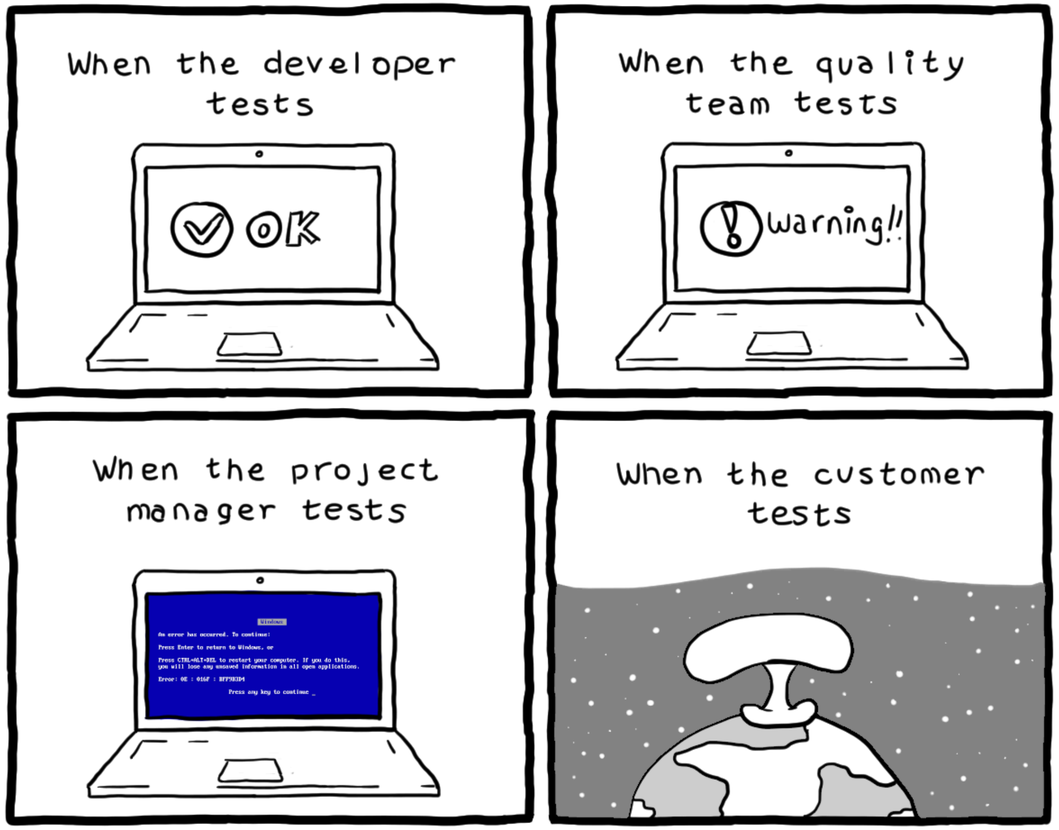
COMP2101
Summer 2022

bash Execution Control

Conditional Execution and Testing



- Command Lists
- Conditional Execution Operators
- Exit Status
- Test Command

Command Lists

- A command list is a list of one or more commands on a single command line in bash
- Putting more than one command on a line requires placement of an operator between the commands so that bash can tell them apart
- Operators also tell bash the rules for running the command sequence
- The ; operator between commands tells bash to run the first command and when it finishes to run the next one without regard for the success or failure of the first command

```
echo -n "Number of non-hidden files in this directory: " ; ls | wc -l
echo -n "Process count by username:" ; ps -eo user --no-headers | sort | uniq -c
echo -n "eno1 interface address:" ; ip a s eno1 | grep -w inet | awk '{print $2}'
```

Bash Conditional Execution

- When a command is run, it may fail and cause other commands to have problems or become unnecessary
- A command may need to be run only under specific circumstances
- A command may depend on another command finishing properly before it can be run
- In order to automate these things, bash provides operators to control the execution of commands in lists

Exit Status

- Every command that runs produces an exit status when it ends
- That exit status can be used to control whether or not the next command in a list should run
- The exit status of a pipeline is the exit status of the last command that ran in the pipeline
- When a script is run, it also produces an exit status
- An exit status of **successful** is the default when a script ends by running out of commands to run
- You can force a script to exit immediately with the default status of **successful** by using the exit command in the script
- To set an unsuccessful exit, put a non-zero number on the exit command line e.g. exit 1
- Any time a command might fail and cause problems, the script should be doing something to recognize and deal with the possible failure

Conditional Command List

- To use exit status as the control over whether to continue running commands in a list, insert a conditional operator between the commands
- Putting the && operator between two commands on one line creates a command list that only runs the second command if the first command succeeds
- Putting the || operator between two commands on one line creates a command list that only runs the second command if the first command fails
- Multiple conditional operators in a command list works, but may need parentheses to specify which command's exit status is used to control which subsequent command(s) in the list making it hard to read and debug - this is not commonly done for this reason
- To use both && and || on a command line, put the && first but consider using an if command instead for readability

```
cd /flooble || exit 1
grep -q dennis /etc/passwd && echo "User dennis is in the passwd file"
ps -eo user | grep -q dennis || echo "User dennis has no processes running"
sudo find / -user dennis -ls || echo "User dennis owns no files on this system"
```

