Lecture 6

Variables

Goals of this lecture:

- UNIX and shell variables
- Environment variables
- Using and setting variables
- Dot files

Unix Variables

Variables in Unix are placeholders for particular values needed by the shell and programs that run within the shell.Variables are a way of passing information from the shell to programs when you run them. Programs look for particular variables and if they are found will use the values stored. These variables are stored in the environment, as described in Lecture 1, and are set by the user, by system, by the shell, or by any program that loads another program.

When a shell is running, three main types of variables are present:

- **Local Variables:** A local variable is a variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at command prompt.
- **Environment Variables:** An environment variable is a variable that is available to any child process of the shell. Some programs need environment variables in order to function correctly. Usually a shell script defines only those environment variables that are needed by the programs that it runs.
- **Shell Variables:** A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables.

Examples

An example of an environment variable is the OSTYPE variable. The value of this is the current operating system you are using.

To see the type of operating system you are running:

```
$ echo $OSTYPE
darwin13
```

Other common environment variables include:

Variable	Meaning
USER	Your login name
HOME	The path of your home directory
HOST	The name of the computer you are using
ARCH	The architecture of the computers processor
DISPLAY	The name of the computer screen to display X windows
PRINTER	The default printer to send print jobs
PATH	The directories the shell should search to find a command

---

Querying Variables

To see the values of currently set variables you can use one of three methods.

printenv - print all or part of the environment

```
$ printenv
TERM_PROGRAM=Apple_Terminal
SHELL=/bin/bash
TERM=xterm-256color
CLICOLOR=true
TMPDIR=/var/folders/f2/3zczb3hx2k321jzm_nrknht80000gp/T/
Apple_PubSub_Socket_Render=/tmp/launch-L9pgwr/Render
TERM_PROGRAM_VERSION=326
TERM_SESSION_ID=87704C3A-8677-4B3B-84C4-6ECFC34FD2AE
USER=smith
SSH_AUTH_SOCK=/tmp/launch-SMOjAu/Listeners
__CF_USER_TEXT_ENCODING=0x1F6:0:0
LSCOLORS=gxdxxxxxbxxxxxbxbxgxgx
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/texbin
__CHECKFIX1436934=1
PWD=/Users/smith
EDITOR=emacs
LANG=en_US.UTF-8
SHLVL=1
HOME=/Users/smith
LOGNAME=smith
SECURITYSESSIONID=186a4
_=/usr/bin/printenv
```

env - print all exported environment

```
$ env
TERM_PROGRAM=Apple_Terminal
SHELL=/bin/bash
TERM=xterm-256color
CLICOLOR=true
TMPDIR=/var/folders/f2/3zczb3hx2k321jzm_nrknht80000gp/T/
Apple_PubSub_Socket_Render=/tmp/launch-L9pgwr/Render
TERM_PROGRAM_VERSION=326
TERM_SESSION_ID=87704C3A-8677-4B3B-84C4-6ECFC34FD2AE
USER=smith
SSH_AUTH_SOCK=/tmp/launch-SMOjAu/Listeners
__CF_USER_TEXT_ENCODING=0x1F6:0:0
LSCOLORS=gxdxxxxxbxxxxxbxbxgxgx
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/texbin
__CHECKFIX1436934=1
PWD=/Users/smith
EDITOR=emacs
LANG=en_US.UTF-8
SHLVL=1
HOME=/Users/smith
LOGNAME=smith
SECURITYSESSIONID=186a4
_=/usr/bin/printenv
```

set - print the name and value of each shell variable

```
$ set
```

```
Apple_PubSub_Socket_Render=/tmp/launch-L9pgwr/Render
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="3" [1]="2" [2]="51" [3]="1" [4]="release" [5]="x86_64-apple-darwin13")
BASH_VERSION='3.2.51(1)-release'
CLICOLOR=true
COLUMNS=80
DIRSTACK=()
EDITOR=emacs
EUID=502
GROUPS=()
HISTFILE=/Users/smith/.bash_history
HISTFILESIZE=500
HISTSIZE=500
HOME=/Users/smith
HOSTNAME=machine.uoregon.edu
HOSTTYPE=x86_64
IFS=$' \t\n'
LANG=en_US.UTF-8
LINES=24
LOGNAME=smith
LSCOLORS=gxdxxxxxbxxxxxbxbxgxgx
MACHTYPE=x86_64-apple-darwin13
MAILCHECK=60
OPTERR=1
OPTIND=1
OSTYPE=darwin13
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/texbin
PIPESTATUS=([0]="0")
PPID=91909
PROMPT_COMMAND='update_terminal_cwd; '
PS1='\h:\W \u\$ '
PS2='> '
PS4='+ '
PWD=/Users/smith
SECURITYSESSIONID=186a4
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
SHLVL=1
SSH_AUTH_SOCK=/tmp/launch-SMOjAu/Listeners
TERM=xterm-256color
TERM_PROGRAM=Apple_Terminal
TERM_PROGRAM_VERSION=326
TERM_SESSION_ID=87704C3A-8677-4B3B-84C4-6ECFC34FD2AE
TMPDIR=/var/folders/f2/3zczb3hx2k321jzm_nrknht80000gp/T/
UID=502
USER=smith
_=clear
__CF_USER_TEXT_ENCODING=0x1F6:0:0
__CHECKFIX1436934=1
```

