北京大學
PEKING UNIVERSITY

瀚高
HIGH GO

测试

开源开发实践-第九周

David & Cary

AGENDA
目 录

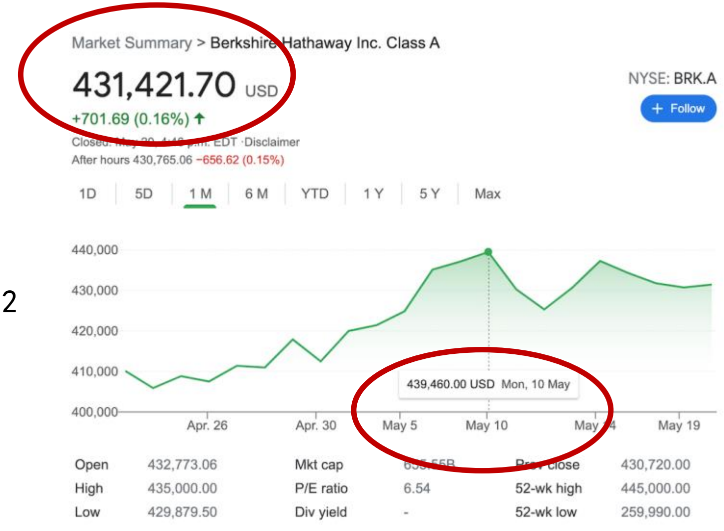
- Software Testing
- PG Regression Framework

Software Testing

Software Testing

The Motivation of Software Testing

- Risk management, Software failure has caused more than inconvenience.
  - ✓ Israel's first attempt to land an unmanned spacecraft on the moon failed on April 11, 2019 due to a software bug with its engine system.
  - ✓ Heartbleed, an OpenSSL vulnerability introduced in 2012 and disclosed in April 2014, removed confidentiality from affected services, causing among other things the shut down of the Canada Revenue Agency's public access to the online filing portion of its website following the theft of social insurance numbers
  - ✓ Nasdaq computers store stock prices as a 32-bit number representing the number of 1/100'ths of a penny, which means the highest dollar amount it can store is $429,496.7296 (2^32/10000).
- Cost management



Market Summary > Berkshire Hathaway Inc. Class A

431,421.70 USD

+701.69 (0.16%) ↑

Closed: May 20, 4:10 p.m. EDT ·Disclaimer

After hours 430,765.06 −656.62 (0.15%)

NYSE: BRK.A

439,460.00 USD Mon, 10 May

Open	432,773.06	Mkt cap	655.55B	Prev close	430,720.00
High	435,000.00	P/E ratio	6.54	52-wk high	445,000.00
Low	429,879.50	Div yield	-	52-wk low	259,990.00

https://en.wikipedia.org/wiki/List_of_software_bugs

Software Testing

What is Software Testing

- The process of devising a set of inputs to a given piece of software that will cause the software to exercise some portion of its code.

- The developer of the software can then check that the results produced by the software are in accord with his or her expectations.

Software Testing

Software Testing Objectives

- Find as many defects as possible.
- Find important problems fast.
- Assess perceived quality risks.
- Advise about perceived project risks.
- Advise about perceived quality.
- Certify to a given standard.
- Assess conformance to a specification (requirements, design, or product claims).

Software Testing

A Testing Cycle

- Requirements Analysis: Testing should begin in the requirements phase of the software life cycle.
- Design Analysis: During the design phase, testers work with developers in determining what aspects of a design are testable and under what parameter those testers work.
- Test Planning: Test Strategy, Test Plan, Test Bed creation.
- Test Development: Test Procedures, Test Scenarios, Test Cases, Test Scripts to use in testing software.
- Test Execution: Testers execute the software based on the plans and tests and report any errors found to the development team.
- Test Reporting: Once testing is completed, testers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.
- Retesting the Defects

Software Testing

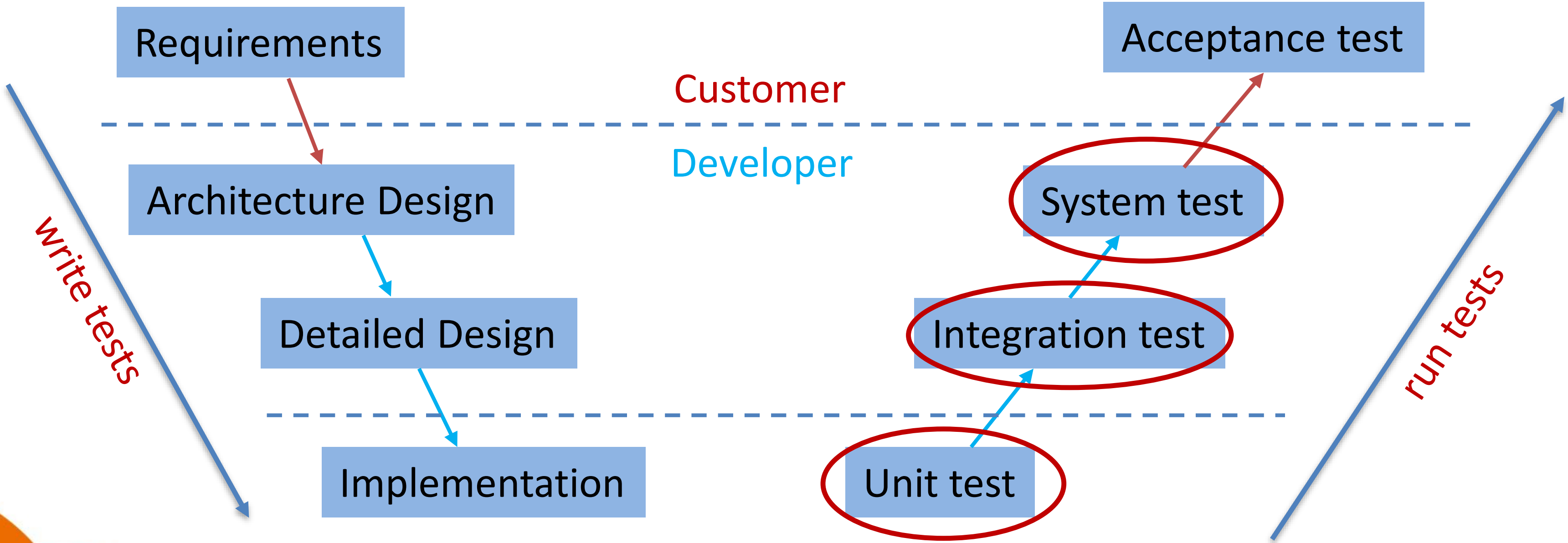
The basics of the testing

- Testing Strategies
  - ✓ Black-Box Testing
  - ✓ White-Box Testing
- Testing Stages
  - ✓ Unit Test
  - ✓ Integration Test
  - ✓ System Test
  - ✓ Acceptance Test
  - ✓ Regression Test

Software Testing

Testing Models

- A testing in the V-Model



Software Testing

Some Testing Terminology

- Error

  An error is a human mistake.

- Fault

  A fault is the representation of an error. It is also called a defect.

