Bashed One Too Many Times

Features of the Bash Shell

St. Louis Unix Users Group

Jeff Muse, Jan 14, 2009

What is a Shell?

- The shell interprets commands and executes them
  - It provides you with an environment in which to work
  - It provides an entire programming language which can be used either in scripts or directly from the command line

Why Bash?

- It is the standard shell on Linux and is available for almost every version of Unix
- As we'll see, it is feature-rich
- It is mostly compatible with the earlier Bourne shell and Korn shell
- It is even available for Windows under Cygwin

What We'll Cover

- Metacharacters
- Variables
- Quoting
- Pattern Matching
- File Descriptors and Redirection
- Built-in Commands
- Startup Files
- Keyboard Shortcuts
- History
- Job Control and Processes
- Options
- Tests
- Flow Control

Metacharacters

- Metacharacters are used to pass special directions to the shell. They include *, [ ],?,&,`,\,{ },> ,>>,<,( ), ;,/, $, and |
- Metacharacters must be escaped if you do not want them interpreted by the shell

Escaping Metacharacters

Variables

- Variables are containers for some value
- They are designated by a '$' unless they are being initialized
- They are restricted to the current shell unless they are exported
- View them with the 'env' command

Variable Expansion

- If there is any ambiguity over exactly what the variable is, use {} to specify

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ HERO="Linus Torvalds"
jeff@brubeck:~$ echo $HERO
Linus Torvalds
jeff@brubeck:~$ echo foo$HERObar
foo
jeff@brubeck:~$ echo foo${HERO}bar
fooLinus Torvaldsbar
jeff@brubeck:~$
```

Unexported Variables

```
jeff@brubeck: ~
File  Edit  View  Terminal  Tabs  Help
jeff@brubeck:~$ a=7
jeff@brubeck:~$ echo $a
7
jeff@brubeck:~$ bash

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

 * Found existing ssh-agent (8415)
SSH_AUTH_SOCK=/tmp/ssh-ajocEm8414/agent.8414; export SSH_AUTH_SOCK;
SSH_AGENT_PID=8415; export SSH_AGENT_PID;

 * Known ssh key: /home/jeff/.ssh/id_dsa
 * Known ssh key: /home/jeff/.ssh/id_rsa

jeff@brubeck:~$ echo $a

jeff@brubeck:~$ exit
exit
jeff@brubeck:~$ echo $a
7
jeff@brubeck:~$
```

Exported Variables

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ a=7
jeff@brubeck:~$ export a
jeff@brubeck:~$ bash

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

 * Found existing ssh-agent (8415)
SSH_AUTH_SOCK=/tmp/ssh-ajocEm8414/agent.8414; export SSH_AUTH_SOCK;
SSH_AGENT_PID=8415; export SSH_AGENT_PID;

 * Known ssh key: /home/jeff/.ssh/id_dsa
 * Known ssh key: /home/jeff/.ssh/id_rsa

jeff@brubeck:~$ echo $a
7
jeff@brubeck:~$
```

Special Variables

- $? - exit code of the last command
- $! - PID of last backgrounded command
- $$ - PID of current shell

Built-in Variables

- PS1
- PS2
- HOME
- PATH
- HISTFILESIZE
- HISTSIZE
- OLDPWD

Quoting (from Uwe Waldman)

- Single quotes ('...') protect everything (even backslashes, newlines, etc.) except single quotes, until the next single quote.
- Double quotes ("...") protect everything except double quotes, backslashes, dollar signs, and backquotes, until the next double quote. A backslash can be used to protect ", \, $, or ` within double quotes. A backslash-newline pair disappears completely; a backslash that does not precede ", \, $, `, or newline is taken literally.

