# Test automation

- Whenever we alter our code, we should re-test
- Our collections of test cases can be substantial, and each test case can be large: manual testing is slow, error-prone
- Ideally, we would like a tool or script that lets us pick a set of test cases and it runs them, checks the results, and reports
- Continuous testing takes this further, and automates the process so that whenever we commit a change it automatically invokes testing/reporting

# Test data

- Each test case might involve a variety of data, such as:
  - a name or identifier for the case, and text description
  - the component(s) under test
  - input files
  - user input
  - command line arguments or parameters
  - expected output (stderr and stdout)
  - expected side effects (files/directories changed/new content)
  - expected return value/exit status
- Our test tool or script should be configurable, knowing how/where to look for all the relevant parts of a test case

# For each test case...

- The test script/tool should obtain all the input information,
- It should run the test case on the specified components
- It should capture all the output, side effects, and return value/status
- It should compare the results to those expected
- It should generate some form of summary/report on the test case result (with a configurable level of reporting detail)

# Tools and scripts

- Many different test tools exist, at a variety of price points and with a variety of features
- We can write our own test scripts/tools in basically any programming language
- We’re going to develop some basic test scripts in bash (and eventually apply them to suites of test cases we develop for the course project)

# Configurable test sets

- When designing our test system, we'd like it to be easy to add, remove, or alter test cases, and to specify sets of test cases to run
- One idea is to put collections of test cases in directories, and to specify which tests to run we tell the test script which directory to use (it then runs all the tests in that directory)
- We also need to come up with file formats to tell the test script all the relevant info for a test case (what input data to use, what results to expect, etc)

# Simple example

- Suppose a program gets a filename as a command line argument, some input data from the user as it runs, and writes results to the specified file, with a program exit status of 0 if ok and 1 if processing fails
- Each test case file might contain: a name/description of the test case, a filename to use as a command line argument, the “user” input data to use, the expected exit status, and the expected file content after running
- It would run the program, passing the command line argument and using I/O redirection to feed the “user input” to stdin, capture the exit status, capture the resulting file content, and compare them to expectations

# Example cont.

- We create a bunch of directories, each containing a set of test case files (e.g. one directory for simple valid cases, another directory for more complex valid cases, another directory for cases with exit status 1, etc)
- When we run our testing script, we pass it one or more directory names, and it runs every test case from each of the directories we mentioned
- For each test case the script prints its name and either passed or failed, and at the end tells us how many cases failed in all