- Failure

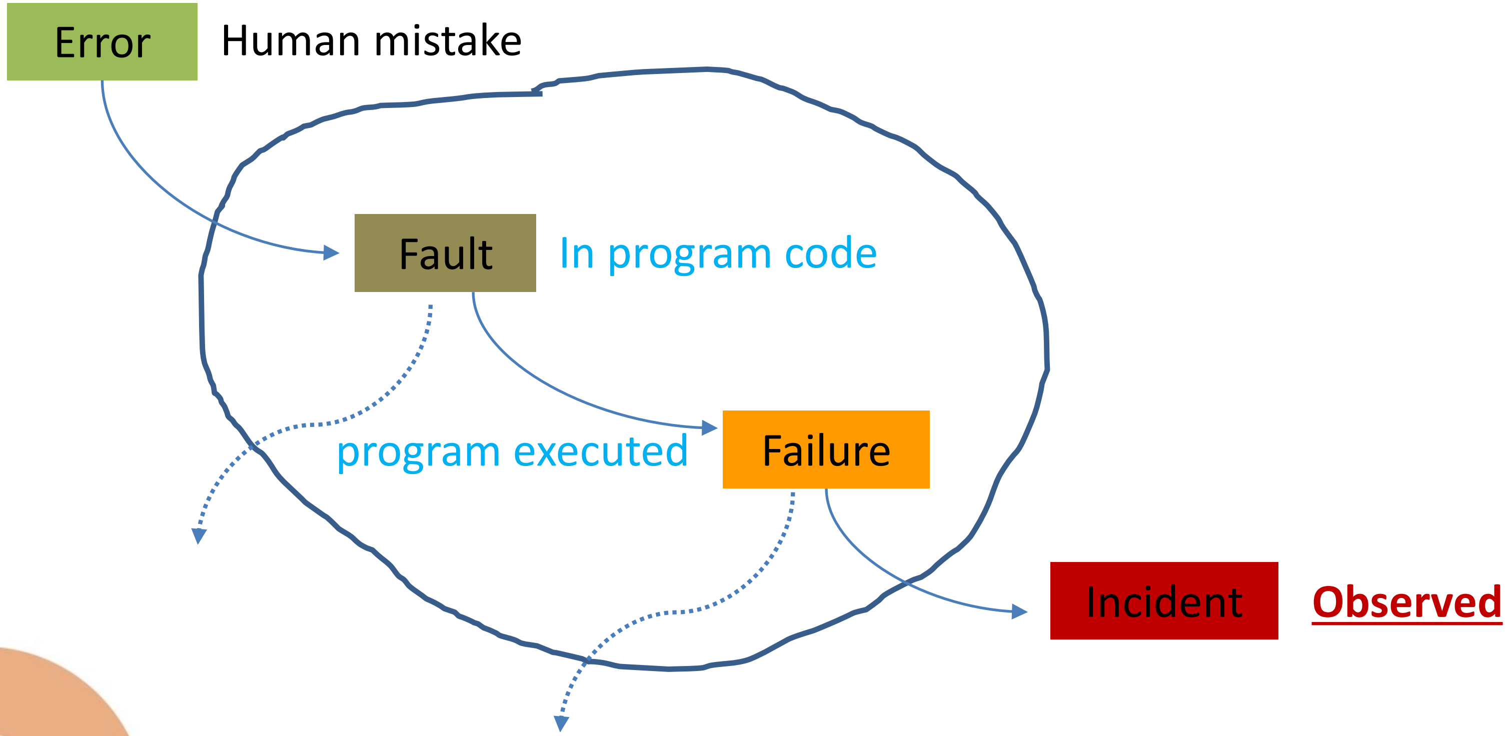
  A failure is what happen when a fault executes.

- Incident

  An incident is the symptom associated with a failure that alerts the user of the occurrence of the failure. It is observed by the user.

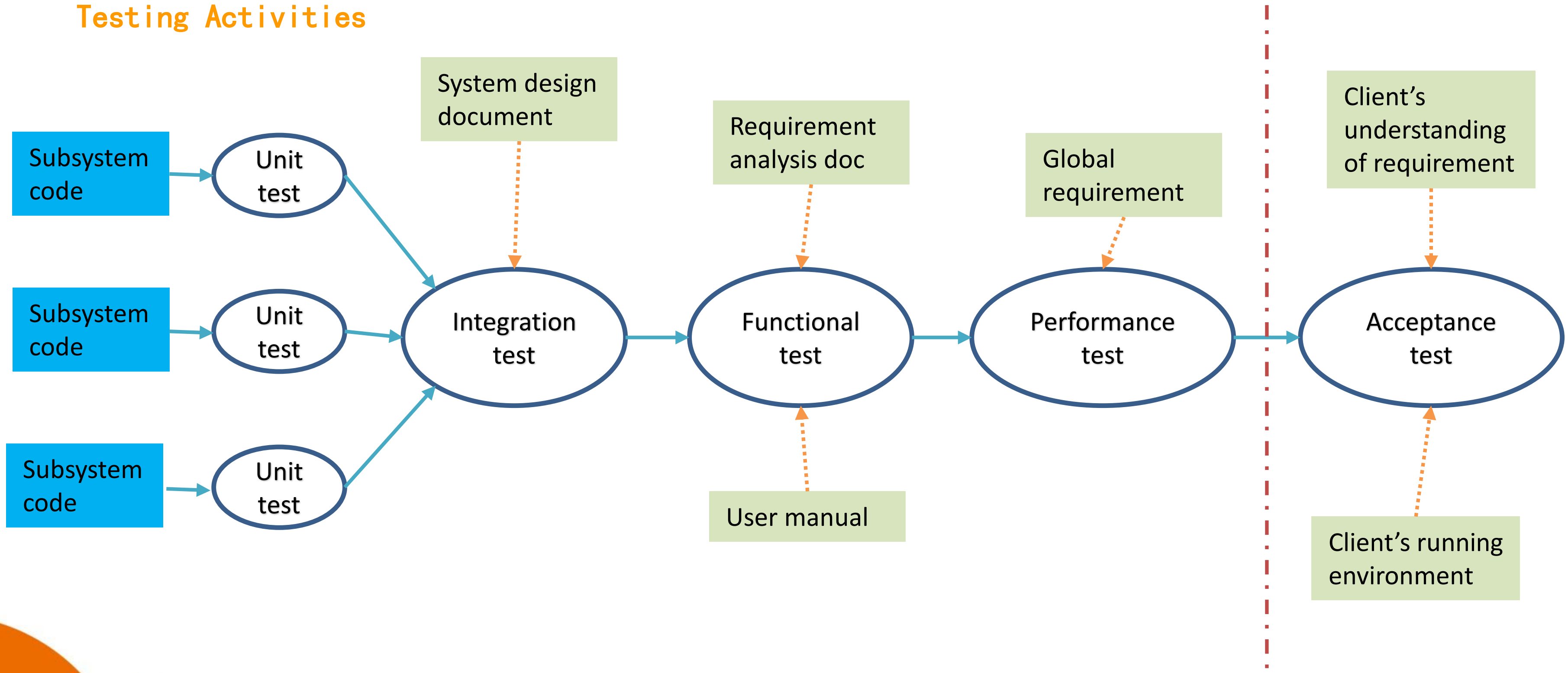
Software Testing

Some Testing Terminology



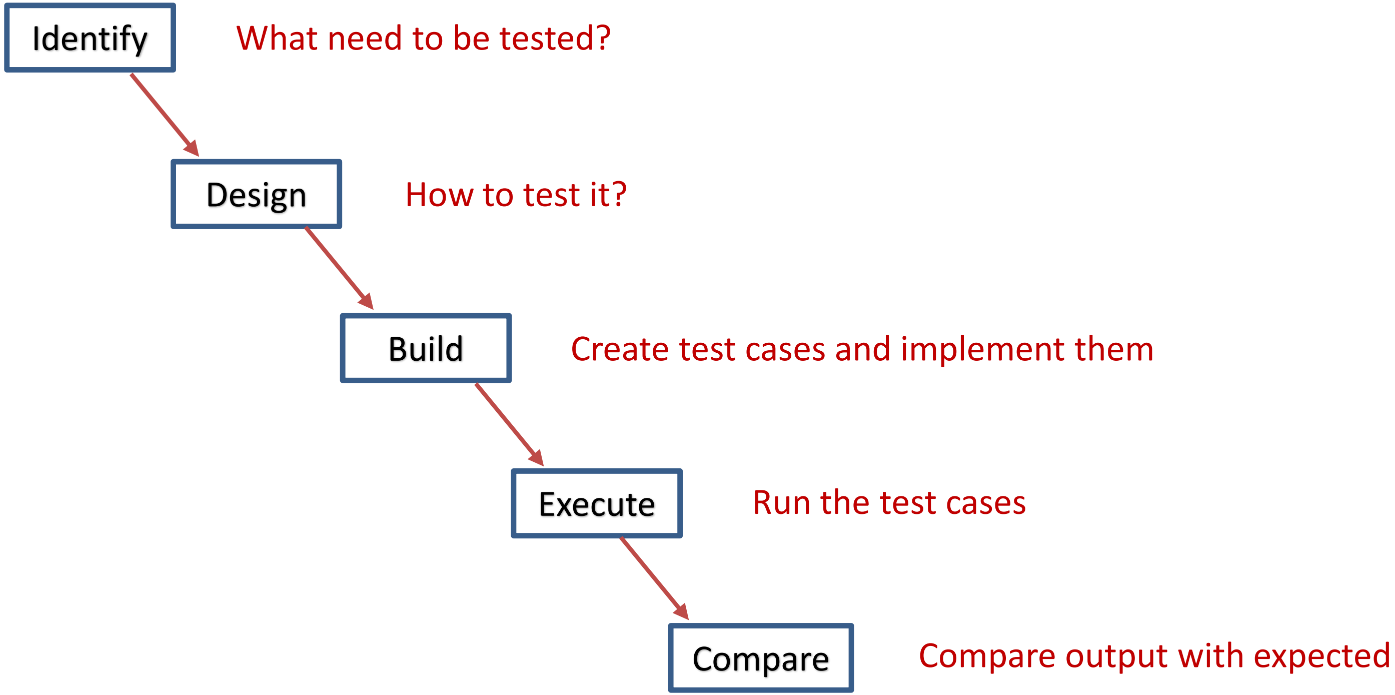
Software Testing

Testing Activities



Software Testing

Testing Activities



Identify — What need to be tested?