Quoting Examples

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ VAR='foo bar'
jeff@brubeck:~$ echo $VAR
foo bar
jeff@brubeck:~$ VAR='foo$bar$baz'
jeff@brubeck:~$ echo $VAR
foo$bar$baz
jeff@brubeck:~$ VAR="foo$bar"
jeff@brubeck:~$ echo $VAR
foo
jeff@brubeck:~$ VAR=`date`
jeff@brubeck:~$ echo $VAR
Sun Jan 11 18:47:46 CST 2009
jeff@brubeck:~$ VAR='`date`'
jeff@brubeck:~$ echo $VAR
`date`
jeff@brubeck:~$
```

Pattern Matching

- ? matches one character
- * matches zero or more characters
- [] is used to match characters in between brackets
- [a-z] will match any lowercase letter
- [1-9] will match any digit
- {} will match a comma-separated list
  {local,lib} will match either 'local' or 'lib'
- {} can also be used to generate substrings

Examples of Pattern Matching

jeff@brubeck: /tmp

File Edit View Terminal Tabs Help

```
jeff@brubeck:/tmp$ touch file1 file2 file3 file13
jeff@brubeck:/tmp$ ls file?
file1  file2  file3
jeff@brubeck:/tmp$ ls file*
file1  file13  file2  file22  file3
jeff@brubeck:/tmp$ ls file[1-2]
file1  file2
jeff@brubeck:/tmp$ ls file[1-5]
file1  file2  file3
jeff@brubeck:/tmp$ ls fi[a-z]e1
file1
jeff@brubeck:/tmp$ ls file{1,13}
file1  file13
jeff@brubeck:/tmp$ ls f[im]e*
ls: cannot access f[im]e*: No such file or directory
jeff@brubeck:/tmp$
```

Substrings

- {} can be used to match substrings
- ${VAR:0} is the entire variable name
- ${VAR:0:1} returns the first character
- ${VAR:0:2} returnes the first two
- ...and so on. The first digit is your start position, the second is the number of digit to return

Substring Example

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ VAR="SLUUG"
jeff@brubeck:~$ echo $VAR
SLUUG
jeff@brubeck:~$ echo ${VAR:0}
SLUUG
jeff@brubeck:~$ echo ${VAR:0:1}
S
jeff@brubeck:~$ echo ${VAR:0:2}
SL
jeff@brubeck:~$ echo ${VAR:0:5}
SLUUG
jeff@brubeck:~$ echo ${VAR:2:4}
UUG
jeff@brubeck:~$
```

Posix Character Classes Support

- alnum
- alpha
- ascii
- blank
- cntrl
- digit
- graph
- lower
- print
- punct
- space
- upper
- word
- xdigit

Character Class Examples

jeff@brubeck: /tmp/posix

File Edit View Terminal Tabs Help

```
jeff@brubeck:/tmp/posix$ touch 33 l33t "leet h@x0r"
jeff@brubeck:/tmp/posix$ ls [[:digit:]][[:digit:]]
33
jeff@brubeck:/tmp/posix$ ls [[:alpha:]]*[[:alpha:]]
l33t  leet h@x0r
jeff@brubeck:/tmp/posix$ ls *[[:space:]]*
leet h@x0r
jeff@brubeck:/tmp/posix$ ls *[[:lower:]]*
l33t  leet h@x0r
jeff@brubeck:/tmp/posix$ touch L33T
jeff@brubeck:/tmp/posix$ ls *[[:upper:]]*
L33T
jeff@brubeck:/tmp/posix$ ls *[[:ascii:]]
33  l33t  L33T  leet h@x0r
jeff@brubeck:/tmp/posix$
```

File Descriptors

- By default, there are three file descriptors
  - 0, STDIN
  - 1, STDOUT
  - 2, STDERR
- They are all pointed to your terminal by default, but can be re-directed

Redirection Operators

- | takes the output of one command and feeds it into another
- > redirects STDOUT and overwrites
- >> redirects STDOUT and appends
- < redirects STDIN

Redirection Example

jeff@brubeck: /tmp

File Edit View Terminal Tabs Help

```
jeff@brubeck:/tmp$ ls -l file* > list.txt
jeff@brubeck:/tmp$ cat list.txt
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file1
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file13
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file2
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:36 file22
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file3
jeff@brubeck:/tmp$ ls -l fooobar >> list.txt
jeff@brubeck:/tmp$ cat list.txt
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file1
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file13
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file2
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:36 file22
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file3
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:48 fooobar
jeff@brubeck:/tmp$ ls -l nonexistent.txt 2> error.txt
jeff@brubeck:/tmp$ cat error.txt
ls: cannot access nonexistent.txt: No such file or directory
jeff@brubeck:/tmp$ ls -l nonexistent.txt 2>> list.txt
jeff@brubeck:/tmp$ cat list.txt
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file1
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file13
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file2
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:36 file22
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:40 file3
-rw-r--r-- 1 jeff jeff 0 2009-01-07 21:48 fooobar
ls: cannot access nonexistent.txt: No such file or directory
jeff@brubeck:/tmp$
```

