Lecture 6: Using BASH Effectively

CS2042 - UNIX Tools

October 10, 2008

Lecture Outline

1 More About BASH
- Variables
- Making BASH Work for You
- Pattern Matching (Globbing)

2 Screen
- What Where Why?
- Handy Key Commands

What Else Is There?

There have been many shells created over the years for UNIX environments:

- **bash** - default shell for OSX and most Linux machines
- **csh** - default shell for BSD-based systems
- **zsh** - possibly the most fully-featured shell
- A frighteningly thorough comparison of the features of many shells can be found here.

Since **bash** is the gold standard of shells and has more than enough features for this course, we'll stick with it.

How do we use BASH?

The servers we use for this class will automatically put us into csh, not bash.

- If you are already logged in to the server, just type **bash**.

If you want the server to automatically put you into bash, you may want to add the following to ∼/**.login**.

Convert to bash from csh on start up

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

Note that ∼/**.login** gets executed each time you log in to the server and csh starts up. Conversely, ∼/**.cshrc** gets executed every time you enter the C-shell even if you were already logged in.

BASH and Variables

- BASH is a full-fledged programming language in addition to a handy shell. If you wanted to, you could write a web server using BASH scripting.
- To get anything done in a programming language, you need support for variables. Variables in BASH are preceded by a dollar sign ($).
- The contents of any variable can be listed using the **echo** command.

Example:

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

Environment Variables

Environment variables are generally used by the system to define aspects of operation. Most of these should not (or cannot) be changed by the user.

- $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
- ...and many others which don't concern us

Local Variables

While we don't get much mileage out of many of our system's environment variables, BASH also allows us to define our own.

Example:

```
x=3
echo $x
3
```

We can also use **export** to define variables.

Example:

```
export seven=7
echo $seven
7
```

Lecture Outline

1. More About BASH
   - Variables
   - Making BASH Work for You
   - Pattern Matching (Globbing)
2. Screen
   - What Where Why?
   - Handy Key Commands

Tab Completion

Did You Know?

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.

Try this at home!!

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 what its value is for now.

\s-\v\$ (default)

It consists mostly of backslash-escaped special characters, like \u. There are a whole bunch of options, all of which can be found online here.

Modifying Your Prompt, cont.

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

Define your prompt

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.

Some example BASH prompts

- **PS1="\u-\h \w\$"** → mjm458-csug06 ~\$
- **PS1="money\j\t "** → money014:23:57

Lecture Outline

1 More About BASH
- Variables
- Making BASH Work for You
- Pattern Matching (Globbing)

2 Screen
- What Where Why?
- Handy Key Commands

More Wildcards!

Earlier we mentioned how useful ”wild card” characters can be when looking for a particular file or trying to perform operations on a group of files. Let’s take a closer look at wildcards which can:

- Match any string
- Match a single character
- Match a single restricted character
- Match a restricted range of characters

The String

The * Wildcard

* - Matches any string, including the null string (an empty string, nothing)

Examples:

Input	Matched	Not Matched
lec*	lecture1.pdf, lecture2.doc, lectures/	election_data/
.mp	foo.mp3, bar.mpeg, .mplayer/	mp3s/, tmp/
mi*r	mirror, mir, minor, mine.rar	mi, mine

The Character

The ? Wildcard

? - Matches any single character (number, letter, punctuation!)

Examples:

Input	**Matched**	**Not Matched**
lecture?.pdf	lecture1.pdf, lecture2.pdf	lecture12.pdf
foo.mp?	foo.mp3, foo.mp4, foo.mpg	foo.mpeg,
min?	mine, mind, ming, mint, mink	minute, min

The Character Range

The [...] Wildcard

[ ] - Matches any one of a list of comma-separated characters. A dash between two characters indicates a range to be matched.

Examples:

Input	**Matched**	**Not Matched**
lecture[1,2].pdf	lecture1.pdf, lecture2.pdf	lecture5.pdf
vacation[4-9].jpg	vacation7.jpg, vacation9.jpg	vacation3.jpg
[a-z,A-Z][0-9].gif	a8.gif, M4.gif, Z0.gif	aY3.gif, 8a.gif

Putting Them Together

These wildcards are handy individually, but by using them in combination with each other, they become very powerful.