Setting Variables

Setting variables depends on the type of shell you are using. Use the echo command to determine your shell type.

```
$ echo $SHELL
/bin/bash
```

Most unix installations use a Bourne-style shell (aka bash, ksh, etc..). To set a variable in this type of shell, you can simply assign it a value:

```
$ FOO = "bar"
```

However, this will only set this variable for this current session, and not set this variable for any child processes.

To set a variable for the current session and any child processes, you must export it.

```
$ FOO = "bar"
$ export FOO
```

or

```
$ export FOO=$FOO:"bar"
```

If you are using the C Shell or its variants (tcsh, etc), you will need to use the `setenv` command:

```
$ setenv FOO=$FOO:"bar"
```

This will set the variable for this and all children processes.

Dot Files

Every time a user logins to a unix session, the system looks for files that describe the initialization of variables. These files are calle *dot* files and their types refers to the fact that they have a dot (**.**) before their filename. The dot preceding the file name means they are hidden in a normal listing of the directory (i.e. `ls`), however the -a flag will show these hidden files (`ls -a`). In these dot files a user can define the variable settings so they are set every time a user logs in. The dot files will be located directly under the ${HOME} (or `~`) directory.

Depending on the type of shell you are using, the name of the dot files will change.
Some of the most common dot files include:

- ~/.profile (mac and most unix distributions)
- ~/.bashrc or .basch_profile (bash shell)
- ~/.tcshrc (tcsh shell)
- ~/.kshrc (ksh shell)
- ~/.zprofile (zsh shell)

This list is not exhaustive, and you may find your dot files have different names.

An example .profile file looks like this:

```
$ more .profile
export CLICOLOR="true"
export LSCOLORS="gxdxxxxxbxxxxxbxbxgxgx"

alias emacs=/Applications/Aquamacs.app/Contents/MacOS/Aquamacs
export EDITOR=emacs

alias showFiles='defaults write com.apple.finder AppleShowAllFiles YES;
killall Finder /System/Library/CoreServices/Finder.app'

alias hideFiles='defaults write com.apple.finder AppleShowAllFiles NO; killall Finder
/System/Library/CoreServices/Finder.app'
```

The first lines set the CLICOLOR and LSCOLORS variables which define how the terminal displays listings of files (specifically they define the colors to be used).

The `alias` command tells the shell to substitute one string with another when executing a command. In the example above, the file is aliasing the emacs command to `/Applications/Aquamacs.app/Contents/MacOS/Aquamacs`, which is the user's preferred version of emacs. The user then exports the EDITOR variable, setting the default editor to a specific version of emacs for all sessions.

The final lines of the .profile file alias commands for the GUI finder on a Mac to show or hide the dot files. Once these lines are read by the system upon login, the user can use the `showFiles` command to display all the hidden files, and the `hideFiles` to hide them. This is a demonstration of how the user can customize commands in the dot files.

Setting the Path

The `PATH` variable defines which directories the shell will look in to find a requested command. That is, when a user types a command, the shell will look at each directory, in order, to find the location of the executable command. If the system returns a message saying "command: Command not found", this indicates that either the command doesn't exist at all on the system or it is not in the user's path. Thus, if you know you have a command on your system (for example, you have just installed a program), but the shell cannot find the command, the solution is to append the directory of the command to the `PATH` variable.

To see the path, use the `echo` command.

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/texbin
```

In the above example, the shell will look for commands in the following directories (in this order):

1. /usr/local/bin
2. /usr/bin
3. /bin
4. /usr/sbin
5. /sbin
6. /usr/texbin

To append to the `PATH` variable, set the `PATH` variable as described above, but use the `:` symbol to append the new path.

```
$ PATH = $PATH:/additionalPath/
$ export PATH
```