Using Redirection to Read Files

jeff@brubeck: /tmp

File Edit View Terminal Tabs Help

```
jeff@brubeck:/tmp$ cat read.txt
line1
line2
line3
jeff@brubeck:/tmp$ while read line; do echo $line; done < read.txt  > out.txt
jeff@brubeck:/tmp$ cat out.txt
line1
line2
line3
jeff@brubeck:/tmp$
```

More Redirection – Tying STDOUT and STDERR

jeff@brubeck: /tmp

File Edit View Terminal Tabs Help

```
jeff@brubeck:/tmp$ ls file* nonexistent.txt > stderr_stdout.txt 2>&1
jeff@brubeck:/tmp$ cat stderr_stdout.txt
ls: cannot access nonexistent.txt: No such file or directory
file1
file13
file2
file22
file3
jeff@brubeck:/tmp$
```

Opening and Closing File Descriptors

jeff@brubeck: /tmp

File Edit View Terminal Tabs Help

```
jeff@brubeck:/tmp$ lsof -a -p $$ -d0,1,2,3
COMMAND   PID USER   FD   TYPE DEVICE SIZE NODE NAME
bash    15401 jeff    0r   CHR  136,1         3 /dev/pts/1
bash    15401 jeff    1w   CHR  136,1         3 /dev/pts/1
bash    15401 jeff    2w   CHR  136,1         3 /dev/pts/1
jeff@brubeck:/tmp$ exec 3<>fd3.txt
jeff@brubeck:/tmp$ lsof -a -p $$ -d0,1,2,3
COMMAND   PID USER   FD   TYPE DEVICE SIZE NODE NAME
bash    15401 jeff    0r   CHR  136,1         3 /dev/pts/1
bash    15401 jeff    1w   CHR  136,1         3 /dev/pts/1
bash    15401 jeff    2w   CHR  136,1         3 /dev/pts/1
bash    15401 jeff    3u   REG    8,9    0 6148 /tmp/fd3.txt
jeff@brubeck:/tmp$ exec 3>&-
jeff@brubeck:/tmp$ lsof -a -p $$ -d0,1,2,3
COMMAND   PID USER   FD   TYPE DEVICE SIZE NODE NAME
bash    15401 jeff    0r   CHR  136,1         3 /dev/pts/1
bash    15401 jeff    1w   CHR  136,1         3 /dev/pts/1
bash    15401 jeff    2w   CHR  136,1         3 /dev/pts/1
jeff@brubeck:/tmp$
```

Here Documents

- Here documents redirect multiple lines of input easily.
- They are often used in shell scripts and are ended by an arbitary string, which by convention is often 'EOF'

Here Document Example

jeff@brubeck: /tmp

File Edit View Terminal Tabs Help

```
jeff@brubeck:/tmp$ cat << EOF > here.txt
> This is a here document.
> It spans multiple lines.
> EOF
jeff@brubeck:/tmp$ cat here.txt
This is a here document.
It spans multiple lines.
jeff@brubeck:/tmp$
```

Built-in Commands

- cd
- eval
- exec
- exit
- export
- test
- alias
- unset
- echo
- shopt
- source
- ulimit

Startup Files

- Login shells (those that had to start via /bin/login reading /etc/passwd) use these files, in order
  - /etc/profile
  - .bash_profile
  - .bash_login
  - . .profile

Startup Files (continued)

- Non-login shells only read .bashrc
- It is not unusual for .bash_profile to source .bashrc

