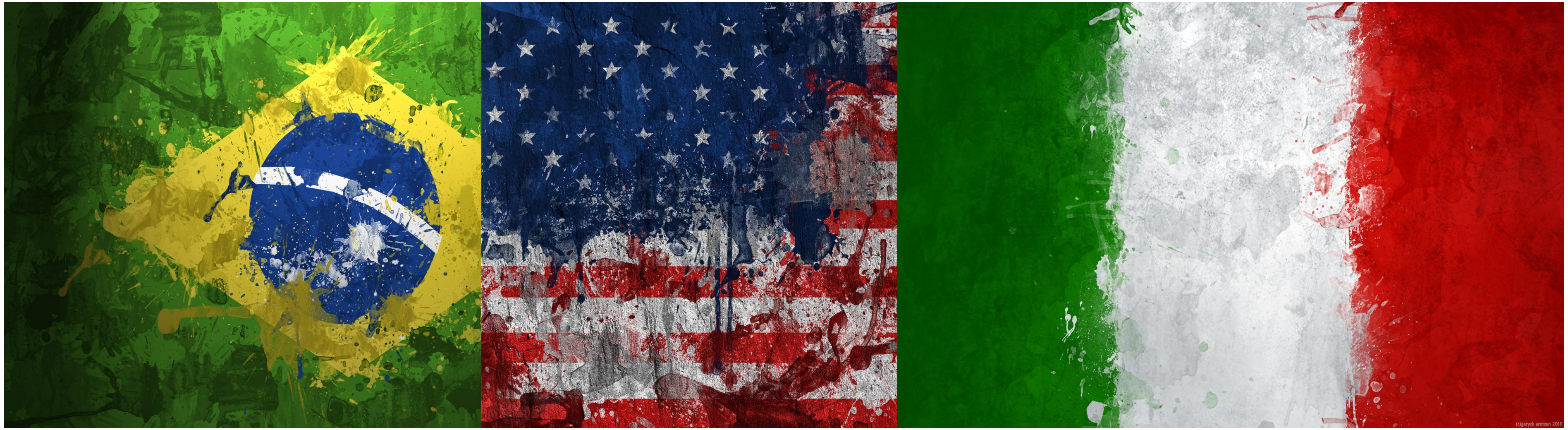
Test Framework to Git



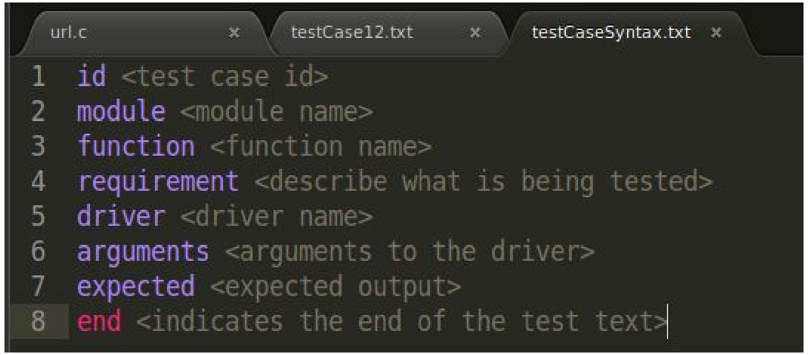
Cassios, Danzel, and Romeo

What is Git?

- Git is a free and open source distributed version control system.
- Designed and developed for Linux kernel development in 2005.
- Original authors: Linus Torvalds.
- Written in: Perl, Tcl, Python, C++.

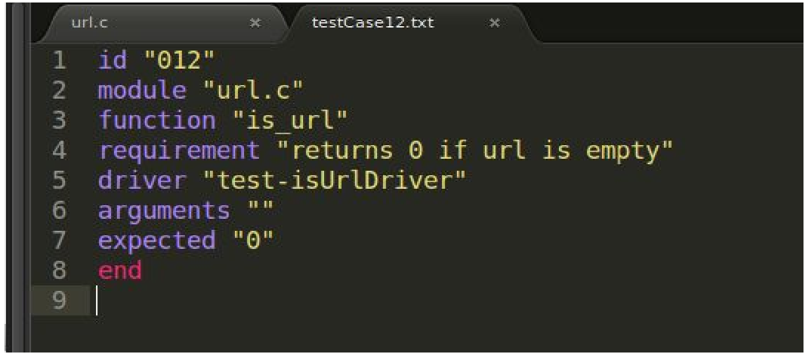
Test Cases Syntax

Syntax



```
id <test case id>
module <module name>
function <function name>
requirement <describe what is being tested>
driver <driver name>
arguments <arguments to the driver>
expected <expected output>
end <indicates the end of the test text>
```

