CART User Guide

1.0 Overview

CART (Complete Automated Regression Tester) is an independent project from VCCK. CART is used to test VCCK.

The official backup for CART source code is Git Hub. (@ https://github.com/victortimothyshulist/cart). It is on Github unencrypted.

VCCK however, is backed up on Google Drive and IS encrypted.

Since CART is its own project but it is required to exist inside VCCK home directory, copy cart.py from its own directory into VCCK directory. (Make sure that ./vcck/cart.py is updated and matches /cart/cart.py).

cp /home/victor/cart/cart.py /home/victor/vcck/

2.0 How it works

VCCK can be invoked and executed in two modes:

- Mode 1 : NOT running under CART
- Mode 2: running under CART.

In Mode 1, you run vcck.py by just calling it as:

```
cd vcck
cd ./vcck.py
```

When it is run this way, TESTING is set to False. When it is False, vcck.py reads input from the console (keyboard) and responds to the screen. But in Mode 2, CART creates a temporary version of vcck, and calls it. Then TESTING is True. When true, vcck.py does not read from keyboard input, but from a file (a file that CART creates).

For a given version of VCCK, say version 1.000, you create a configuration file.

As an example, for version 1.000, you would create a file in...

./testpackages/v1.000.conf

This file contains the list of input strings to process. It also contains the names of the compressed tar ball files that contain 'state' to process each input string under. The 'state' is the directories and files for VCCK to operate under, for each input. These directories for example are...

./conversation-history/

./sessions/

./tls_csv/

./interpretations/

./compiled-classes/

So that, if you test the processing of input string 'x' a year ago, with specific compiled-classes and conversation-history, you know that every time you run it in the future, it will run under the same circumstances ('state' – files in the above directories).

So, if you run CART for say v1.000, CART will decompress the files in

testpackages/files/v1.000/

and run each input listed in testpackages/v1.000.conf.

There are 2 modes you run CART in...

- Mode 1 : '**create** 'mode. In this mode you are **establishing** what the expected results are for processing each line in your *.conf file ('v1.000.conf' in this example). In mode 1, VCCK creates /home/victor/cart/testpackages/files/v1.000/v1.000.tar.gz. Let's say you just coded version 1.000, and all test cases pass, let's say this is June 01, 2020. In mode 1, testpackages/files/v1.000/v1.000.tar.gz is **created.** (in mode 2, it is not created, but compared to the results produced in results_cart_tests).
- Mode 2 : 'test' mode. In mode 1 your expected results file was created. In Mode 2, let's say you have just coded version 2.000. Well, before you even

start developing the test cases for version 2.000, you want to make sure you didn't break any v1.000 functionality. So, first you re-run the tests for v1.000. You do this in Mode 2 … where CART runs VCCK with all tests for v1.000 and the output goes into
/home/victor/vcck/results_cart_tests
Let's say you just coded version 2.000 and it is June 01, 2022. Well, you coded v1.000 two years ago !! SO, did the new code you added to vcck.py (which provides version 2.000 functionality), did that new code of v2.000 break the version 1.000 code? Well, the output produced in
/home/victor/vcck/results_cart_tests (from test results in 2022 ('just now')
on June 01, 2022, is compared with the expected results for v1.000 from 3 years ago (the file
/home/victor/cart/testpackages/files/v1.000/v1.000.tar.gz – test results from 3 years ago). CART decompresses that tar.gz file and compares it to results in
/home/victor/vcck/results_cart_tests
any mismatch at all in contents results in 'regression test failed' ! Error message appearing on screen.

3.0 Example of using cart for version 1.000 and 2.000 of VCCK

Let's say we just coded version 1.000. So, we develop the test cases, and then create the v1.000.conf file. You create **testpackages/v1.000.conf** let's say it has this for contents....

```
# CART test package file for version 1.000.
areas = red, green, blue

[state.1.tar.gz]

“input A”
“input B”

[state.2.tar.gz]

“my line X”
“my line Y”

[state.3.tar.gz]

"my line A”
“my line B”
```

After creating all your 'state' files/folders (any state files that VCCK depends on, example, ./conversation-history/, ./sessions/,

./tls_csv/, ./interpretations/ and ./compiled-classes/) you would figure out the inputs for vcck to process while it is in that 'directory state'. If all tests pass, then create a tar.gz file for all those state files, and call it (in this example, "state.1.tar.gz").