Keyboard shortcuts

- The readline library is used by bash to provide command-line editing
  - CTL-A – go to beginning of line
  - CTL-E – go to end of line
  - CTL-U – erase all of line
  - CTL-K – erase from cursor to end of line
  - CTL-R – reverse search
  - CTL-D – logout (technically EOF, and only at the beginning of a line – otherwise, delete)

History

- Up arrow - go one command back in history
- The 'history' command shows the last ten commands and their numbers ('history -*n*' shows the last *n* commands)
- The 'fc' command can be used to edit and execute earlier commands
- The last command is aliased as '!!'
- The last argument to the last command is aliased as '!$'
- Other arguments have numbers assigned to them

History Examples

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ echo foo bar baz
foo bar baz
jeff@brubeck:~$ echo !:1
echo foo
foo
jeff@brubeck:~$ echo foo bar baz
foo bar baz
jeff@brubeck:~$ echo !:2
echo bar
bar
jeff@brubeck:~$ echo foo bar baz
foo bar baz
jeff@brubeck:~$ echo !:3
echo baz
baz
jeff@brubeck:~$ echo foo bar baz
foo bar baz
jeff@brubeck:~$ echo !$
echo baz
baz
jeff@brubeck:~$
```

'history' and 'fc'

- The 'history' command recalls prior commands by number. You can specify *n* commands to recall with 'history *n*'
- 'fc' does much the same thing when run as 'fc -l'
- 'fc *n*' drops you into an editor (you may need to define your editor in $FCEDIT) to edit command *n*. Once you save and quit, your edited command is run.

Job Control

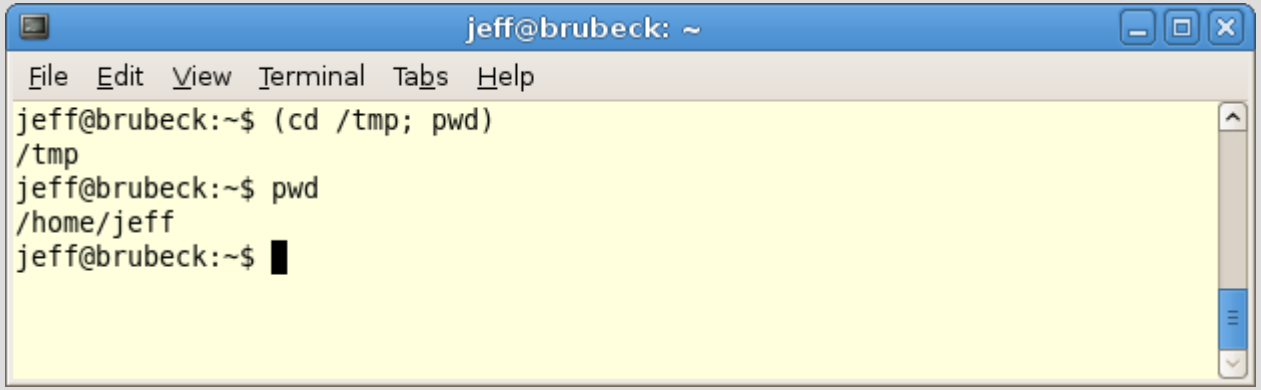
- Commands can be backgrounded with '&'
- Commands can be foregrounded with 'fg'
- Commands can be suspended with CTL-Z
- Commands can be killed with CTL-C
- Suspended jobs can be listed with the 'jobs' command
- Suspended jobs can be killed with 'kill %*n*' where *n* is the job number
- Backgrounded jobs can be detached from a terminal with 'disown'

Job Control (continued)

- Process can be started in subshells with parentheses
- Commands can be run in backticks to capture their output

Subshells

- A subshell is spawned by commands run inside of parentheses. When these commands are finished, the state of the parent shell is maintained. Subshells inherit the parent shell's environment.



Subshells in SSH

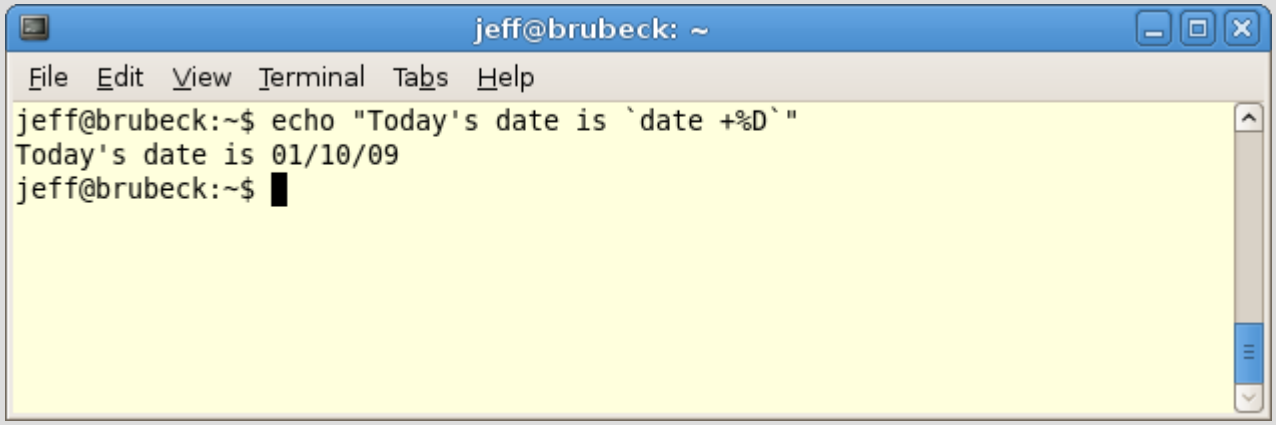
jeff@brubeck: /tmp

File Edit View Terminal Tabs Help