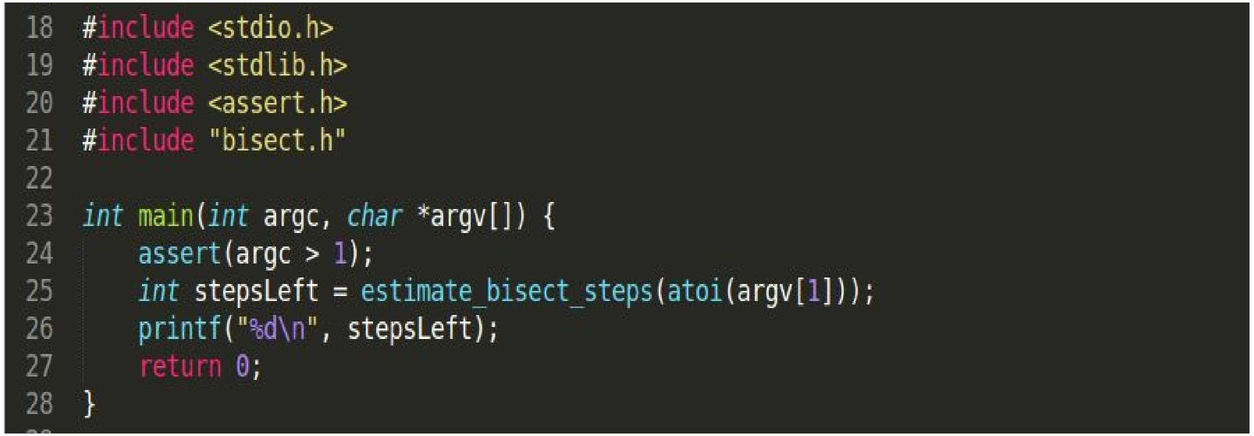
Example



```
id "012"
module "url.c"
function "is_url"
requirement "returns 0 if url is empty"
driver "test-isUrlDriver"
arguments ""
expected "0"
end
```

Drivers

- One driver to each function being tested
- Add driver to be compiled by Makefile



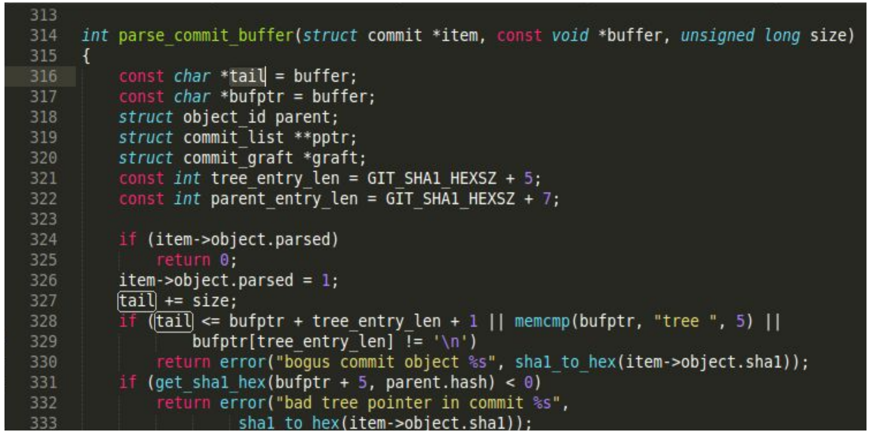
```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "bisect.h"

int main(int argc, char *argv[]) {
    assert(argc > 1);
    int stepsLeft = estimate_bisect_steps(atoi(argv[1]));
    printf("%d\n", stepsLeft);
    return 0;
}
```