Design — How to test it?

Build — Create test cases and implement them

Execute — Run the test cases

Compare — Compare output with expected

Software Testing

Testing Activities: identify

- What need to be tested?
- Descriptions of circumstances that could be examined, such as event or item.
- Categories: functionality, performance, stress, robustness ...
- Derive
  - ✓ Using testing techniques
  - ✓ Refer to the V-Model

Software Testing

Testing Activities: design

- Design test cases
- Input values
- Expected outcomes
- Things created (output)
- Things changed/updated
- Things deleted
- Timing
- Environment prerequisites: file, network connection
- ...

Software Testing

Testing Activities: build

- Build test framework
- Create test cases
- Implement test case
- Set up the environment
- Prepare test scripts
- Use test automation tools

Software Testing

Testing Activities: execute

- Run test cases
- What screen data to capture
- When/where to read input and output
- Control information
  - ✓ Repeat a set of inputs
  - ✓ Make a decision based on output
- Testing concurrent activities

Software Testing

Testing Activities: compare

- Compare test outcomes with expected outcomes
- Simple/complex
- Different types of outcomes
  - ✓ Variable values (in memory)
  - ✓ Disk-based (textual, non-textual, database, binary)
  - ✓ Screen-based (char, GUI, images)
  - ✓ Others (multimedia, communicating apps.)
- Compare: actual output vs. expected output
  - Yes ➔ **Pass** (assumption: Test case was "instrumented.")
  - No ➔ **Fail** (assuming that there is no error in test case, preconditions)

Software Testing

Black-box Testing

- Focus on the input and output behavior. If for any given input, we can predict the output, then the module passes the test.
- Almost always impossible to generate all possible inputs ("test cases")
- Goal: Reduce number of test cases by equivalence partitioning:
  - ✓ Divide input conditions into equivalence classes
  - ✓ Choose test cases for each equivalence class. (Example: If an object is supposed to accept a negative number, testing one negative number is enough)

Software Testing

White-box Testing

- Statement Testing: Test single statements
- Loop Testing:
  - ✓ Cause execution of the loop to be skipped completely
  - ✓ Loop to be executed exactly once
  - ✓ Loop to be executed more than once
- Path testing:
  - ✓ Make sure all paths in the program are executed
- Branch Testing （Conditional Testing）: Make sure that each possible outcome from a condition is tested at least once

Software Testing

White-box Testing

```
void BootStrapXLOG(void)
{
    … …
    /* Now create pg_control */
        InitControlFile(sysidentifier);
        ControlFile->time = checkPoint.time;
        ControlFile->checkPoint = checkPoint.redo;
        ControlFile->checkPointCopy = checkPoint;

    /* some additional ControlFile fields .. */
    WriteControlFile();

    /* Enable key manager if required */
    if (ControlFile->key_management_version > 0)
        BootStrapKmgr();

    /* Bootstrap the commit log, too */
    BootStrapCLOG();
    … …
}
```

- Test Case 1:
  ControlFile->key_management_version > 0

- Test Case 2:
  ControlFile->key_management_version = 0

- Test Case 3:
  ControlFile->key_management_version < 0

Software Testing

Completion Criteria

- When are we done testing? This is still a research topic.

1. One view: testing is never done, the burden simply shifts from the developer to the customer
2. Testing is done when you run out of time or money
3. Use a statistical model:
   - ✓ Assume that errors decay logarithmically with testing time
   - ✓ Measure the number of errors in a unit period
   - ✓ Fit these measurements to a logarithmic curve
   - ✓ Can then say:
     "with our experimentally valid statistical model we have done sufficient testing to say that with 95% confidence the probability of 1000 CPU hours of failure free operation is at least 0.995"

Software Testing

Testing Productivity

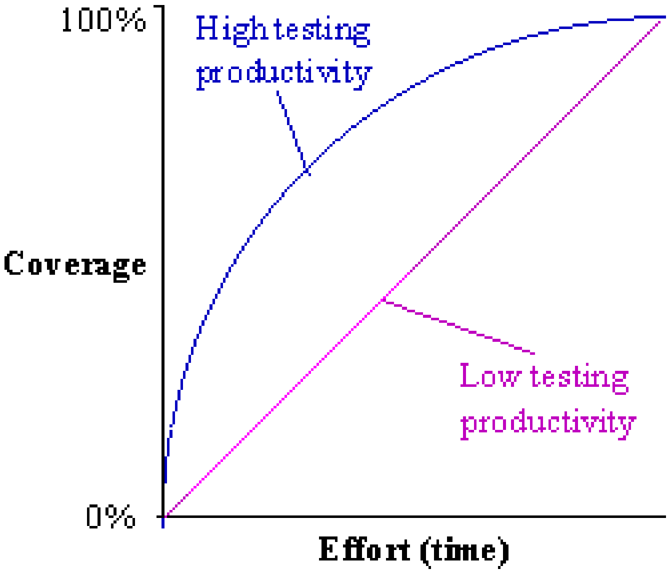


Figure 1: Coverage rate

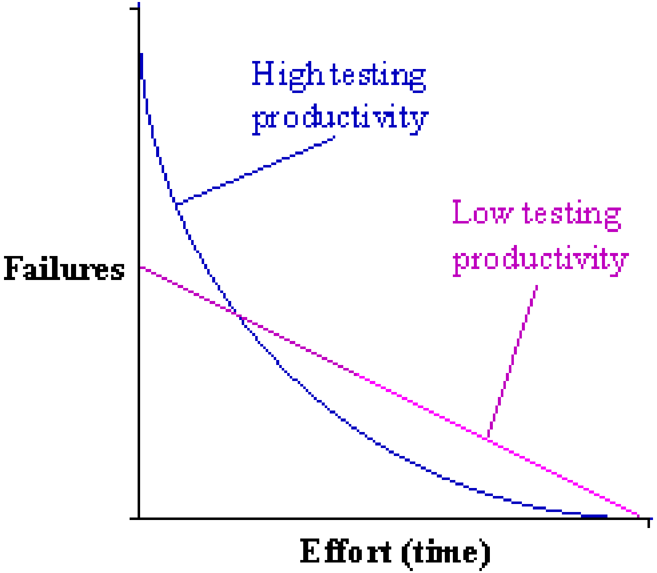


Figure 2: Failure discovery rate

http://www.bullseye.com/coverage.html

Software Testing

Summary

- Testing is an important part of the Software Lifecycle
- Highly technical and challenging
- It is affected by the selected process
- Quality Assurance is paramount both for mission critical and non-critical systems
- Software Evolution aims to keep systems operational when environment changes occur