```
jmuse@bud:~$ ls fil*
file1  file2  file3
jmuse@bud:~$ ssh budlight ls /tmp/file*
This system for use by SLUUG members ONLY. Unauthorized access prohibited.
jmuse@budlight's password:
ls: /tmp/file*: No such file or directory
jmuse@bud:~$ tar cvf - file* | ssh budlight '(cd /tmp; tar xvf -)'
file1
file2
file3
This system for use by SLUUG members ONLY. Unauthorized access prohibited.
jmuse@budlight's password:
file1
file2
file3
jmuse@bud:~$ ssh budlight ls /tmp/file*
This system for use by SLUUG members ONLY. Unauthorized access prohibited.
jmuse@budlight's password:
/tmp/file1
/tmp/file2
/tmp/file3
jmuse@bud:~$
```

Command Substitution

- `` (backticks) are used to run a command and send its output to a variable or another command



Options

- Options can be configured with 'set -o optionname'
- They are un-configured with 'unset optionname'
- Some interesting options:
  - emacs|vi
  - noclobber
  - ignoreeof
  - noglob

More options

- Options can be configured with the built-in shopt command

jeff@brubeck: /tmp/case

File Edit View Terminal Tabs Help

```
jeff@brubeck:/usr/local$ shopt -u cdspell
jeff@brubeck:/usr/local$ cd /usr/lcoal
bash: cd: /usr/lcoal: No such file or directory
jeff@brubeck:/usr/local$ shopt -s cdspell
jeff@brubeck:/usr/local$ cd /usr/lcoal
/usr/local
jeff@brubeck:/usr/local$ cd /tmp/case
jeff@brubeck:/tmp/case$ ls
file.flac  file.mp3  file.ogg
jeff@brubeck:/tmp/case$ shopt -s nocaseglob
jeff@brubeck:/tmp/case$ ls File*
file.flac  file.mp3  file.ogg
jeff@brubeck:/tmp/case$
```

Math Examples

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ a=$((3+2))
jeff@brubeck:~$ echo $a
5
jeff@brubeck:~$ a=$((4/2))
jeff@brubeck:~$ echo $a
2
jeff@brubeck:~$ a=$((7-4))
jeff@brubeck:~$ echo $a
3
jeff@brubeck:~$ a=$((2*4))
jeff@brubeck:~$ echo $a
8
jeff@brubeck:~$ a=$((22/7))
jeff@brubeck:~$ echo $a
3
jeff@brubeck:~$
```

Tests

- Tests are used to evaluate the truth of an expression
- 'test *expr'* and '[*expr*]' are equivalent. [is a built-in command
- '[[*expr*]]' is an alternate syntax, without file globbing. [[is a reserved word
- Tests are generally negated with '!'