Test Command

- The `test` command evaluates an expression and sets its exit status based on whether the expression evaluates as `true` or `false`
- The exit status of the `test` command can be used to control whether other commands run, in effect running commands based on the result of the `test`

```
test -d ~ && echo "I have a home directory"
test -f myfile || echo "myfile is missing"
test -d ~/Downloads || (mkdir ~/Downloads && echo "Made Downloads dir")
```

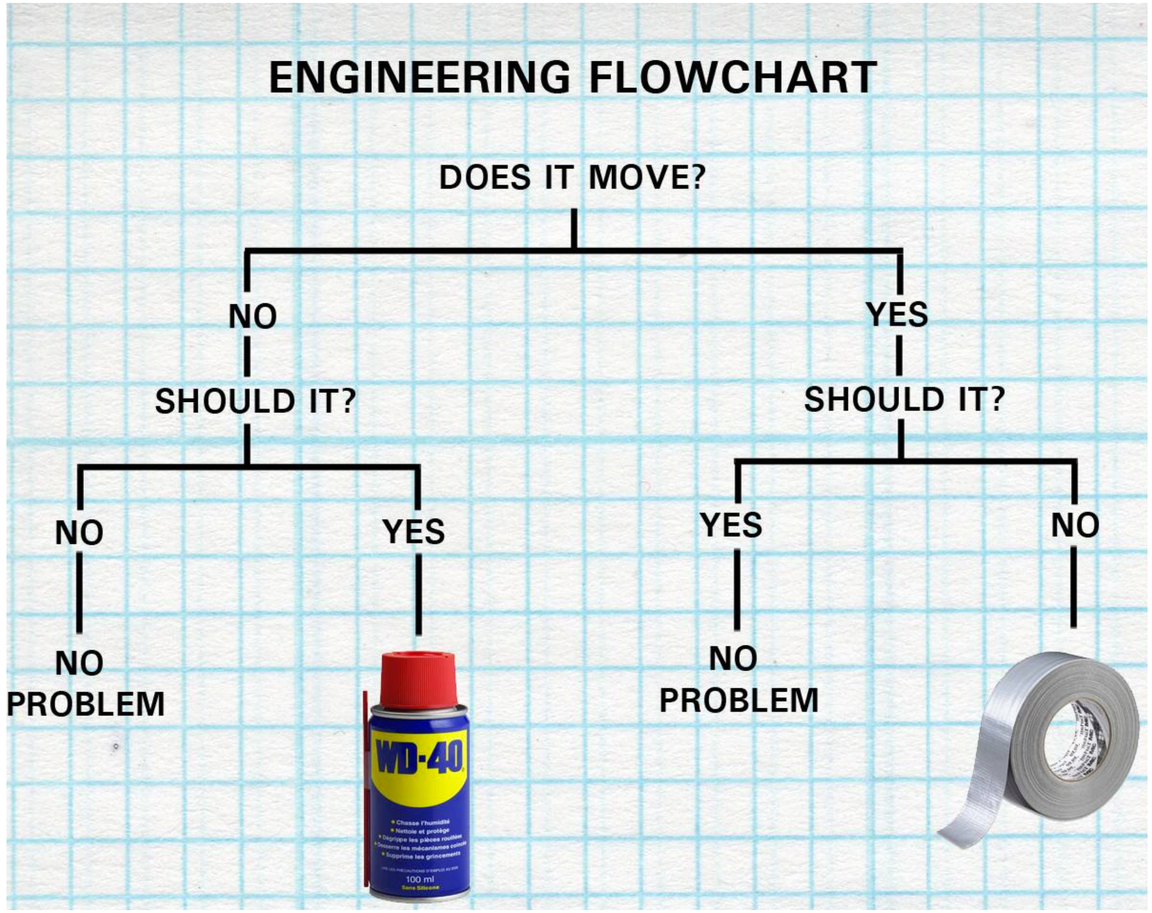
Long command lines

- Conditional execution operators and pipelines can make command lines get quite long
- These are easier to read and debug if each command is separated out onto its own line
- When they are they last character on a line, most operators will cause bash to continue to the next line as a single command list, semicolon is a notable exception
- When using continuation lines like this, it is good practice to indent the continuation lines to make it clear to the reader that they are continuation lines

```
mkdir foo &&
   echo "Made foo" ||
   echo "Failed to make foo"
```

```
echo -n "eno1 interface address:"
ip a s eno1 |
    grep -w inet |
    awk '{print $2}'
```

Test Expressions



- Testing data
- Testing files

Test Command

- The test command evaluates an expression and sets its exit status based on whether the expression evaluates as true or false
- The test command can also be run using the [alias, which uses a] to mark the end of the expression on the command line
- Multiple expressions can be evaluated by putting -a (and) or -o (or) between expressions

```
test -d /etc && echo "/etc exists"
[ -d /etc ] && echo "/etc exists"
[ -d /etc -a -r /etc ] && echo "/etc exists and is readable"
```