Software Testing

Reference

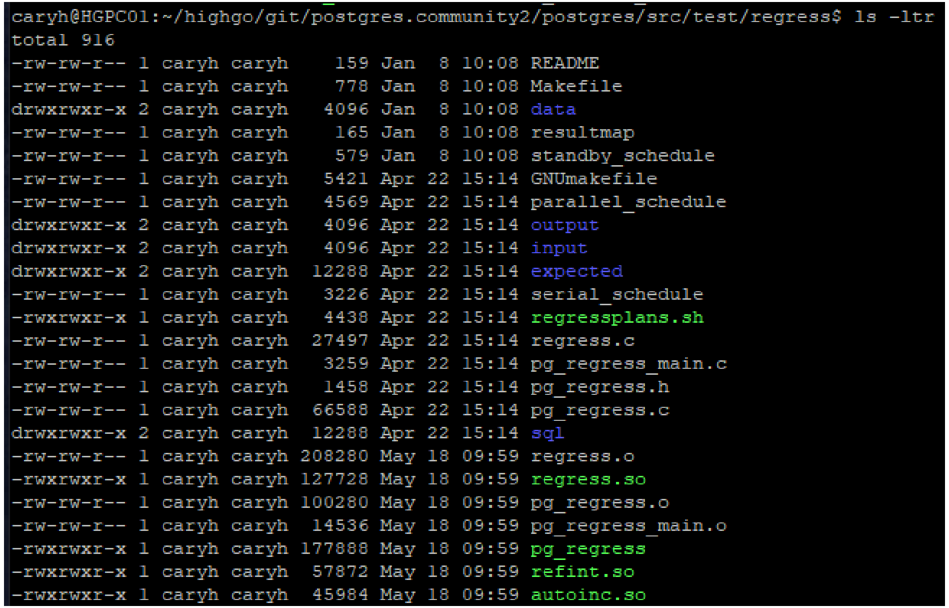
- https://en.wikipedia.org/wiki/List_of_software_bugs
- https://cs.uwaterloo.ca/~palencar/cs447/lectures

PG Regression Test Framework

Regression Framework

Using PG Regression Framework

- The regression tests are a comprehensive set of tests for the SQL implementation in PostgreSQL.
- They test standard SQL operations as well as the extended capabilities of PostgreSQL.
- So… whatever you are doing with PostgreSQL, you need to pass all of these regression tests to be “SQL” compliant.
- The framework is located in src/test/regress subfolder.

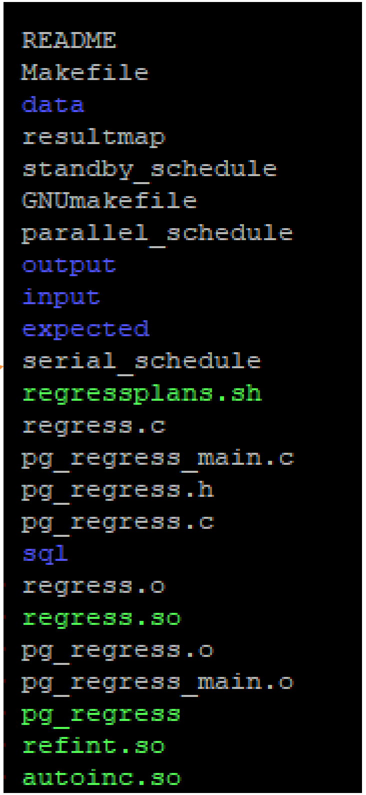


```
caryh@HGPC01:~/highgo/git/postgres.community2/postgres/src/test/regress$ ls -ltr
total 916
-rw-rw-r-- 1 caryh caryh    159 Jan  8 10:08 README
-rw-rw-r-- 1 caryh caryh    778 Jan  8 10:08 Makefile
drwxrwxr-x 2 caryh caryh   4096 Jan  8 10:08 data
-rw-rw-r-- 1 caryh caryh    165 Jan  8 10:08 resultmap
-rw-rw-r-- 1 caryh caryh    579 Jan  8 10:08 standby_schedule
-rw-rw-r-- 1 caryh caryh   5421 Apr 22 15:14 GNUmakefile
-rw-rw-r-- 1 caryh caryh   4569 Apr 22 15:14 parallel_schedule
drwxrwxr-x 2 caryh caryh   4096 Apr 22 15:14 output
drwxrwxr-x 2 caryh caryh   4096 Apr 22 15:14 input
drwxrwxr-x 2 caryh caryh  12288 Apr 22 15:14 expected
-rw-rw-r-- 1 caryh caryh   3226 Apr 22 15:14 serial_schedule
-rwxrwxr-x 1 caryh caryh   4438 Apr 22 15:14 regressplans.sh
-rw-rw-r-- 1 caryh caryh  27497 Apr 22 15:14 regress.c
-rw-rw-r-- 1 caryh caryh   3259 Apr 22 15:14 pg_regress_main.c
-rw-rw-r-- 1 caryh caryh   1458 Apr 22 15:14 pg_regress.h
-rw-rw-r-- 1 caryh caryh  66588 Apr 22 15:14 pg_regress.c
drwxrwxr-x 2 caryh caryh  12288 Apr 22 15:14 sql
-rw-rw-r-- 1 caryh caryh 208280 May 18 09:59 regress.o
-rwxrwxr-x 1 caryh caryh 127728 May 18 09:59 regress.so
-rw-rw-r-- 1 caryh caryh 100280 May 18 09:59 pg_regress.o
-rw-rw-r-- 1 caryh caryh  14536 May 18 09:59 pg_regress_main.o
-rwxrwxr-x 1 caryh caryh 177888 May 18 09:59 pg_regress
-rwxrwxr-x 1 caryh caryh  57872 May 18 09:59 refint.so
-rwxrwxr-x 1 caryh caryh  45984 May 18 09:59 autoinc.so
```

Regression Framework

What is inside…

- The “data” folder
  - Contains additional data files to include during regression tests
- The “parallel_schedule” file
  - Defines what regression tests to be run “together” in parallel.
- The “serial_schedule” file
  - Defines what regression tests to be run in sequence.

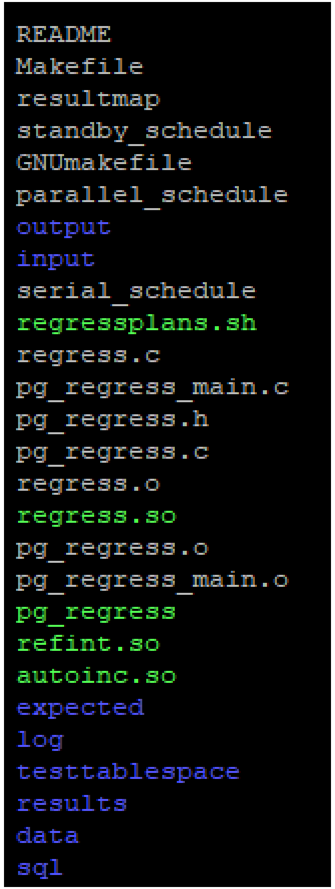


```
README
Makefile
data
resultmap
standby_schedule
GNUmakefile
parallel_schedule
output
input
expected
serial_schedule
regressplans.sh
regress.c
pg_regress_main.c
pg_regress.h
pg_regress.c
sql
regress.o
regress.so
pg_regress.o
pg_regress_main.o
pg_regress
refint.so
autoinc.so
```

Regression Framework

What is inside…

- The **pg_regress** application
  - The driver program to run the regression tests
- The “**expected**” folder
  - Contains the expected outputs of all of your tests
- The “**results**” folder
  - Contains the results of your test scripts
- The “**sql**” folder
  - Contains all of your test scripts

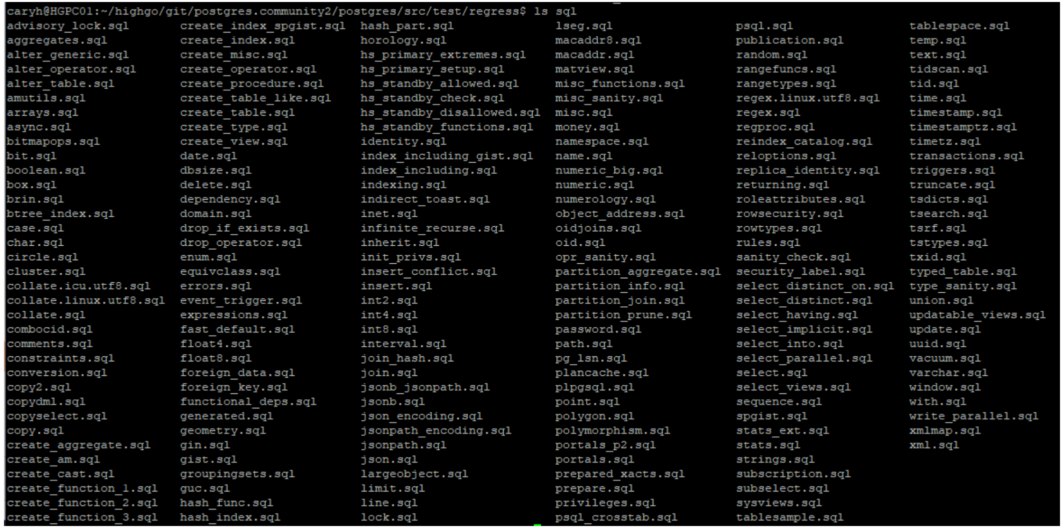


```
README
Makefile
resultmap
standby_schedule
GNUmakefile
parallel_schedule
output
input
serial_schedule
regressplans.sh
regress.c
pg_regress_main.c
pg_regress.h
pg_regress.c
regress.o
regress.so
pg_regress.o
pg_regress_main.o
pg_regress
refint.so
autoinc.so
expected
log
testtablespace
results
data
sql
```

What is being tests?