File Tests

- -d Directory
- -e Exists
- -f Regular file
- -h Symbolic link (also -L)
- -p Named pipe
- -r Readable by you
- -s Not empty
- -wWritable by you

Numeric Tests

- -lt – less than
- -gt – greater than
- -eq – equal to
- -le – less than or equal to
- -ge – greater than or equal to

String Comparisons and Tests

- =,== Equal to
- != Not equal to
- > ASCII value is greater than
- < ASCII value is less than
- -z String is zero length
- -n String is not null
- Caution – always quote strings when testing them!

Single Bracket Tests

- Note the errors!

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ [ -d /tmp ]
jeff@brubeck:~$ echo $?
0
jeff@brubeck:~$ [ -e /nothere ]
jeff@brubeck:~$ echo $?
1
jeff@brubeck:~$ string="test"
jeff@brubeck:~$ [ -z $string ]
jeff@brubeck:~$ echo $?
1
jeff@brubeck:~$ [-n $string ]
bash: [-n: command not found
jeff@brubeck:~$ [ -n $string ]
jeff@brubeck:~$ echo $?
0
jeff@brubeck:~$ [ 1 = 2 ]
jeff@brubeck:~$ echo $?
1
jeff@brubeck:~$ [ 1 < 2 ]
bash: 2: No such file or directory
jeff@brubeck:~$
```

Double Bracket Tests

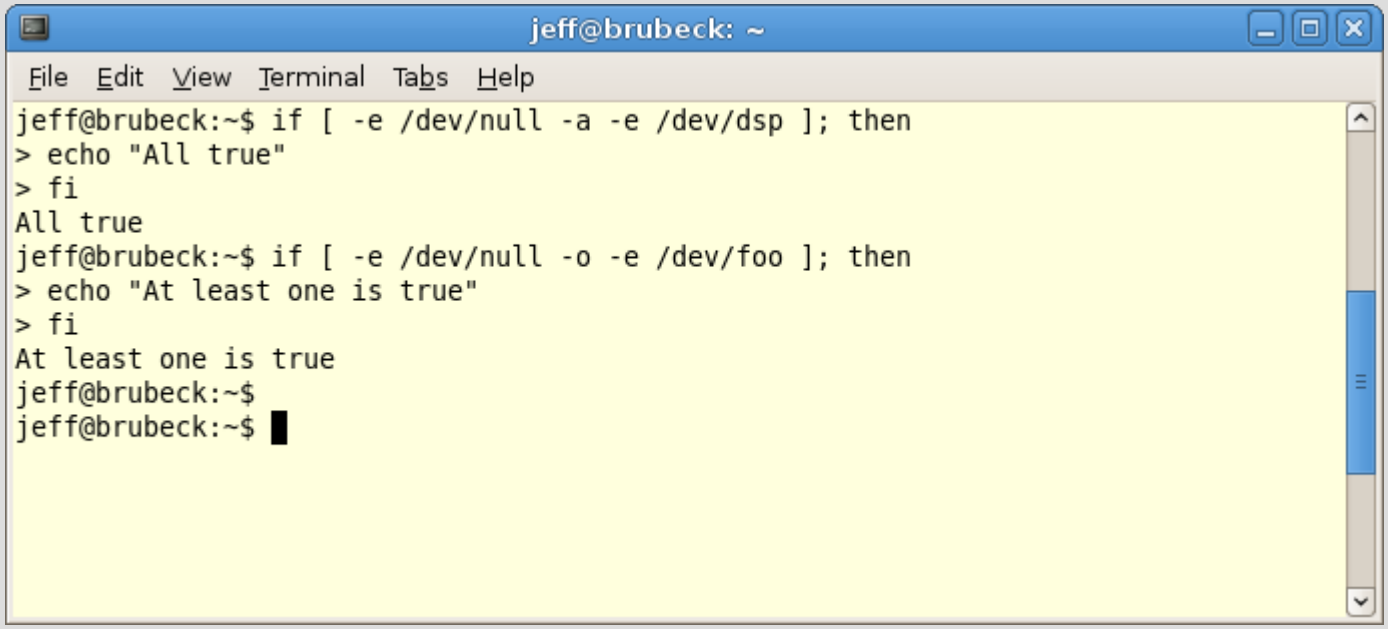
jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ [[ 0 > 1 ]] && echo "True"
jeff@brubeck:~$ [[ 0 > 1 ]] || echo "True"
True
jeff@brubeck:~$ [[ 0 < 1 ]] && echo "True"
True
jeff@brubeck:~$ [[ -e /dev/null ]] && echo "/dev/null exists"
/dev/null exists
jeff@brubeck:~$ [[ -e /dev/foo ]] && echo "/dev/foo exists"
jeff@brubeck:~$ [[ -e /dev/foo ]]
jeff@brubeck:~$ echo $?
1
jeff@brubeck:~$
```