Examples:

Input	**Matched**	**Not Matched**
i[a-z]e	gift_ideas, profile.doc, notice	dRiVeR.eXe
[b,f][a,o][r,o].mp?	foo.mp3, bar.mp4, for.mpg	foo.mpeg
*min[a-z]y	minty, pepperminty, mindy	minutely, hominy

Lecture Outline

1 More About BASH
- Variables
- Making BASH Work for You
- Pattern Matching (Globbing)

2 Screen
- What Where Why?
- Handy Key Commands

What is the Problem?

There are a few problems with your basic BASH session. Some of these you may even have encountered already:

- Your session isn't preserved if you close your ssh connection
- It's a pain to switch back and forth between files/the prompt
- Sometimes using two or three shells at once would be really convenient!

All of these complaints can be resolved by using **screen**.

Intro to Screen

The screen Command

screen - a screen manager with terminal emulation

- (Lets you do all that cool stuff from the last slide!)

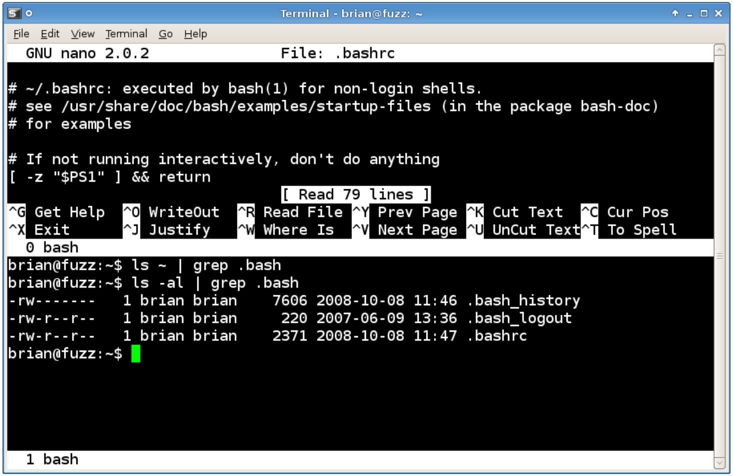
Generally **screen** can be used just as you would normally use a terminal window. However, special commands can be used to allow you to save your session, create extra shells, or split the window into multiple independent panes.

Passing Commands to screen

Each **screen** command consists of CTRL-a (hereafter referred to as C-a) followed by another character (case-sensitive!).

Screen in Action

A screenshot of a screen terminal:



```
  GNU nano 2.0.2                File: .bashrc

# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
[ -z "$PS1" ] && return
                               [ Read 79 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
  0 bash
brian@fuzz:~$ ls ~ | grep .bash
brian@fuzz:~$ ls -al | grep .bash
-rw-------   1 brian brian    7606 2008-10-08 11:46 .bash_history
-rw-r--r--   1 brian brian     220 2007-06-09 13:36 .bash_logout
-rw-r--r--   1 brian brian    2371 2008-10-08 11:47 .bashrc
brian@fuzz:~$ 





  1 bash
```

Detaching/Reattaching

Detach a screen

C-a d

- Detaches the current screen session, allowing you to resume it later from a different location without losing your work!

Resume a screen

screen -r [pid.tty.host]

- Resumes a detached screen session

screen -x [pid.tty.host]

- Attach to a non-detached screen session

If you have only one screen, the [pid.tty.host] string is unnecessary.

Identifying Screen Sessions

Screen Listing

screen -ls or **screen -list**

- Lists your screen sessions and their statuses

These screen sessions are the [pid.tty.host] strings required for resuming!

Resuming a Screen

If **screen -ls** returns *9951.pts-2.fuzz (Detached)*...

- **screen -r 9951.pts-2.fuzz** will resume our screen

Lecture Outline

1 More About BASH
- Variables
- Making BASH Work for You
- Pattern Matching (Globbing)

2 Screen
- What Where Why?
- Handy Key Commands

Creating More Shells

Create a New Shell Window

C-a c

- Creates a new shell in a new window and switches to it
- Useful for opening multiple shells in a single terminal
- Concept is similar to tabbed browsing/tabbed IMs

But how do we switch between windows? (hint: every window is numbered by order of creation)

Window Selection

C-a 0 - Switch to window 0
C-a 9 - Switch to window 9

Splitting Screen

Split Screen Computing

C-a S - splits your terminal area into multiple panes
C-a tab - changes the input focus to the next pane

- The 'S' is case-sensitive!
- Each split results in a blank pane
- Use C-a c to create a new shell in a pane
- Use C-a <num> to move an existing window to a pane

Note:

When you reattach a split **screen**, the split view will be gone. Just re-split the view, then switch between panes and reopen the other windows in each with C-a <num>