# CS2042 - Unix Tools

## Lecture 3
## Making Bash Work For You
## Fall 2010

Hussam Abu-Libdeh

based on slides by David Slater

September 13, 2010

# A little homework

- Homework 1 out now
- Due on Thursday at 11:59PM
- Moving around and GNU file tools

# Shell's Again

Many shells for UNIX-like systems:

- `sh`: The Bourne Shell -
  a popular shell made by Stephen Bourne

- `bash`: The Bourne Again Shell -
  default shell for the GNU OS, most Linux distros, and OSX

- `csh`: The C Shell -
  interactive and close to C
  default shell for BSD-based systems

- `zsh`: The Z Shell -
  possibly the most fully-featured shell inspired by sh, bash, ksh, and tcsh

# Shell's Again

- Since `bash` is the gold standard of shells and has more than enough features for this course, we'll stick with it.
- For more info, use Wikipedia as a starting point: http://en.wikipedia.org/wiki/Comparison_of_command_shells

# But the CSUG machines do not default to Bash :(

The CSUG machines automatically put us into csh not bash.

- If you are already logged in to the server, just type `bash`
- More importantly we would like the server to automatically put us into bash when we login. One way to do this is by editing the file `/.login` which gets executed each time you log in to the server and csh starts up.

**Start bash automatically**

Add the following line to the end of `/.login`

```
if ( -f /bin/bash) exec /bin/bash --login
```

If you had root privileges you could just edit /etc/passwd and find the line corresponding to the current user.

# Variables!

- Bash scripting is very powerful! If you wanted to you could write a web server using Bash scripting.
- To get anything done we need variables. In Bash, all variables are preceded by a dollar sign ($).
- The contents of any variable can be listed using the `echo` command
- Two types of variables: Local and Environment.

**Example:**

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

# Environment Variables

- Environment Variables are used by the system to define aspects of operation.
- The Shell passes environment variables to its child processes
- Examples:
  - $Shell - which shell will be used by default
  - $PATH - a list of directories to search for binaries
  - $HOSTNAME - the hostname of the machine
  - $HOME - current user's home directory
- To get a list of all current environment variables type `env`

## New Environment Variable:

To set a new environment variable use `export`

```
hussam@rumman:~$ export X=3
hussam@rumman:~$ echo $X
3
```

Note: NO Spaces around the = sign.

# Local Variables

We can also define local variables, which exist only in the current shell:

**Example:**

```
hussam@rumman:~$ x=3
hussam@rumman:~$ echo $x
3
```

**Note:** There cannot be a space after the x nor before the 3!

# A Word About the Difference

The main difference between environment variables and local variables is environment variables are passed to child processes while local variables are not:

## Local Variable:

```
hussam@rumman:~$ x=3
hussam@rumman:~$ echo $x
3
hussam@rumman:~$ bash
hussam@rumman:~$ echo $x

hussam@rumman:~$
```

## Environment Variable:

```
hussam@rumman:~$ export x=myvalue
hussam@rumman:~$ echo $x
myvalue
hussam@rumman:~$ bash
hussam@rumman:~$ echo $x
myvalue

hussam@rumman:~$
```

# Environment Variables Again...

When we say the Shell passes environment variables to its child processes, we mean a copy is passed. If the variable is changed in the child process it is **not** changed for the parent

**Example:**

```
hussam@rumman:~$ export x=value1
hussam@rumman:~$ bash
hussam@rumman:~$ echo $x
value1
hussam@rumman:~$ export x=value2
hussam@rumman:~$ exit
hussam@rumman:~$ echo $x
value1
```

We will talk about why this is important once we have more programs at our disposal.

# Listing and Removing Variables

- `env` - displays all environment variables
- `set` - displays all shell/local variables
- `unset name` - remove a shell variable
- `unsetenv name` - remove an environment variable

# Using Bash Efficiently

Now lets talk about how bash makes life easier.

## Tab Completion

You can use the Tab key to auto-complete commands, parameters, and file and directory names. If there are multiple choices based on what you've typed so far, Bash will list them all!

# Shell Expansion

In a bash shell, if we type:

```
$ echo This is a test
This is a test
```

But if we type

```
$ echo *
Lec1.pdf Lec1.dvi Lec1.tex Lec1.aux
```

What happened?

# Shell Expansion

In a bash shell, if we type:

```
$ echo This is a test
This is a test
```

But if we type

```
$ echo *
Lec1.pdf Lec1.dvi Lec1.tex Lec1.aux
```

What happened?

The shell expanded * to all files in the current directory. This is an example of path expansion, one type of shell expansion.

# Interpreting Special Characters

The following are special characters:

$ * < > & ? { } [ ]

- The shell interprets them in a special way unless we escape (\$) or place them in quotes "$".
- When we first invoke a command, the shell first translates it from a string of characters to a UNIX command that it understands.
- A shell's ability to interpret and expand commands is one of the powers of shell scripting.

# Shell Expansions

The shell interprets $ in a special way.