All the test scripts…

- There are lots of test scripts located inside the “sql” folder
- Each script is designed to test a particular “SQL” syntax or behavior.
- You are free to add more test scripts inside this folder



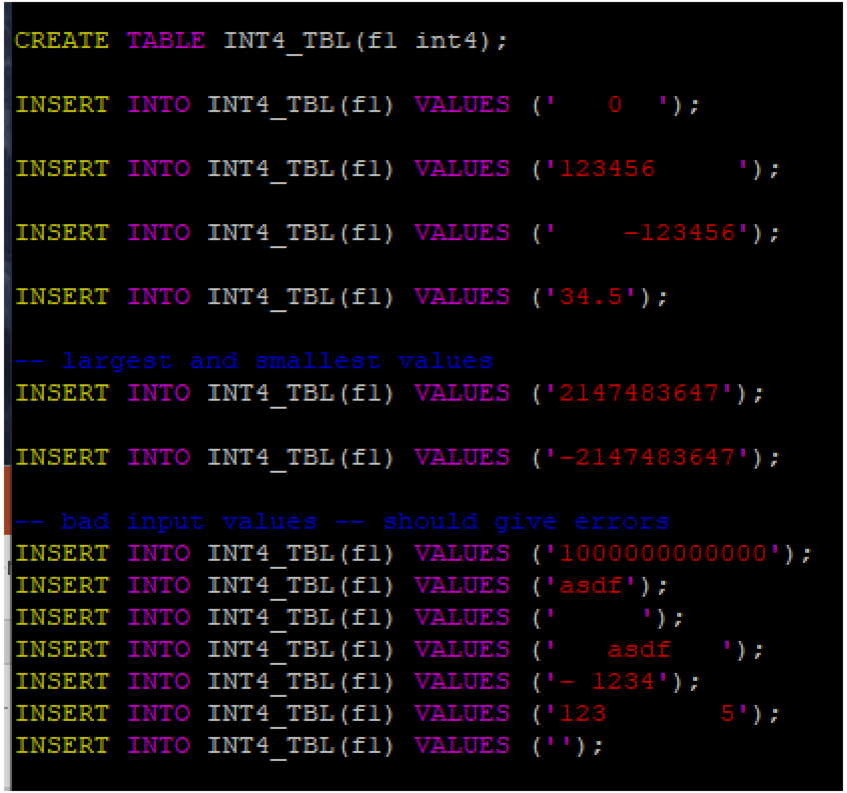
```
caryh@HGPC01:~/highgo/git/postgres.community2/postgres/src/test/regress$ ls sql
advisory_lock.sql       create_index_spgist.sql  hash_part.sql              lseg.sql                 psql.sql                tablespace.sql
aggregates.sql          create_index.sql         horology.sql               macaddr8.sql             publication.sql         temp.sql
alter_generic.sql       create_misc.sql          hs_primary_extremes.sql    macaddr.sql              random.sql              text.sql
alter_operator.sql      create_operator.sql      hs_primary_setup.sql       matview.sql              rangefuncs.sql          tidscan.sql
alter_table.sql         create_procedure.sql     hs_standby_allowed.sql     misc_functions.sql       rangetypes.sql          tid.sql
amutils.sql             create_table_like.sql    hs_standby_check.sql       misc_sanity.sql          regex.linux.utf8.sql    time.sql
arrays.sql              create_table.sql         hs_standby_disallowed.sql  misc.sql                 regex.sql               timestamp.sql
async.sql               create_type.sql          hs_standby_functions.sql   money.sql                regproc.sql             timestamptz.sql
bitmapops.sql           create_view.sql          identity.sql               namespace.sql            reindex_catalog.sql     timetz.sql
bit.sql                 date.sql                 index_including_gist.sql   name.sql                 reloptions.sql          transactions.sql
boolean.sql             dbsize.sql               index_including.sql        numeric_big.sql          replica_identity.sql    triggers.sql
box.sql                 delete.sql               indexing.sql               numeric.sql              returning.sql           truncate.sql
brin.sql                dependency.sql           indirect_toast.sql         numerology.sql           roleattributes.sql      tsdicts.sql
btree_index.sql         domain.sql               inet.sql                   object_address.sql       rowsecurity.sql         tsearch.sql
case.sql                drop_if_exists.sql       infinite_recurse.sql       oidjoins.sql             rowtypes.sql            tsrf.sql
char.sql                drop_operator.sql        inherit.sql                oid.sql                  rules.sql               tstypes.sql
circle.sql              enum.sql                 init_privs.sql             opr_sanity.sql           sanity_check.sql        txid.sql
cluster.sql             equivclass.sql           insert_conflict.sql        partition_aggregate.sql  security_label.sql      typed_table.sql
collate.icu.utf8.sql    errors.sql               insert.sql                 partition_info.sql       select_distinct_on.sql  type_sanity.sql
collate.linux.utf8.sql  event_trigger.sql        int2.sql                   partition_join.sql       select_distinct.sql     union.sql
collate.sql             expressions.sql          int4.sql                   partition_prune.sql      select_having.sql       updatable_views.sql
combocid.sql            fast_default.sql         int8.sql                   password.sql             select_implicit.sql     update.sql
comments.sql            float4.sql               interval.sql               path.sql                 select_into.sql         uuid.sql
constraints.sql         float8.sql               join_hash.sql              pg_lsn.sql               select_parallel.sql     vacuum.sql
conversion.sql          foreign_data.sql         join.sql                   plancache.sql            select.sql              varchar.sql
copy2.sql               foreign_key.sql          jsonb_jsonpath.sql         plpgsql.sql              select_views.sql        window.sql
copydml.sql             functional_deps.sql      jsonb.sql                  point.sql                sequence.sql            with.sql
copyselect.sql          generated.sql            json_encoding.sql          polygon.sql              spgist.sql              write_parallel.sql
copy.sql                geometry.sql             jsonpath_encoding.sql      polymorphism.sql         stats_ext.sql           xmlmap.sql
create_aggregate.sql    gin.sql                  jsonpath.sql               portals_p2.sql           stats.sql               xml.sql
create_am.sql           gist.sql                 json.sql                   portals.sql              strings.sql
create_cast.sql         groupingsets.sql         largeobject.sql            prepared_xacts.sql       subscription.sql
create_function_1.sql   guc.sql                  limit.sql                  prepare.sql              subselect.sql
create_function_2.sql   hash_func.sql            line.sql                   privileges.sql           sysviews.sql
create_function_3.sql   hash_index.sql           lock.sql                   psql_crosstab.sql        tablesample.sql
```

Example: sql/int4.sql

Look familiar, doesn’t it?