File Testing

- The following are commonly used file tests, although there are more not included here
- -e filename : filename exists as any kind of filesystem object
- -f filename : filename exists and is a regular file (can hold data)
- -d filename : filename exists and is a directory
- -r filename : filename exists and is readable by whoever is doing the test
- -w filename : filename exists and is writable by whoever is doing the test
- -h filename : filename exists and is a symbolic link
- -s filename : filename exists and is not empty
- Putting ! in front of the test operator (the letter with a dash in front of it) inverts the test

Test Expressions

- Test is generally used to either test data (usually in variables), or to test file attributes
- Test expressions may test single things for some characteristic, or attribute - this is known as a unary test
- Test expressions may compare two things - this is known as a binary test
- Unary test expressions take the form -x thing, where -x is the test operator specifying what kind of test to perform - characteristics testing
- Binary test expressions take the form thing1 operator thing2 where operator tells the test command what kind of comparison to perform - comparison testing
- Putting ! in front of the test operator inverts the test

Unary Operators

- The only unary operator that checks a variable is -v variablename which is true if the variable exists and false if it doesn't
- Text strings can be tested to see if they have no text in them with -z "sometext" or have some text with -n "sometext"
- Since it would make no sense to do such a test on literal text, use some kind of dynamic data with the -n or -z tests

Binary Operators For Text

- Text strings can be compared using the following operators
- "string1" = "string2" is true if the two strings are identical
- "string1" != "string2" is true if the two strings are not identical
- Strings consisting of digits are compared as text by these operators, not as numbers

Binary Operators For Numbers

- To compare numbers, there are several binary test operators available
- X -eq Y is true if X and Y are the same number
- X -ne Y is true if X and Y are not the same number
- X -lt Y is true if X is numerically less than Y
- X -gt Y is true if X is numerically more than Y
- X -le Y is true if X is numerically less than Y or X is equal to Y
- X -ge Y is true if X is numerically more than Y or X is equal to Y

Testing Command Success

- When a child process exits, the shell can retrieve the child's exit status from the special variable ?
- It can be used in a test expression when testing whether the immediately preceding command failed
- An exit status of zero means success
- Scripts can set their exit status with the exit command e.g. exit 3

```
grep -q '^dennis:' /etc/passwd
if [ $? -ne 0 ]; then
  echo "Adding user"
  sudo adduser dennis
else
  echo "user already exists"
fi
```

Conditional Script Blocks

- Running multiple commands based on a test - using if

Bash Execution Control

- Scripts commonly can evaluate situations and make simple decisions about actions to take
- Simple evaluations and actions can be accomplished using && and ||
- Complex evaluations and multi-step actions are better handled using more sophisticated execution control commands

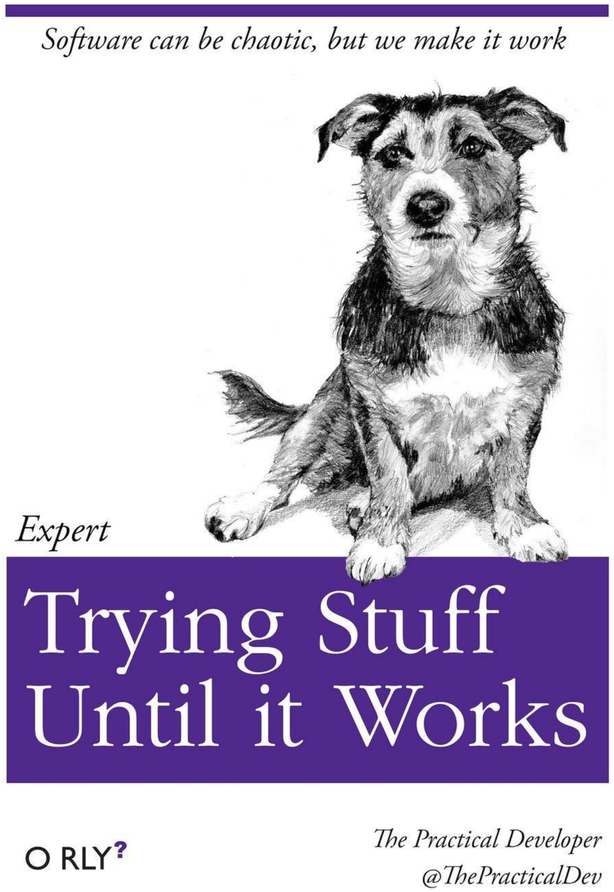
Conditional Script Block Execution

- A list of action commands can be run, or not run, based on the success or failure of a testing command list
- The test command can evaluate expressions, so it is the most common command for the testlist
- An arbitrary number of testlist/actionlist elifs can be used to take one of several actions based on multiple tests
- A default actionlist to run if no testlists are successful can be included by using else

```
if [ expr ]; then
    actionlist
fi
```

```
if testlist; then
    actionlist
elif testlist; then
    actionlist
else
    actionlist
fi
```

Controlling Execution



- data and command testing
- script blocks
- unary and binary operators
- third challenge script