Class Development Environment

The following has been tested on Ubuntu, but should work with other systems as well. For people who want to use the school computing facilities, a compute server has been built for the course: cs4472a.gaul.csd.uwo.ca . There, current versions of gcc and chrome have been built. A version of ruby is also there, but you would still need to do the rbenv and ruby-build installations discussed in item 2 and below. Note: although the professor is interested in any problems with the computing facilities that impact the course, he can not generally do anything about them directly – so be sure to report any problems with the department computing facilities to the systems staff at https://help.sci.uwo.ca/servicedesk/customer/portal/2/user/login?destination=portal%2F2 .

1. I am running version 8.1 of gcc. This can be installed on ubuntu using apt-get. For details see https://askubuntu.com/questions/1039244/how-do-i-use-the-latest-gcc-on-ubuntu-ubuntu-18-04-gcc-8-1?noredirect=1 . To verify the version you are using type **gcc --version** .
2. As noted in https://www.ruby-lang.org/en/documentation/installation/ , apt-get installs Ruby at version 2.3 . Unfortunately, as noted in https://en.wikipedia.org/wiki/Ruby_(programming_language) , 2.5.1 is the current stable release version (2.6 is in preview status https://www.ruby-lang.org/en/news/2018/05/31/ruby-2-6-0-preview2-released/). However, Ruby developers often keep multiple versions of Ruby around for compatibility checking. And, of course, there are multiple software packages to help manage multiple versions of Ruby. I use rbenv https://www.ruby-lang.org/en/documentation/installation/#rbenv and the related plugin https://www.ruby-lang.org/en/documentation/installation/#ruby-build .
3. rbenv is available from github https://github.com/rbenv/rbenv . **rbenv --version** on my system reports **rbenv 1.1.1-37-g1c772d5**
    1. following the instructions in https://github.com/rbenv/rbenv , we use **rbenv init** to set up the shell.
    2. there should be installed in a subdirectory of your home directory named **.rbenv**
4. ruby-build is available from github as well https://github.com/rbenv/ruby-build . **ruby-build --version** on my system reports **ruby-build 20180601-6-ge73041e**
5. in order for rbenv to do things right, you need to add three directories to the front of your path variable (so they override your default system paths):
    1. the absolute path to **.rbenv/bin**
    2. the absolute path to **.rbenv/plugins/ruby-build/bin**
    3. the absolute path to **.rbenv/shims**
        1. note: the shims directory is where rbenv keeps symbolic links to your `current' version of ruby managed by https://github.com/rbenv/rbenv#rbenv-global
6. **rbenv install 2.5.0-dev** will then build the latest version of 2.5 for you. **ruby --version** on my system reports **ruby 2.5.1p59 (2018-03-31 revision 63049) [x86_64-linux**] .
    1. note: after the install, you still need to do **rbenv global 2.5.0-dev** to tell **rbenv** to set things up so that when you type **ruby** or **irb** you get this version.
7. you should also note a program called gem is in the shim directory along with the rest of Ruby 2.5.1 . In my case, the version of gem is 2.7.6 . rbenv manages ruby versions, ruby-build installs ruby versions, and gem manages ruby software packages. https://guides.rubygems.org/
8. following the instructions in https://github.com/rbenv/rbenv , the next thing to install is bundler by doing **gem install bundler** , which installs bundler 1.16.3 in the shims directory . bundler https://bundler.io/ makes sure the right gems (specified in a file called **Gemfile**) are available for the Ruby application that is being worked on.
9. for unit testing ruby code, we will use rspec . following https://github.com/rspec/rspec , rspec is installed by **gem install rspec** , which gives us version 3.8 of rspec in the shims directory . rspec is a very common testing tool and is used as an example in the rbenv documentation to explain the shims directory https://github.com/rbenv/rbenv/wiki/Understanding-binstubs .
10. while people do ship code that has not been tested, this course is focussed on how to test stuff. one of the first goals that people have with testing is to be sure every line of code gets tested. for Ruby applications, there is a gem called simplecov https://github.com/colszowka/simplecov that will check this. however, one quickly realizes that lines of code can be complicated and justify multiple tests.
11. one approach to getting a finer level of detail than line level testing is mutation testing https://en.wikipedia.org/wiki/Mutation_testing . a useful gem that supports this approach is mutant, which is installed via: **gem install mutant-rspec** , which installs version 0.8.15 in the shims directory.
    1. Note when mutant installed, it installed rspec 3.7 instead of 3.8. the workaround is reported in https://github.com/mbj/mutant/issues/736 .
    2. also note the complaint during install:

```
warning: parser/current is loading parser/ruby25, which
recognizes
warning: 2.5.0-compliant syntax, but you are running 2.5.1.
warning: please see https://github.com/whitequark
/parser#compatibility-with-ruby-mri.
```

12. mutation testing is quite useful for unit tests, but become harder to use at the full application level. one reason is that running a single method is much faster than running a full application. another reason is that mutation testing works by trying to break your code and then see if your tests catch the break, but sometimes the break doesn't actually change the behavior of the code (the equivalent mutant problem) and so there is no reason for your tests to catch it. unfortunately, this can only be told by studying the code and determining if the break is real or not. the larger the code, the harder it is to figure this out – and the full application is usually much larger than its component units.
13. An alternative to mutation testing is random testing. The problem with random testing is when you send random stuff as input to some code, how do you know if the output is correct. One way to address this is to view the code as having certain properties that must hold

at various points in the execution and then insert code to check to see if these properties hold. Sometimes this idea is called `assertions' and sometimes `contracts'. There is a ruby gem called Contracts that aids this. It would be indicated in an application's Gemfile and is described in Use cases for code contracts (talk) and Designing by Contract: Using Types to Write Safer Code – Thomas Reynolds (talk) .

14. full application testing is important for two reasons: 1) the parts of an application might work in isolation, but not together and 2) it is at the application that the client specifies what they want. thus it is useful to have a way to do testing that can be read by non-programmers. for Ruby applications, this generally done using a tool called cucumber. installing cucumber **gem install cucumber** installs version 3.1.2 in the shims directory. see https://docs.cucumber.io/ .
15. sometimes an application runs in a web browser, which complicates testing. in that case, I tend to use selenium with the chrome browser, see Testing a Web App with Plain Ruby -- irb and Selenium (note) .
    1. a useful api for working with web applications in ruby is capybara https://en.wikipedia.org/wiki/Capybara_(software) and https://github.com/teamcapybara/capybara . note the application pulls this in with the Gemfile . capybara works with both rspec and cucumber, see Rails Conf 2013 BDD and Acceptance Testing with RSpec & Capybara (talk)
16. in addition to testing, another approach to maintaining software quality is code review https://en.wikipedia.org/wiki/Code_review . some tools that automate part of the code review process are:
    1. reek https://github.com/troessner/reek , **gem install reek** gives me version 5.0.2
    2. rubocop https://github.com/rubocop-hq/rubocop , **gem install rubocop** gives me version 0.58.2