- Each (.sql) script in the /sql folder contains a series of SQL instructions that you would type in an active “psql” connection.
- For example,
    - CREATE TABLE xxxxx
    - INSERT INTO xxxx
    - DELETE xxxx
- Each of these SQL statement will produce an output and store in results/int4.out

src/test/regress/sql/int4.sql



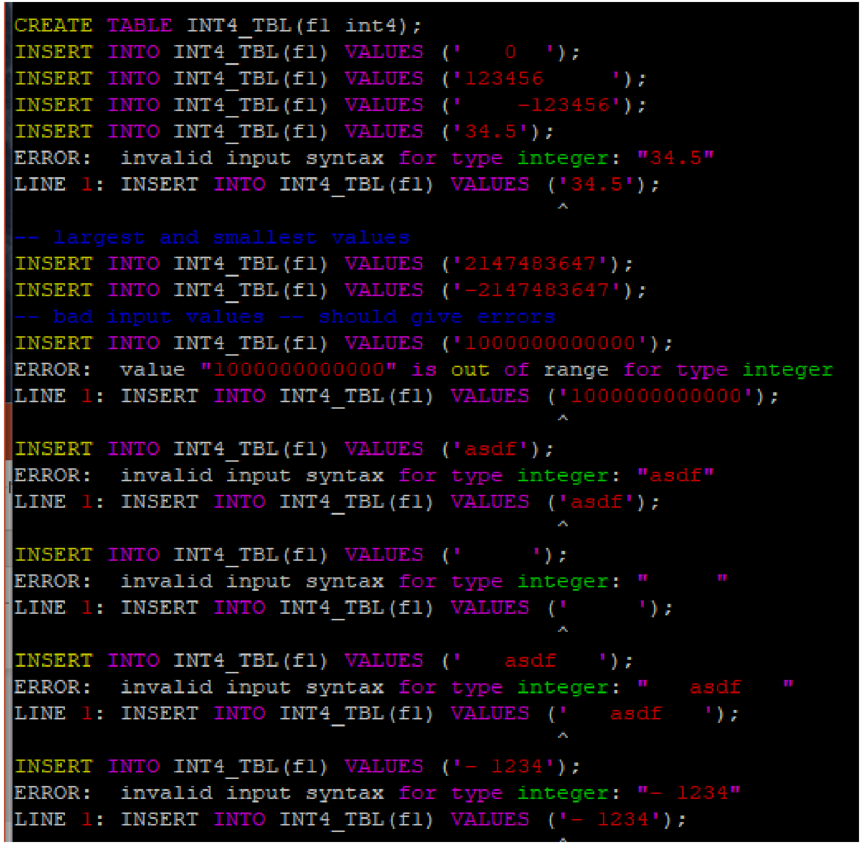
```sql
CREATE TABLE INT4_TBL(f1 int4);

INSERT INTO INT4_TBL(f1) VALUES ('   0  ');

INSERT INTO INT4_TBL(f1) VALUES ('123456     ');

INSERT INTO INT4_TBL(f1) VALUES ('    -123456');

INSERT INTO INT4_TBL(f1) VALUES ('34.5');

-- largest and smallest values
INSERT INTO INT4_TBL(f1) VALUES ('2147483647');

INSERT INTO INT4_TBL(f1) VALUES ('-2147483647');

-- bad input values -- should give errors
INSERT INTO INT4_TBL(f1) VALUES ('1000000000000');
INSERT INTO INT4_TBL(f1) VALUES ('asdf');
INSERT INTO INT4_TBL(f1) VALUES ('    ');
INSERT INTO INT4_TBL(f1) VALUES ('   asdf   ');
INSERT INTO INT4_TBL(f1) VALUES ('- 1234');
INSERT INTO INT4_TBL(f1) VALUES ('123       5');
INSERT INTO INT4_TBL(f1) VALUES ('');
```

Example: results/int4.out

Look familiar, doesn’t it?

- Each (.out) file in the /results folder contains the same SQL instructions in the (.sql) scripts plus results/error messages.
- The tests cover positive and negative cases, so it is common to see a lot of errors in the (.out) files.

src/test/regress/results/int4.out



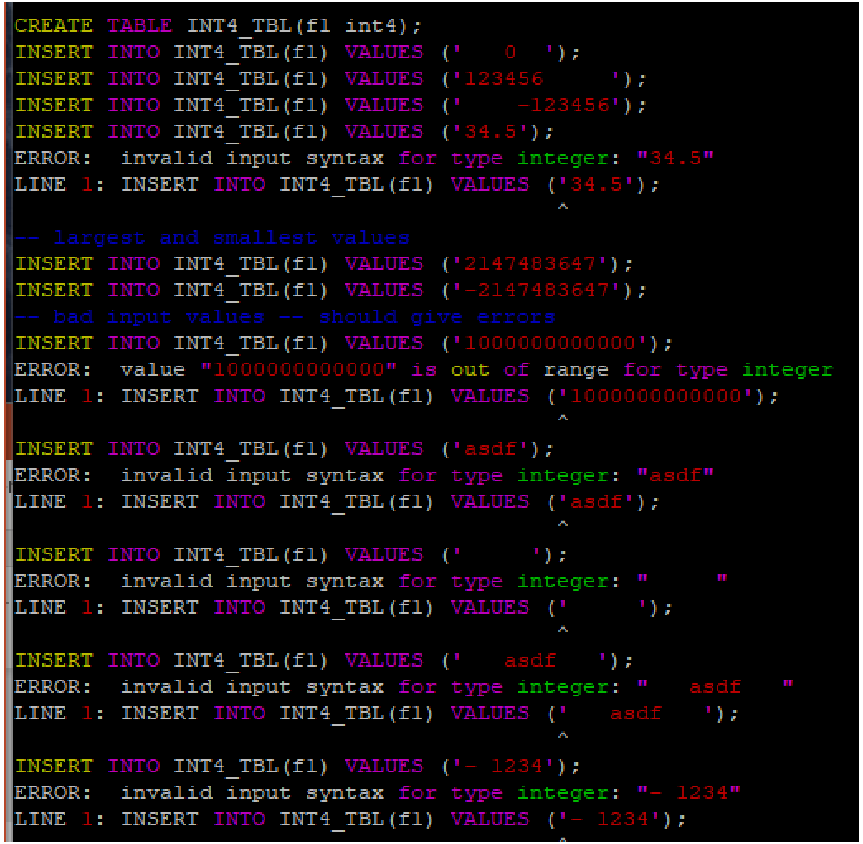
```
CREATE TABLE INT4_TBL(f1 int4);
INSERT INTO INT4_TBL(f1) VALUES ('   0  ');
INSERT INTO INT4_TBL(f1) VALUES ('123456     ');
INSERT INTO INT4_TBL(f1) VALUES ('    -123456');
INSERT INTO INT4_TBL(f1) VALUES ('34.5');
ERROR:  invalid input syntax for type integer: "34.5"
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('34.5');
                                         ^

-- largest and smallest values
INSERT INTO INT4_TBL(f1) VALUES ('2147483647');
INSERT INTO INT4_TBL(f1) VALUES ('-2147483647');
-- bad input values -- should give errors
INSERT INTO INT4_TBL(f1) VALUES ('1000000000000');
ERROR:  value "1000000000000" is out of range for type integer
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('1000000000000');
                                         ^

INSERT INTO INT4_TBL(f1) VALUES ('asdf');
ERROR:  invalid input syntax for type integer: "asdf"
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('asdf');
                                         ^

INSERT INTO INT4_TBL(f1) VALUES ('      ');
ERROR:  invalid input syntax for type integer: "      "
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('      ');
                                         ^

INSERT INTO INT4_TBL(f1) VALUES ('   asdf   ');
ERROR:  invalid input syntax for type integer: "   asdf   "
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('   asdf   ');
                                         ^

INSERT INTO INT4_TBL(f1) VALUES ('- 1234');
ERROR:  invalid input syntax for type integer: "- 1234"
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('- 1234');
```

Example: expected/int4.out

Look familiar, doesn’t it?