So state.1.tar.gz will contains all 'state' files/folders (./interpretations/* ,etc).

Then put that file (state.1.tar.gz) in ./testpackages/files/v1.000/ directory.

Now, CART (in this example) will have VCCK process the LUI, for each input (in this example ("input A" and "input B").

VCCK's output (from '**cartlog**()' output, discussed later), will be produced in **results_cart_tests** (and, if in '**create**' mode, ./cart/testpackages/files/v1.000/v1.000.tar.gz).

If all tests pass, testpackages/files/v1.000/v1.000.tar.gz will not ever change (and it will be used to compare to, years in future). So we ran in 'create' mode and that tar.gz file was created, say it's the year 2020.

To have testpackages/files/v1.000/v1.000.tar.gz created, run this...

./cart.py v1.000 create

after running that, testpackages/files/v1.000/v1.000.tar.gz is created.

Now, years later, say you're on version 15.0 of VCCK. Well, we want to re-test all previous versions (under the exact same directory state as we did years ago).

Now, we run CART not in 'create' mode, but 'test' mode.
In '**test**' mode ./testpackages/files/v1.000/v1.000.tar.gz is **NOT** created, but it is assumed to already exist (and CART will error out if it does not exist).

In **test** mode, CART will run VCCK and the output ("now" - say we are in the year 2035). Now, the output from VCCK is produced in **results_cart_tests.** So, now, in test mode in year 2035 (test mode for v1.000), CART decompresses testpackages/files/v1.000/v1.000.tar.gz (that was made 15 years ago), with the latest output (that is in **results_cart_tests).**
Any differences result in 'regression test ' failed ! Message appearing.

The command to do this is...

./cart.py v1.000 test

so when CART is run as in the above, it does ****NOT**** create the testpackages/files/v1.000/v1.000.tar.gz file, but instead just compares the output (that VCCK produced) in results_cart_tests with the results that are stored in the tar.gz file (from 15 years ago,), in testpackages/files/v1.000/v1.000.tar.gz.

When CART is run in **'create' mode,** the file testpackages/files/v1.000/v1.000.tar.gz is created. In CREATE mode, the contents of that tar.gz file are identical to the non-compressed data in **results_cart_tests** directory so that you can easily look at the output without having to manually decompress the tar.gz. (They are the same because right after data is produced in **results_cart_tests,** CART creates v1.000.tar.gz from the contents in **results_cart_tests).**

But, in '**test**' mode, the data in **results_cart_tests** may or may not be the same as what is in v1.000.tar.gz. In test mode, v1.000.tar.gz is NOT created (but instead just consulted – in test mode, v1.000.tar.gz was created years ago), but **results_cart_tests** was produced 'just now' (the current test run). If the two compare perfectly, then regression test pass, if they differ at all even in slightest way, regression test is reported as failing.

calls to "cartlog()". The first argument to cartlog has to be a string. That first string argument MUST BE exactly, one of the "areas" given in your configuration file.
The second argument is also a string – and it is whatever test results you want to record.

Looking at the configuration file again, (for a specific version –one *.conf file per version).

[state.1.tar.gz]

"input A"
"input B"

[state.2.tar.gz]

"my line X"
"my line Y"

This tells CART.... (IF run with "./cart.py v1.000 create')....

1. Decompress the tar.gz in /home/victor/vcck/testpackages/files/v1.000/state.1.tar.gz into the root of ./vcck directory.

2. then, have VCCK execute the inputs

"input A" and then "input B" (set "lui" variable to each one of those, one at a time).

Then vcck.py calls "cartlog()" to report testing output (which ends up in results_cart_tests).

Then, CART cleans up the directory (gets rid of all files from decompressing state.1.tar.gz.

