Bash Control Structures

Class 8

Environment Variables

- in the previous slides we saw how variables are created
- and where some environment variables are created
- but we didn't talk about why they are created

A Portion of My .bashrc

```
EDITOR=/usr/bin/emacs
PAGER=/usr/bin/less
LESS="-X -R"
ENSCRIPT="-G -r -2"

PATH=$HOME/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/sbin:/usr/sbin

export PATH EDITOR PAGER LESS HOME ENSCRIPT
```

EDITOR many commands (git, crontab, vipw, etc) automatically open an editor; this is the one I want them to use

PAGER many commands (git, man, mutt, etc) automatically invoke a pager; I want them to use less and not the default program more

More Environment Variables

LESS when less starts, it looks to see if this environment variable exists; if it does, it uses these options
when I enter `$ less foo` it's as though I entered `$ less -X -R foo`

ENSCRIPT a pretty-printing program like a2ps; these are the default options I want it to assume so I don't have to type them

PATH

- the final environment variable I set in .bashrc is PATH
- most commands I issue are in /bin or /usr/bin
- system management commands are in /sbin or /usr/sbin

- there are some non-standard programs I routinely run: a password generator, the Arduino IDE, etc.
- they live in /usr/local/bin
- there are some scripts and programs I wrote that I routinely run: the shell script to detect and configure the portable scanner I often use, the Perl script to allow non-members to post to the CS faculty mailing list, etc.
- they live in ∼/bin

PATH

- let's say I wish to run the password generator
- I don't want to type `$ /usr/local/bin/passgen`
- I want to type `$ passgen`

- when I type a command, bash first looks to see if it is a built-in
- if not, it looks in the first directory listed in the PATH environment variable
- then the next directory, etc

Finding a Command

- `$ echo $PATH`
  /home/jbeck/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/sbin:/usr/sbin
- note that the current directory . is not listed
- what if my PATH variable were this?
  .:/home/jbeck/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/sbin:/usr/sbin
- in this case, I could type `$ foo` and if foo was an executable file in the current directory, foo would execute
- but I do not have the current directory . in my path, so instead I have to enter `$ ./foo`
- why?

777 Directory

- let's say I have . on my path, and I also have a directory with permissions 777
- a 777 directory allows anyone on the system to create a file there
- so a malicious user can put an executable file named "ls" in that directory
- the next time I'm in that directory and issue the ls command, I don't run the system's ls, I run the bad guy's ls

Line Length

- in order to grade your programs, I print them, write comments on them, scan them, and upload the scans
- I print in portrait mode, and in order to be readable, that limits lines to 78 columns
- what do you do when you have a long line in a bash script?

- a newline is irrelevant in a C program
- in bash it is a metacharacter that causes the interpreter to accept the line for execution
- to "wrap" a long line, you must escape the newline with a backslash

```
$ foobar foobar foobar foobar foobar foobar \
      foobar foobar foobar foobar foobar \
      foobar foobar foobar
```

Exit Status

- every process on a Unix computer eventually ends
- when it ends, like a function, it returns an integer to the process that called it
- that integer value is the exit status or return status of the process

- the return status can be seen by the calling process
- in bash, the return value of a command is in the variable $?

Exit Status

```
$ ls
$ echo $?
0

$ ls xxfoobarxx
$ echo $?
2
```

from the ls man page:
Exit status:
0 if OK,
1 if minor problems (e.g., cannot access subdirectory),
2 if serious trouble (e.g., cannot access command-line argument).

Another Example

from the man page of pdftotext:

```
EXIT CODES
       The Xpdf tools use the following exit codes:

       0       No error.
       1       Error opening a PDF file.
       2       Error opening an output file.
       3       Error related to PDF permissions.
       99      Other error.
```

Exit Status

shell scripts have an exit status also

```
#!/bin/bash

...

exit 5
```

Control Flow Constructs

bash has constructs for

- branching
  - if - then - elif - else - fi
  - case - esac
- looping
  - for - do - done
  - while - do - done
  - until - do - done
- functions

if - fi

```
if expression
then
  command
  ...
elif expression
then
  command
  ...
else
  command
  ...
fi
```

- zero or more elifs
- zero or one elses
- note indenting style

- expression is a list of commands
- built-in or external
- the ultimate result of the expression is an exit status
- if the exit status is 0, that equates to true
- a non-zero exit status equates to false

if - fi Examples

- by far the most common command in expression is test
- test is a built-in command (there's an external version rarely used)

```
if test -a foo
then
  echo "foo exists"
else
  echo "foo does not exist"
fi
```

Second Form of Test

- almost no one ever uses "test"
- instead, there is a synonym for test named [
- [is both the name of a command, and an open-delimiter that does nothing except provide syntax
- as an open-delimiter, it must have a matching close-delimiter
- there must be a space after [and before]
- so the previous slide becomes:

```
if [ -a foo ]
then
  echo "foo exists"
else
  echo "foo does not exist"
fi
```

New Form of Test

- new code never uses [(although you'll find lots of old examples on the web)
- modern bash uses the new form [[expression]]
- there must be a space after [[and before]]
- so the previous slide now becomes:

```
if [[ -a foo ]]
then
  echo "foo exists"
else
  echo "foo does not exist"
fi
```

Common Options for Test

File Tests		Integer Tests	
Option	Result	Option	Result
-d file	file exists and is a directory?	i -eq j	true if i == j
-f file	is file a regular file?	i -ge j	true if i >= j
-r file	is file readable?	i -gt j	true if i > j
-s file	is file nonempty?	i -le j	true if i <= j
-w file	is file writable?	i -lt j	true if i < j
-x file	is file executable?	i -ne j	true if i != j

String Testing

Option	Return
str	true if str is not an empty string
s1 = s2	true if s1 and s2 are ASCII identical
s1 != s2	true if s1 and s2 are not identical strings
-n str	true if str is not empty
-z str	true if the length of str is zero
s1 < s2	true if s1 comes before s2 lexicographically in ASCII
s1 > s2	true if s1 comes after s2 lexicographically in ASCII

Complex Expressions

- simple expressions can be combined with
  - and — old way: -a new way: &&
  - or — old way: -o new way: ||
  - not !
  - grouping ( )

```
if [[ ('bar' < 'foo') && ('1' -lt '02') ]]
then
  echo 'yes'
else
  echo 'no'
fi
```

Demo

example script if_demonstration.sh

```
$ rsync -vuptz user@sand.truman.edu:/tmp/if_demonstration.sh .
```

or

```
$ rsync -vuptz /tmp/if_demonstration.sh .
```