- So far, we have
  - sql/int4.sql, which defines our tests statements
  - results/int4.out, which contains the result of execution of sql/int4.sql
- How do we know if the test produces the expected results?
- We can simply compare the files between results/int4.out and expected/int4.out.
- The expected file located in expected/in4.out is something we must prepare in order to tell regression framework what we want the output to be
- If both are the same, test passes, if not, test fails.

src/test/regress/expected/int4.out



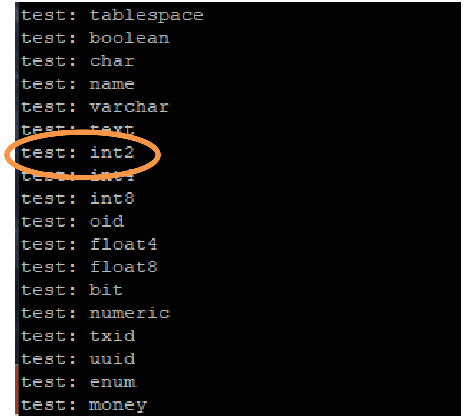
```
CREATE TABLE INT4_TBL(f1 int4);
INSERT INTO INT4_TBL(f1) VALUES ('   0  ');
INSERT INTO INT4_TBL(f1) VALUES ('123456     ');
INSERT INTO INT4_TBL(f1) VALUES ('    -123456');
INSERT INTO INT4_TBL(f1) VALUES ('34.5');
ERROR:  invalid input syntax for type integer: "34.5"
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('34.5');
                                         ^

-- largest and smallest values
INSERT INTO INT4_TBL(f1) VALUES ('2147483647');
INSERT INTO INT4_TBL(f1) VALUES ('-2147483647');
-- bad input values -- should give errors
INSERT INTO INT4_TBL(f1) VALUES ('1000000000000');
ERROR:  value "1000000000000" is out of range for type integer
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('1000000000000');
                                         ^

INSERT INTO INT4_TBL(f1) VALUES ('asdf');
ERROR:  invalid input syntax for type integer: "asdf"
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('asdf');
                                         ^

INSERT INTO INT4_TBL(f1) VALUES ('     ');
ERROR:  invalid input syntax for type integer: "     "
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('     ');
                                         ^

INSERT INTO INT4_TBL(f1) VALUES ('   asdf   ');
ERROR:  invalid input syntax for type integer: "   asdf   "
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('   asdf   ');
                                         ^

INSERT INTO INT4_TBL(f1) VALUES ('- 1234');
ERROR:  invalid input syntax for type integer: "- 1234"
LINE 1: INSERT INTO INT4_TBL(f1) VALUES ('- 1234');
```

Configuring The Test

Let' s make a test run!

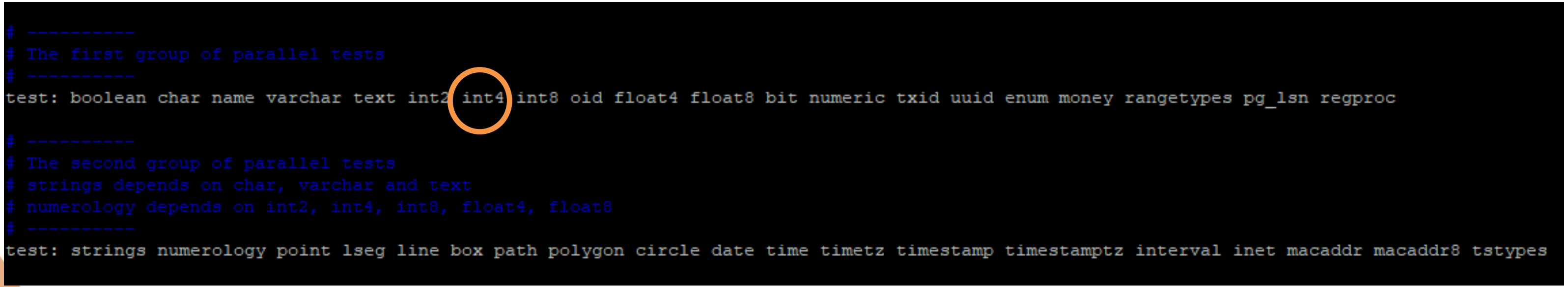
- Next, we need to tell PG regression framework about our test script and include that in the regression test.
- This is done by modifying both "parallel_schedule" and "serial_schedule" files and add your test in the appropriate places

src/test/regress/serial_schedule



```
test: tablespace
test: boolean
test: char
test: name
test: varchar
test: text
test: int2
test: int4
test: int8
test: oid
test: float4
test: float8
test: bit
test: numeric
test: txid
test: uuid
test: enum
test: money
```

src/test/regress/parallel_schedule

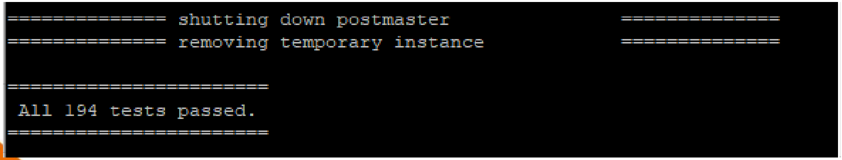


```
# ----------
# The first group of parallel tests
# ----------
test: boolean char name varchar text int2 int4 int8 oid float4 float8 bit numeric txid uuid enum money rangetypes pg_lsn regproc

# ----------
# The second group of parallel tests
# strings depends on char, varchar and text
# numerology depends on int2, int4, int8, float4, float8
# ----------
test: strings numerology point lseg line box path polygon circle date time timetz timestamp timestamptz interval inet macaddr macaddr8 tstypes
```

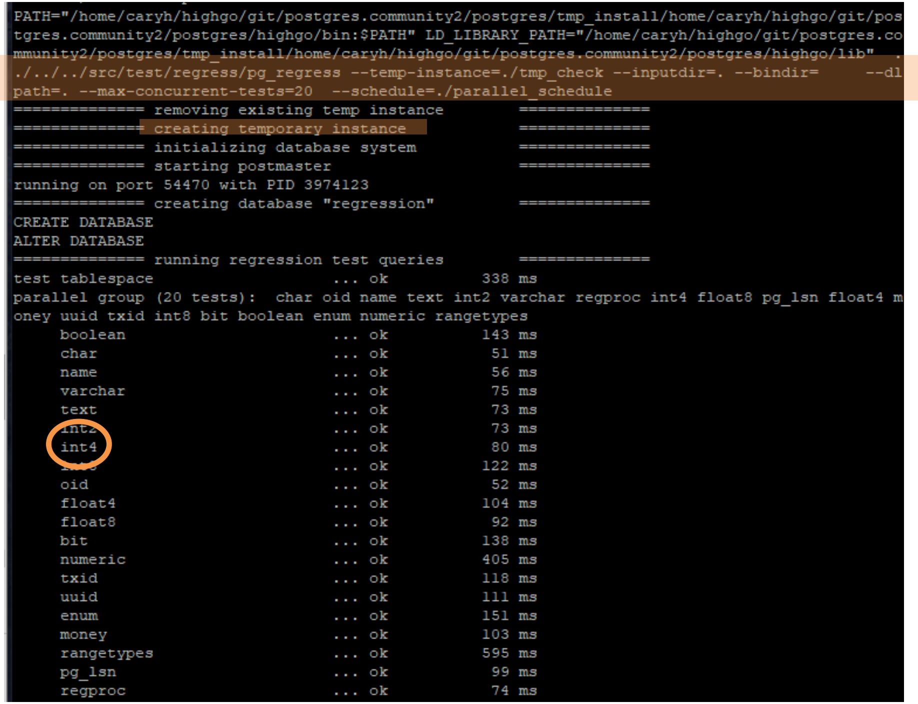
Running The Test on Temp Instance

Run against a temporary installation

- You can trigger the regression test on a temporary database instance simply by using the "**make check**" command in the root of PostgreSQL directory.
- By default, the command will use **parallel_schedule** configuration to run the test.
- At the end of the execution, a test summary will be provided



```
============== shutting down postmaster               ==============
============== removing temporary instance            ==============

=======================
 All 194 tests passed.
=======================
```