Then CART decompresses state.2.tar.gz, and then directs VCCK to process the "last user input" (lui) values of... "my line X" and "my line Y"

again, output going into results_cart_tests. (output goes into results_cart_tests when vcck.py calls cartlog() – discussed below.

Question --- How does output from VCCK make it into **results_cart_tests** ? (and, IF in '**create**' mode, into v1.000.tar.gz)?

Answer – by VCCK calling cartlog()

OK, so , how does VCCK call cartlog() ?

Question --- what is cartlog() for ?
Cartlog() --- is the means for VCCK to report the test output.

Question – how to call cartlog() ? (areas-- what is that?)

**** NO actual executable Python code for CART testing is ever actually put into vcck.py.**

The only thing that is done is insertions of comments into vcck.py.
Example...

#CART_INCLUDE_v1.000_file1.py

if you have that comment in your vcck.py source code, it would cause CART to (as part of its creation of a temporary version of vcck.py)... to...

look into (IF you were running for version 1.000)...

look into... testpackages/files/v1.000 to find a file "file1.py".

The above comment line would be replaced with all the lines in testpackages/files/v1.000/file1.py.

This is very clean – because the only change made to your actual VCCK source code is the addition of Python comments !!!

If you were testing another version, say v2.500, and you wanted to include testing file 'myfile.py', then you would create

testpackages/files/v2.500/myfile.py

then, at the appropriate place inside vcck.py, have a comment....

#CART_INCLUDE_v2.500_myfile.py

then, ONLY **IF** you ran CART with...

```
./cart.py   v2.500  create
```

then and only then would the comment line

```
#CART_INCLUDE_v2.500_myfile.py
```

be replaced with all lines from

```
testpackages/files/v2.500/myfile.py
```

if you ran for another version (example "**./cart.py v3.500 create**"), then it would leave the comment line

```
#CART_INCLUDE_v2.500_myfile.py
```

alone.

So, in the above, the file "myfile.py" is called a "CART Include File".

What does a CART Include File contain?

It contains calls to "cartlog()".

Let's say you wanted to produce the results of doing a "pick highest" functionality. So you want VCCK to report the results of it doing that. So, first you'd create a file, maybe called "pickh.py". (short for pick highest). Maybe there's a python function "def pick_highest" that we are testing.

So, inside vcck.py, right after calling the function, we put a include....

```
pick_highest()
```

and then add your comment...

```
pick_highest()
#CART_INCLUDE_v5.690_pickh.py
```

this example says we are testing version 5.690 of VCCK. (so in order for this include to actually be replaced by contents of pickh.py file, you'd have to run "./cart.py v5.690 create").

Now, create the file

```
./testpackages/files/v5.690/pickh.py
```

now, the contents of that file would be, for example...

```
for thevalue in somelist:
    cartlog(pickhighest, thevalue)
```

So we are saving all the values in the Python list variable "somelist" to the CART results directory,

in a file called " pickhighest".

However, in order for this to work, you have to Define the "area" inside

./testpackages/v5.690.conf.

Inside the /testpackages/v5.690.conf you would have to have a line....

areas = pickhighest

(and whatever other areas you want, comma separated).

What are areas?? they are functionalities you want tested. The idea is to logically separate (into files), all results for one feature or perhaps one function, in the same file.

The two modes VCCK is called...

Mode 1:

CART calls vcck.py (well, it makes a copy of vcck.py and creates a temporary version of it (called ./vcck-cart-temp.py) that is same as vcck.py except with the CART include comments replaced by the actual cart include files (which live in testpackages/files/v1.000 directory – or whatever version).

>> Example calling from CART...

./vcck-cart-temp.py -T 0:temp_cart_input.txt

When called this way, VCCK only gets input to process (user input strings) from a file that CART dynamically creates for that test run. VCCK does NOT read in keyboard input for strings to process. This is when VCCK is run in "Testing mode"

Mode 2:

when called as in...

./vcck.py

Then VCCK reads strings to process from keyboard input only. NOT from a file. This is the "normal mode" , or "production mode".

