Understanding bash

Prof. Chris GauthierDickey

COMP 2400, Fall 2008

How does bash start?

- It begins by reading your configuration files:
  - If it's an interactive login-shell, first /etc/profile is executed, then it looks for:
    - .bash_profile, .bash_login and ~/.profile
  - If it's a just an interactive shell, /etc/bash.bashrc is executed, followed by:
    - ~/.bashrc
  - Usually, people put login stuff in .profile and interactive stuff in .bashrc and do a source ~/.bashrc from your .profile or other login script

I/O Redirection

- Sometimes you want to use a Unix utility, but it doesn't take standard input
  - Try adding a '-' at the end
- If you want to save output to a file
  - Use '>' which sends standard output to filename
  - Use '>>' to append the standard output to filename
- If you want to take input from a file
  - Use '< filename' which reads standard input from filename

UNIVERSITY OF DENVER

Piping

- We use the pipe command '|' to take the standard output from one command and send it to the standard input of another command
  - cat file | more

UNIVERSITY OF DENVER

Background Jobs

- We can execute any command automatically in the background by adding a & At the end
  - If a program is running, hit CTRL-Z and then type 'bg'
- To move a job to the foreground, type 'fg'
- To list your jobs, type 'jobs'

Saving Typing

- Bash keeps a history of all the commands you execute
  - Enter a command, then hit the up-arrow key
  - Type 'history'
  - The variable HISTSIZE sets the size of your history

UNIVERSITY OF DENVER

More History

- List the history: notice the numbers?
  - history 20 will show the last 20
  - !num, where num is one of those numbers will repeat that command
  - !! repeats the last command
  - !: will let you enter a command...remember sed?
    - !:s/xy/yx
  - !foo will repeat the last command starting with foo

UNIVERSITY OF DENVER

History cont.

- ^^ is for substitution
  - cat myflie
  - ^li^il will sub the last command as 'cat myfile'
- !$ will return the last argument of the last command
  - cat myfile
  - 'rm $!' will be substituted with 'rm myfile'
- !:n* will return the nth (0-9) command to the end

- !?foo? repeats the last command with foo anywhere in it
- !! & adds an & to the last command
- !* is shorthand for all but the command name of the last command

UNIVERSITY OF DENVER

File Permissions

- Remember back at the start of class when we discussed ls -l?
  - _rwxrwxrwx lists the permissions
  - Use the command 'chmod' to change permissions
  - Each triplet is represented by an octal number:
    - 4=r, 2=w, 1=x
      - chmod 754 = rwxr_xr__

Shell Scripting

- Shell scripting provides an easy way to combine commands using bash
- Begin your file with #!/usr/bin/bash

Variables

- Variables are assigned to by =
  - foo=1
- Variables are read from by using $ in front
  - 'echo $foo' will print 1 on the screen
- Arguments to the script are in variables $0 to $n
- $* contains a list of all the args
- $# returns the number of arguments

More on Variables

- $name is actually a shortcut for ${name}
  - ${10} for example, is necessary to access argument 10
  - ${varname:-word} returns varname or returns word if varname doesn't exist or is null
    - ${foo:-10} returns $foo or 10 if $foo doesn't exist
  - ${varname:=word} does the same as -, but sets the variable to the default value in the process
  - ${varname:?message} prints message if varname doesn't exist or returns varname

- ${varname:+word} if varname exists and isn't null, word is returned

- ${varname:offset:len} returns the offset through len characters of the string (counting from 0)

UNIVERSITY OF DENVER

If/else

- The if/else command lets us do conditional branching
- Truth in Unix is typically 0, for a non-error exit state
- False is anything else
- The last command executed is the exit status by default

```
if condition
then
   statements
elif condition
   then
      statements
else
   statements
fi
```

UNIVERSITY OF DENVER

Conditions

- Commands return their status
- We can combine with && and ||
  - if statement1 && statement2
  - if statement1 || statement2
- For bracketed conditions:
  - [condition1] && [condition2]
  - [condition1] || [condition2]

Using []

- [] perform various non-exit-status tests
  - [str1 = str2]
  - [str1 != str2]
  - [str1 \< str2]
  - [str1 \> str2]
    - We must escape < and > in the [ ] construct
  - [-n str1] : str1 is not null (0 length)
  - [-z str1] : str1 is null (has 0 length)

UNIVERSITY OF DENVER

- -a file or -e file: file exists
- -d file: file exists and is a directory
- -f file: file exists and is a regular file
- -r file: you can read the file
- -s file: file exists and is non-empty
- -w file: you can write to the file
- -x file: you can execute the file

UNIVERSITY OF DENVER

- -N file: file was modified since it was last read
- -O file: you own the file
- -G file: file belongs to one of your groups
- file1 -nt file2: file1 is newer than file2
- file1 -ot file2: file1 is older than file2

[[ ]] as an alternative

- [[ ]] is an alternative to [ ]
  - It can use && and || within it, for example
  - It will not do globbing (expanding wildcards), but will substitute variables and do command substitution
- if [[$1 < $2]]; then ...
  - Notice that we don't have to escape < this time (in fact, it's an error if we do)
- It takes the same kinds of arguments as [ ]

Returns

- We can exit by a return or exit statement:
  - return, return 0, return 1, etc...
  - return without an argument returns the value of the last command run
  - exit statements exit the entire script, returns can return from functions (later)