Compound Tests

- Use '-o' and '-a' inside of test/single brackets
- The equivalents in [[are '&&' and '||'



Flow Control

- If – do something if condition is true
- If -then-else – do one thing if something is true, otherwise do something else
- For – do something upon each member of a list
- While
- Until
- Case – like 'if-then-else' but gracefully handles more possibilities

If

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ if [ -d $HOME ]; then echo "You exist"; fi
You exist
jeff@brubeck:~$ if [ ! -d $HOME ]; then echo "You are homeless"; fi
jeff@brubeck:~$ if [ -e /dev/null ]; then echo "mknod not needed"; fi
mknod not needed
jeff@brubeck:~$ grep foo nonexistent.txt
grep: nonexistent.txt: No such file or directory
jeff@brubeck:~$ if [ $? -ne 0 ]; then echo 'Grep Error!'; fi
Grep Error!
jeff@brubeck:~$
```

If-then-else

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ if [ 1 -gt 0 ]; then echo "One is greater than zero"; else echo "
Shouldn't get here"; fi
One is greater than zero
jeff@brubeck:~$ if [ 1 -lt 0 ]; then echo "One is greater than zero"; else echo "
Shouldn't get here"; fi
Shouldn't get here
jeff@brubeck:~$
```

For

jeff@brubeck: /tmp

File Edit View Terminal Tabs Help

```
jeff@brubeck:/tmp$ for i in 1 2 3
> do echo $i
> done
1
2
3
jeff@brubeck:/tmp$ ls *mp3
bar.mp3  foo.mp3
jeff@brubeck:/tmp$ for file in *mp3
> do mv $file `echo $file | sed -e 's/mp3/ogg/'`
> done
jeff@brubeck:/tmp$ ls *mp3
ls: cannot access *mp3: No such file or directory
jeff@brubeck:/tmp$ ls *ogg
bar.ogg  foo.ogg
jeff@brubeck:/tmp$
```

While

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ i=0;while [ $i -lt 5 ]; do echo $i; i=`expr $i + 1`; done
0
1
2
3
4
jeff@brubeck:~$
```

Until

jeff@brubeck: ~

File Edit View Terminal Tabs Help

```
jeff@brubeck:~$ i=0;until [ $i -gt 5 ]; do echo $i; i=`expr $i + 1`; done
0
1
2
3
4
5
jeff@brubeck:~$
```

Case

jeff@brubeck: /tmp/case

File Edit View Terminal Tabs Help

```
jeff@brubeck:/tmp/case$ ls
file.flac  file.mp3  file.ogg
jeff@brubeck:/tmp/case$ for file in *; do
> case "$file" in *mp3)
> echo "$file is good";
> ;;
> *ogg) echo "$file is better";
> ;;
> *flac) echo "$file is best";
> ;;
> esac
> done
file.flac is best
file.mp3 is good
file.ogg is better
jeff@brubeck:/tmp/case$
```

References

http://www.gnu.org/software/bash/manual/bashref.html
http://tldp.org/LDP/abs/html/
http://learnlinux.tsf.org.za/courses/build/shell-scripting/ch12s04.html
http://linuxshellaccount.blogspot.com/2008/02/finding-and-reading-files-in-shell-when.html

Questions?