5.0 Important Note about calling cartlog()

When calling cartlog()... do **NOT** write the output like this....

```
For  x in  var_1:
       cartlog(somearea,  "x = " + x)
       for y in var_2[x]:
            cartlog(somearea, "\t" + "y = " + y)
```

Instead, have **EVERY LINE** COMPLETELY qualified....

```
For  x in  var_1:
       for y in var_2[x]:
            cartlog(somearea, "x = " + x + ", y = " + y)
```

This is so that, after CART sorts all output files in results_cart_tests, that the diff will work.

So again, we will **NOT** require that you sort before calling cartlog() --- nothing in your testpackages NOR vcck.py should care about sorting.. CART will do all sorting before comparing last run with previous expected results file.

Calls to cartlog() should, EVERY TIME, have ALL relevant values of variables that pertain to that test, right from the "main call" (from code that is in "main" – that is, outside any other function). Every call to cartlog() should include ALL relevant variable values and never exclude the value of any parent variables. The only exceptions are: (1) the directory state database ID and (2) index into "LUI" number... in other words, the 2 directories right under results_cart_tests.

The results_cart_tests directory

If you had the following in your *.conf file. (testpackage configuration file)... (only one area defined but can be more, comma separated)...

areas = myarea_1

[state1.tar.gz]
"xxx"
"yyy"

[state2.tar.gz]
"jjj"
"kkk"

Then if you ran CART in **create** mode, you look inside our **results_cart_tests** directory...

first directory under **results_cart_tests** is the directory state ID, and second directory is the input number.....

results_cart_tests/0/0/myarea_1.res

this is data that is sent via “cartlog(' myarea_1', …..)

This is for when the directory state is as inside the state1.tar.gz file (after CART uncompresses this file into the root of vcck, then runs input # 0, “xxx”.

results_cart_tests/0/1/myarea_1.res

same as above but when “yyy” is processed (the “1” in “/0/1/” of the path... index 1... or 2nd input line (“yyy”)...

results_cart_tests/1/0/myarea_1.res

here the first directory under **results_cart_tests** is “1” .. which means index 1.. or 2nd directory state (state2.tar.gz)... and /0/ in path means index 0.. or 1st input line (“jjj”). So this is all the output of 'cartlog('myarea_1',.....)' when under that state of directories and files, and that input. (first.. or index 0.. 'jjj')

results_cart_tests/1/1/myarea_1.res

same as above... first '/1/' in path means under directory state given by contents of state2.tar.gz, and Second /1/ means input index 1 (== 2nd ---> “kkk”).

The 'dir_states' directory

When you run in either create or test mode... cart, before it clears out the directory, backs up all your VCCK directories in “dir_states”. It creates a tar.gz file of all your vcck state directories – time stamped, in dir_states. Example, if you created some files in tls_csv, and then run CART in test or create mode, it destroys tls_csv (and the other files) . ..**but, before it does that**.. it collects and tar ball's up them and puts them in dir_states.

That is, usually, when you run in test or create mode, CART clears the directory, and installs all files from the tar.gz files that are mentioned in your test configuration file. This has the effect of course of destroying the current directory contents. But, now, with latest version, it backs that up in “dir_states”.

Current mode

what if you just want to run VCCK from within CART but do not want to disturb the current directory ? Well, you run in “current” mode ! If you call cart.py with 2nd parameter of “current” (no quotes) then it runs in current mode. What happens in current mode?

1) CART ONLY looks at input lines to send to VCCK that are under “ [] “ in your configuration file.
2) CART does not touch the directories – (the VCCK directories – tls_csv, compiled-classes, etc),
3) CART sends all data from cartlog() to “current” inside results_cart_tests.

So, if you don't want to have your VCCK directories changed, you just want to run VCCK from within CART (so that you can use “include” files), but do not want to touch the state directories,

run cart in 'current' mode.

When run in “current” mode, CART only sends to VCCK, the lines that are under the “ [] “ indicator in your configuration file.

In your configuration file, if you had:

[state.1.tar.gz]
“aa”
“bb”

[]
“cc”
“dd”

then, if run in “current” mode (2nd argument is 'current'), then CART leaves all VCCK state directories alone and just calls VCCK (with include files expanded) ,and just sends “cc” and “dd” to it.

