Exit status and conditionals

Lecture 6

Michael J. Wise

Meaning of *True*

- When programs conclude, apart from other things, they return an exit status
- Exit status of 0 implies successful
- Any other positive integer implies an error state (with the value indicating type)
- Also used as Boolean.

  0 ⇒ True; >0 ⇒ False (most typically 1)
- Exit status of last command available as `$?`, e.g. `echo $?`

Should be Exit Status 0

Aside: grep

`grep` *[options] <pattern string> <file> …*

- Search for the specified pattern string in one or more files.
- Returns matches on stdout (or nothing)
- Also returns an exit status `0` (found) `1` (not found)

```
grep Alice Alice_in_Wonderland.txt > /dev/null
echo $?
```

Demo

- Let’s look at

```
grep Alice Alice_in_Wonderland.txt >
/dev/null
echo $?
```

- What do you see if you’re in the directory containing the text?
- What do you if you search with `fred` ?
- What do you see if you search with `A lice` ?
- Use the command `> empty_book.txt` to create a zero-length file, and then run the search again looking for Alice in both the Alice text and the null book.

Conditionals: `if` .. `else`

- The conditional execution command use by Bash is fairly conventional, at least in appearance.

```
if <condition>
then
    <statements>
[elif <condition>
    <statements>] .....
[else
    <statements>]
fi
```

- The minimum statement uses `if  then  fi`

Conditionals: `if` .. `else`

- The `if` <condition> is evaluated and if true, the `then` statements are executed.
- Otherwise, if an `elif` clause exists its condition is

evaluated, and if that is successful, its then

statements are executed.

- This is repeated for each successive `elif` until either one is satisfied (and the then statements are executed) or no `elif` clauses remain.
- If neither the `if` condition, nor any of the `elif` conditions are satisfied, and an `else` clause exists, its statements are executed.

Conditionals: `if` .. `else`

```
if grep Alice Alice_in_Wonderland.txt >
/dev/null

then

   echo Found Alice

else

   echo Alice wasn\'t in Wonderland

fi
```

test

- `test` <*test expression*> is a program that doesn't output anything, but rather executes the test expression and returns the outcome in the exit status
- `man test` for all the tests. Some common examples:

`-d` <*directory*> Tests if the directory exists

`-f` <*file*> Tests if the file exists as ordinary file

`-s` <*file*> Test if the file exists and is not 0 length

`-n` <*string*> Is the string non-zero length

`-z` <*string*> Is the string zero length

Demo

```
> empty_book.txt # create empty file
if test -f empty_book.txt
then
    echo All Good
else
     echo Not so good
fi
```

- What do I see
- What do I see if the test is `-s` ?
- What do I see if the test is `-n` ?

`[[ .. ]]` replace `test`

- `test` is a program (`/bin/test`)
- Bash also implements built tests mimicking `test`
- Tests are between `[[` and `]]`
  - *leave space around brackets*

[[-s empty_book.txt]]

- `test` is ubiquitous; `[[ ]]` in Bash and Ksh (also more efficient)

Testing numbers

- There are also infix tests which can be easier in the brackets version

`[[ 1038 <  999 ]]` tests whether string `1038` is less than string `999`

  - *try* *`test a < b`*

- Can also handle numerical comparisons

```
[[ 1038 -lt 999 ]]
echo $?
```

Testing numbers

```
#!/usr/bin/env bash
if [[ $1 -ge 75 ]]
then
   echo Excellent
elif [[ $1 -ge 65 ]]
then
   echo Very Good
elif [[ $1 -gt 50 ]]
then
   echo Good
elif [[ $1 -eq 50 ]]
then
   echo "Well, OK"
else
   echo Aaaaaaaaaaaagggggggggggghhhhhhhh
fi
```

👉👉👉Anti-bugging 👈👈👈

- Anti-bugging is to not simply assume that the input users provide is in fact what the program expects, but to take steps to ensure it is, by adding prior tests.
  - *Sanity checks*
  - *This is important; GIGO is a thing*

Demo: Anti-bugging `count_occurences`

- `count_occurences` has two inputs, the text whose words are to be counted, and the number of top words to report. Add antibugging to make sure.

Starecat.com

Time for some Anti-bugging!