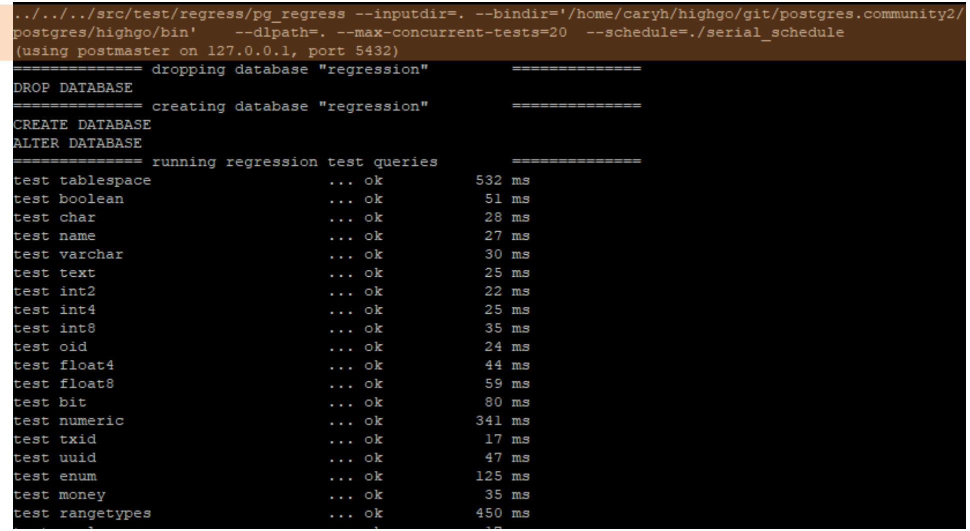


```
PATH="/home/caryh/highgo/git/postgres.community2/postgres/tmp_install/home/caryh/highgo/git/pos
tgres.community2/postgres/highgo/bin:$PATH" LD_LIBRARY_PATH="/home/caryh/highgo/git/postgres.co
mmunity2/postgres/tmp_install/home/caryh/highgo/git/postgres.community2/postgres/highgo/lib" .
./../../src/test/regress/pg_regress --temp-instance=./tmp_check --inputdir=. --bindir=     --dl
path=. --max-concurrent-tests=20  --schedule=./parallel_schedule
============== removing existing temp instance        ==============
============== creating temporary instance            ==============
============== initializing database system           ==============
============== starting postmaster                    ==============
running on port 54470 with PID 3974123
============== creating database "regression"         ==============
CREATE DATABASE
ALTER DATABASE
============== running regression test queries        ==============
test tablespace                   ... ok          338 ms
parallel group (20 tests):  char oid name text int2 varchar regproc int4 float8 pg_lsn float4 m
oney uuid txid int8 bit boolean enum numeric rangetypes
     boolean                      ... ok          143 ms
     char                         ... ok           51 ms
     name                         ... ok           56 ms
     varchar                      ... ok           75 ms
     text                         ... ok           73 ms
     int2                         ... ok           73 ms
     int4                         ... ok           80 ms
     int8                         ... ok          122 ms
     oid                          ... ok           52 ms
     float4                       ... ok          104 ms
     float8                       ... ok           92 ms
     bit                          ... ok          138 ms
     numeric                      ... ok          405 ms
     txid                         ... ok          118 ms
     uuid                         ... ok          111 ms
     enum                         ... ok          151 ms
     money                        ... ok          103 ms
     rangetypes                   ... ok          595 ms
     pg_lsn                       ... ok           99 ms
     regproc                      ... ok           74 ms
```

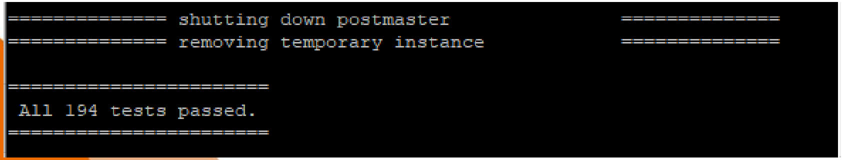
Running The Test On Your Instance

Run against a Running installation

- You can trigger the regression test on your own running database instance using the following commands:
  - export PGHOST=127.0.0.1
  - export PGPORT=5432
  - “make installcheck”
- By default, the command will use serial_schedule configuration to run the test.
- At the end of the execution, a test summary will be provided



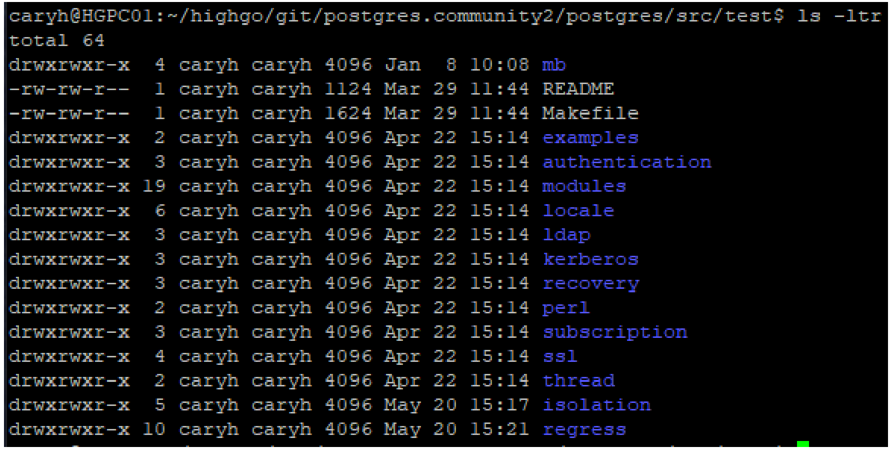
```
../../../src/test/regress/pg_regress --inputdir=. --bindir='/home/caryh/highgo/git/postgres.community2/
postgres/highgo/bin'    --dlpath=. --max-concurrent-tests=20  --schedule=./serial_schedule
(using postmaster on 127.0.0.1, port 5432)
============== dropping database "regression"         ==============
DROP DATABASE
============== creating database "regression"         ==============
CREATE DATABASE
ALTER DATABASE
============== running regression test queries        ==============
test tablespace                   ... ok          532 ms
test boolean                      ... ok           51 ms
test char                         ... ok           28 ms
test name                         ... ok           27 ms
test varchar                      ... ok           30 ms
test text                         ... ok           25 ms
test int2                         ... ok           22 ms
test int4                         ... ok           25 ms
test int8                         ... ok           35 ms
test oid                          ... ok           24 ms
test float4                       ... ok           44 ms
test float8                       ... ok           59 ms
test bit                          ... ok           80 ms
test numeric                      ... ok          341 ms
test txid                         ... ok           17 ms
test uuid                         ... ok           47 ms
test enum                         ... ok          125 ms
test money                        ... ok           35 ms
test rangetypes                   ... ok          450 ms
```



```
============== shutting down postmaster               ==============
============== removing temporary instance            ==============

=======================
 All 194 tests passed.
=======================
```

Running The Full Test

Run against a Running installation



```
caryh@HGPC01:~/highgo/git/postgres.community2/postgres/src/test$ ls -ltr
total 64
drwxrwxr-x  4 caryh caryh 4096 Jan  8 10:08 mb
-rw-rw-r--  1 caryh caryh 1124 Mar 29 11:44 README
-rw-rw-r--  1 caryh caryh 1624 Mar 29 11:44 Makefile
drwxrwxr-x  2 caryh caryh 4096 Apr 22 15:14 examples
drwxrwxr-x  3 caryh caryh 4096 Apr 22 15:14 authentication
drwxrwxr-x 19 caryh caryh 4096 Apr 22 15:14 modules
drwxrwxr-x  6 caryh caryh 4096 Apr 22 15:14 locale
drwxrwxr-x  3 caryh caryh 4096 Apr 22 15:14 ldap
drwxrwxr-x  3 caryh caryh 4096 Apr 22 15:14 kerberos
drwxrwxr-x  3 caryh caryh 4096 Apr 22 15:14 recovery
drwxrwxr-x  2 caryh caryh 4096 Apr 22 15:14 perl
drwxrwxr-x  3 caryh caryh 4096 Apr 22 15:14 subscription
drwxrwxr-x  4 caryh caryh 4096 Apr 22 15:14 ssl
drwxrwxr-x  2 caryh caryh 4096 Apr 22 15:14 thread
drwxrwxr-x  5 caryh caryh 4096 May 20 15:17 isolation
drwxrwxr-x 10 caryh caryh 4096 May 20 15:21 regress
```

- PG is a huge system consisting of many, many features and functions.
- What we have covered so far is only 194 test cases that are required to be SQL compliant.
- There are in fact many more tests included in PostgreSQL that do not get run by default.
- They are located in these folders
- There is a command shortcut to trigger all of the tests to be run
- The “make check-world” command
- It could take some time to complet all of the tests…

PG Regression Demonstration

I will now show you how you can create your own Regression test cases and run them…

瀚高
HIGH GO

融知与行 瀚且高远

THANKS