**warning about Current Mode*

When running in CURRENT mode ...

Make sure, that, if you have any logic that does something based on the value of the _CART_FILEDB variable, that you also include

if _CART_FILEDB has value 'current'

otherwise, you will get Different results when running under current mode for file state "x" than you will for running 'test' mode for SAME file state 'x' !!!

Let's say in the below that when _CART_FILEDB==7 we are using file state in "x1.tar.gz"

```
------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------
# just make sure the shared lists are 'new' .. to make sure LCC is calculated.

if (_CART_FILEDB == '4') or (_CART_FILEDB == '5') or (_CART_FILEDB == '6') or
(_CART_FILEDB == '7') or (_CART_FILEDB == 'current') :
    print("* Sleeping for 1.1 seconds - so that files are 'new' (shared-lists/*)")
    time.sleep(1.1)
    os.system("touch shared-lists/*")
    os.system("touch tls_csv/000000004/cvnavs.data")
------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------
```

Well, if we **did not** have

or (_CART_FILEDB == 'current')

included in the above, then we'd get different results between...

1) the run in "test" mode when _CART_FILEDB == '7'... using "x1.tar.gz"

2) the run in "current" mode ... using "x1.tar.gz"

that is, both above are using file state database "x1.tar.gz", but (1) and (2) would still give different results if you did NOT have the

or (_CART_FILEDB == 'current')

included.

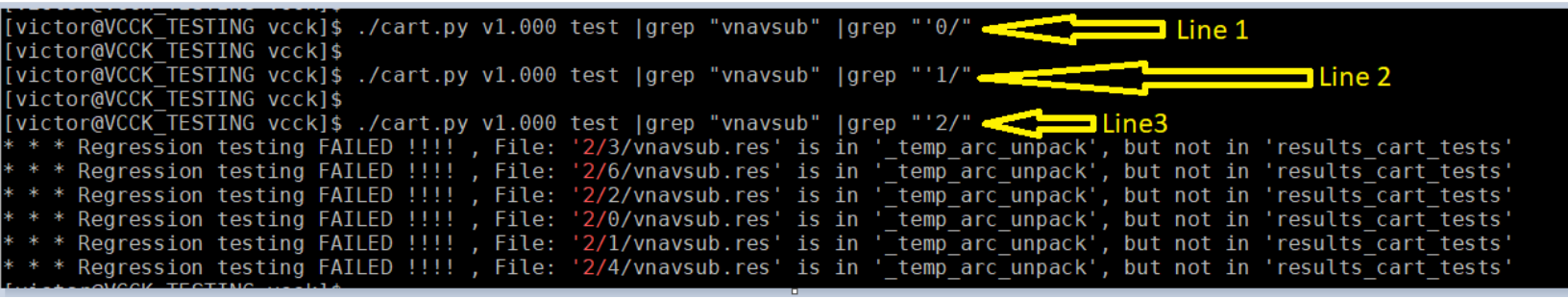
The line

if (_CART_FILEDB == '4') or (_CART_FILEDB == '5') or (_CART_FILEDB == '6') or (_CART_FILEDB == '7') or (_CART_FILEDB == 'current') :

is just an example, and of course is specific to whatever test case logic you are dealing with.

Checking which T.C. (Test cases) passed and which failed

Say if you wanted to run all test cases for version 1.000 and view all results for FILE STATE DATABASE index 0 (so the first File state database in v1.000.conf)...



So in the above...

Line "1" – we are viewing all results for failures for TEST AREA " vnavsub" (So all lines that call **cartlog**('**vnavsub**', '.....')).

