- Variables only run within a single terminal session when set, as if they were local variables, but they can be exported as global variables with export

Example: export myVariable

Bash

- Bash if statements support a number of standard operators
  - -eq - equal to *#used with numbers*
  - -ne - not equal to *#used with numbers*
  - -lt - less than *#used with numbers*
  - -le - less than or equal to *#used with numbers*
  - -gt - greater than *#used with numbers*
  - -ge - greater than or equal to *#used with numbers*
  - = - equal to *#used with strings*
  - != - not equal to *#used with strings*
  - -z - empty string *#used with strings*
  - -n - not empty string *#used with strings*

Bash

- If statements are done with if, then, elif, else and fi

example:

```
myNum1=10;

if [ $myNum1 -eq 10 ]
then
        echo $myNum1;

elif [ $myNum1 -gt 10 ]
then
        echo “Greater than 10”;

else
        echo “Number was less than 10”;
fi
```

Bash

- While loops are done with while, do and done

Example:

```
counter=0;

while [ $counter -lt 100 ];
do
        echo $counter;
        let counter=counter+1;
done
```

Bash

- Until loops are very similar to while loops, but stop when the operator has been reached

Example:

```
counter=0;

until [ $counter -eq 10 ];
do
        echo $counter;
        let counter=counter+1;
done
```

Bash

- Arrays are spaced delimited in Bash, but can be used like arrays in most languages;

  Arrays can also be declared add as associative in Bash

  Example:

```
array1=(“one” “two” “three” “four”);

declare  -A  array2;
array2[“person1”]=“Nick”;
array2[“person2”]=“Samantha”;
array3[“person3”]=“Robert”;
```

Bash

- For loops in Bash are similar to loops in most languages, but can use expanding characters { } to get all values out of an array

  Example:

```
array1=(“one” “two” “three” “four”);

for  i  in “${array1[@]}”
do
      echo $i;
done
```

Configuration Files

- Everything in Linux is a file. Period
- These files might be binaries, executables, scripts, configuration files or even hardware (IE: the cd drive and ram are both files)
- Configuration files are files designed to control other files by defining run parameters, variables and constants
- Configuration files can take on almost any form and are defined by the developers of the project that uses it

Configuration Files

- Most configuration files will have the .ini or .conf extension, but many do not have an extension at all

- The extension of the file and the location of the file are defined by the developers of the project that use it

- Because of this configuration files are application specific and are placed in many areas around the system;

  Commons places are in the user's home directory if they are user specific or in the application's folder if settings are system specific

Configuration Files

- Here are some examples of configuration files

redshift.conf

```
[redshift]
location=provider=manual
temp-day=5500
temp-night=5500
adjustment-method=vidmode

[manual]
lat=43.12
lon=77.76
```

preferences.conf

```
<group
      id="ui"
      language="english"  >
</group>

<group>
      id="screen"
      size="fullscreen"
      borders="on"  >
</group>
```

Configuration Files

- As you can see, configuration files can be vastly different
- Most applications have a configuration file specific to their needs and language
- Even the processor itself has configuration files that control the maximum number of threads, clock speed, core id, power management, etc
- You will need to make your own configuration file(s) for the project