- If var is a variable, then `$var` is the value stored in the variable var.
- If `cmd` is a command, then `$(cmd)` is translated to the result of the command `cmd`.

```
hussam@rumman:~$ echo $USER
hussam
hussam@rumman:~$ echo $(pwd)
/home/hussam
```

# Shell Expansions

`*` `^` `?` `{` `}` `[` `]` Are all "wildcard" characters that the shell uses to match:

- Any string
- A single character
- A phrase
- A restricted set of characters

# Shell Expansions

- `*` matches any string, including the null string (i.e. 0 or more characters).

**Examples**:

| Input | Matched | Not Matched |
|---|---|---|
| `Lec*` | `Lecture1.pdf Lec.avi` | `ALecBaldwin/` |
| `L*ure*` | `Lecture2.pdf Lectures/` | `sure.txt` |
| `*.tex` | `Lecture1.tex Presentation.tex` | `tex/` |

# Shell Expansions

- ? matches a single character

**Examples**:

| Input | Matched | Not Matched |
|---|---|---|
| `Lecture?.pdf` | `Lecture1.pdf Lecture2.pdf` | `Lecture11.pdf` |
| `ca?` | `cat can cap` | `ca cake` |

# Shell Expansions

- `[...]` matches any character inside the square brackets
  - Use a dash to indicate a range of characters
  - Can put commas between characters/ranges

**Examples**:

| Input | Matched | Not Matched |
|---|---|---|
| `[SL]ec*` | `Lecture Section` | `Vector.tex` |
| `Day[1-4].pdf` | `Day1.pdf Day2.pdf` | `Day5.pdf` |
| `[A-Z,a-z][0-9].mp3` | `A9.mp3 z4.mp3` | `Bz2.mp3 9a.mp3` |

# Shell Expansions

- `[^...]` matches any character **not** inside the square brackets

**Examples**:

| Input | Matched | Not Matched |
|---|---|---|
| `[^A-P]ec*` | `Section.pdf` | `Lecture.pdf` |
| `[^A-Za-z]*` | `90210 9Days.avi .bash_profile` | `vacation.jpg` |

# Shell Expansions

- **Brace Expansion:** $\{\ldots,\ldots\}$ matches any phrase inside the comma-separated brackets

**Examples**:

| Input | Matched |
|---|---|
| `{Hello,Goodbye}\ World` | `Hello World Goodbye World` |

**NOTE:** brace expansion must have a list of patterns to choose from (i.e. at least two options)

# Shell Expansions

And of course, we can use them together:

| Input | Matched | Not Matched |
|---|---|---|
| `*i[a-z]e*` | `gift_ideas profile.doc ice` | `DriVer.exe` |
| `[bf][ao][ro].mp?` | `bar.mp3 foo.mpg far.mp4` | `foo.mpeg` |

# Quoting

If we want the shell to not interpret special characters we can use quotes:

- Single Quotes ('): No special characters are evaluated
- Double Quotes ("): Variable and command substitution is performed
- Back Quotes (`): Execute the command within the quotes

## Example

```
hussam@rumman:~$ echo "$USER owes me $ 1.00"
hussam owes me $ 1.00
hussam@rumman:~$ echo '$USER owes me $ 1.00'
$USER owes me $ 1.00
hussam@rumman:~$ echo "I am $USER and today is
`date`"+
I am hussam and today is Wed Feb 11 16:23:30 EST 2009
```

# Arithmetic Expansion

The shell will expand arithmetic expressions that are encased in $(( expression ))

## Examples

```
hussam@rumman:~$ echo $((2+3))
5
hussam@rumman:~$ echo $((2 < 3))
1
hussam@rumman:~$ echo $((x++))
3
```

And many more.

**Note:** the post-increment by 1 operation (++) only works on variables

# Aliases

The more you use Bash the more you see what options you use all the time. For instance `ls -l` to see permissions, or `rm -i` to insure you don't accidentally delete a file. Wouldn't it be nice to be able to make shortcuts for these things?

## Alias:

`alias name=command`

- The alias allows you to rename or type something simple instead of typing a long command
- You can set an alias for your current session at the command prompt
- To set an alias more permanently add it to your `.bashrc` or `.bash_profile` file.

# Alias Examples

## Examples

```
alias ls='ls --color=auto'
alias dc=cd
alias ll='ls -l'
```

- Quotes are necessary if the string being aliased is more than one word
- To see what aliases are active simply type `alias`
- Note: If you are poking around in .bashrc you should know that any line that starts with # is commented out.

# Modifying your Prompt

The environment variable $PS1 stores your default prompt. You can modify this variable to spruce up your prompt if you like:

**Example**

First `echo $PS1` to see its current value
`\s-\v\$` (default)

It consists mostly of backslash-escaped special characters, like `\s` (name of shell) and `\v` (version of bash). There are a whole bunch of options, which can be found at

http://www.gnu.org/software/bash/manual/bashref.html#Printing-a-Prompt

# Modifying Your Prompt

Once you have a prompt you like, set your $PS1 variable

## Define your prompt

```
hussam@rumman:~$ export PS1="New Prompt String"
```

- Type this line at the command prompt to temporarily change your prompt (good for testing)
- Add this line to `~/.bashrc` or `~/.bash_profiles` to make the change permanent.

**Note:** Parentheses must be used to invoke the  characters.

## Examples

`PS1="\u \w \t_"` ⇒ `hussam ~ 12:12:12_`

`PS1="\W \j \d\:"` ⇒ `~ 0 Oct 02:`