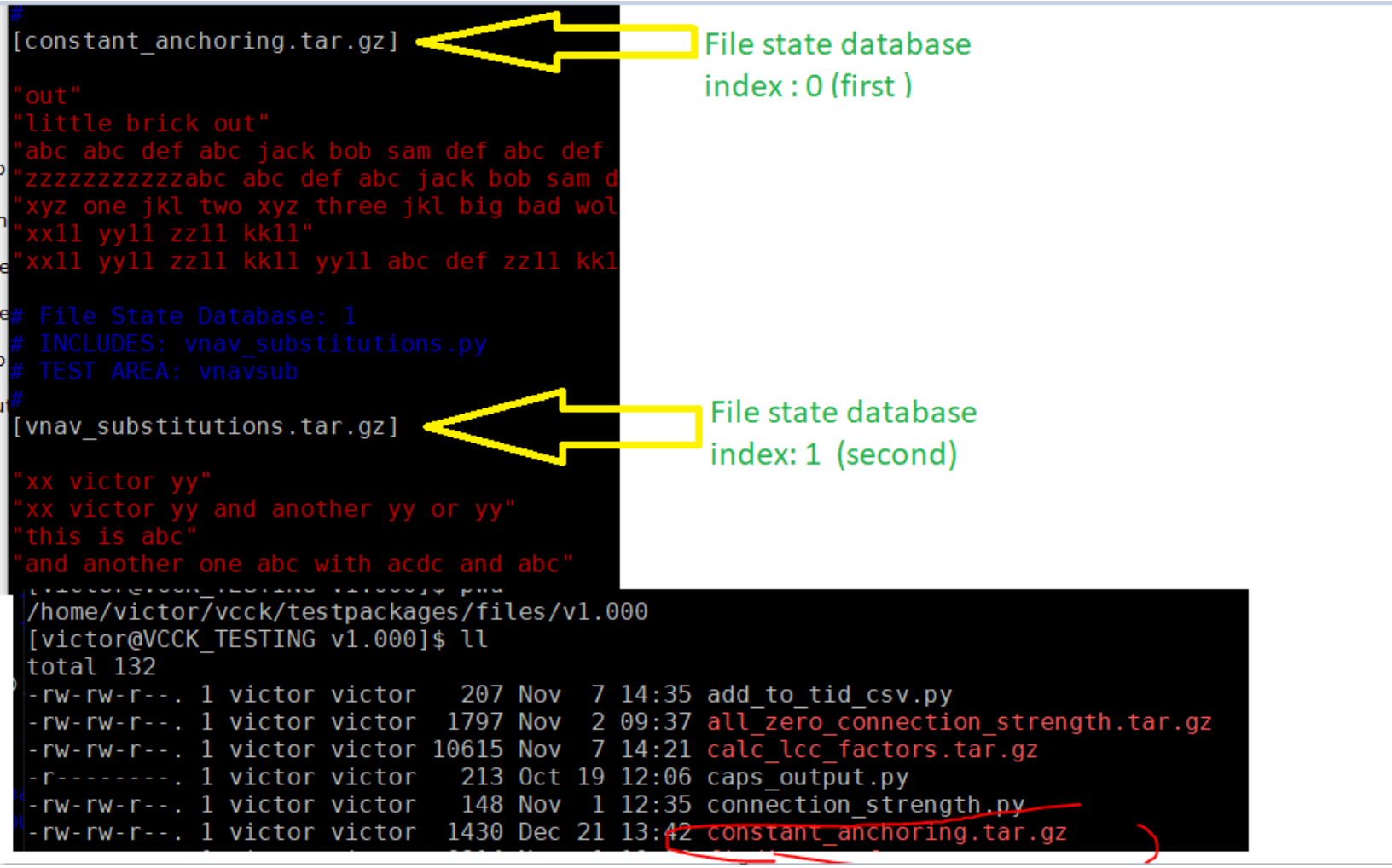
We see no output for Line 1 – so we know that all TC (test cases) for File state database index 0 (so the first file state gz file –first square bracketed filename in v1.000.conf) ALL PASSED.

We also see that File state database index 1 (line 2 .. grep "'1/")... so the second file state database also has all passed tests for test area vnavsub).

So for all inputs for file state database 1 and file state database 2 – for test area " vnavsub" ALL

PASS.

But there are FAILED test results for file state database index 2 (3rd file state database given in v1.000.conf).



File state database index 0 (first) means... system will unzip and untar "constant_anchoring.tar.gz" into ./vcck so that all files "come into effect" when testing the corresponding inputs ("out", "little brick out", etc).