Script

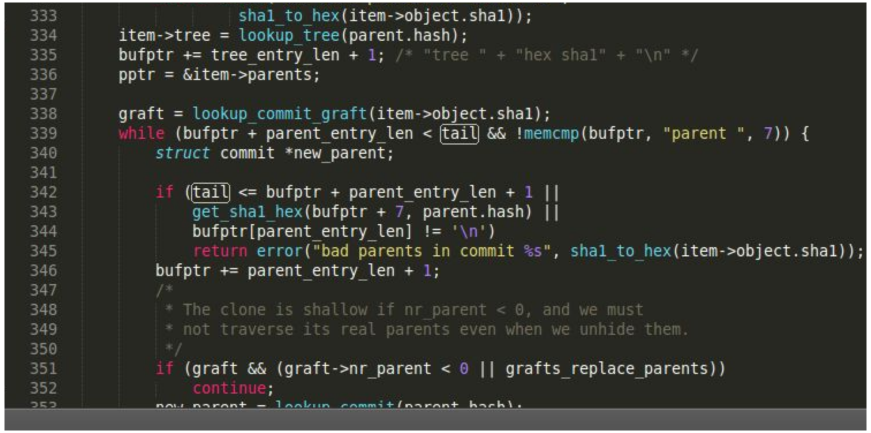
- Copy drivers in executables directory to git directory
- Compile project
- Get test cases
- For each testCase
    - Read test case metadata
    - Read the driver’s name
    - Read arguments
    - Execute driver with arguments
    - Compare output against expected value
    - Add results to html file
- Open browser with results

Difficulties in testing functions

- A lot of dependent function involved
- Use environment variables
- Huge functions



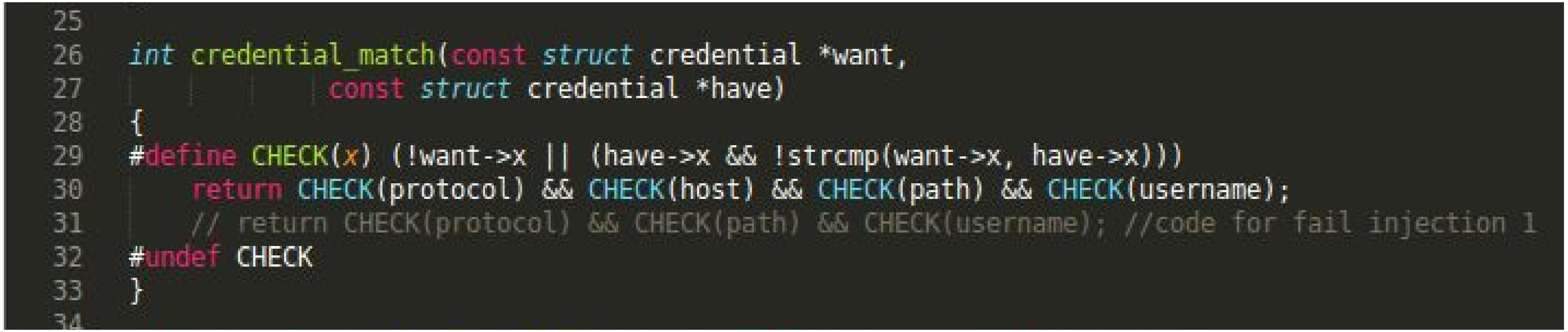
```c

int parse_commit_buffer(struct commit *item, const void *buffer, unsigned long size)
{
    const char *tail = buffer;
    const char *bufptr = buffer;
    struct object_id parent;
    struct commit_list **pptr;
    struct commit_graft *graft;
    const int tree_entry_len = GIT_SHA1_HEXSZ + 5;
    const int parent_entry_len = GIT_SHA1_HEXSZ + 7;

    if (item->object.parsed)
        return 0;
    item->object.parsed = 1;
    tail += size;
    if (tail <= bufptr + tree_entry_len + 1 || memcmp(bufptr, "tree ", 5) ||
            bufptr[tree_entry_len] != '\n')
        return error("bogus commit object %s", sha1_to_hex(item->object.sha1));
    if (get_sha1_hex(bufptr + 5, parent.hash) < 0)
        return error("bad tree pointer in commit %s",
                 sha1_to_hex(item->object.sha1));
```



```c
                 sha1_to_hex(item->object.sha1));
    item->tree = lookup_tree(parent.hash);
    bufptr += tree_entry_len + 1; /* "tree " + "hex sha1" + "\n" */
    pptr = &item->parents;

    graft = lookup_commit_graft(item->object.sha1);
    while (bufptr + parent_entry_len < tail && !memcmp(bufptr, "parent ", 7)) {
        struct commit *new_parent;

        if (tail <= bufptr + parent_entry_len + 1 ||
            get_sha1_hex(bufptr + 7, parent.hash) ||
            bufptr[parent_entry_len] != '\n')
            return error("bad parents in commit %s", sha1_to_hex(item->object.sha1));
        bufptr += parent_entry_len + 1;
        /*
         * The clone is shallow if nr_parent < 0, and we must
         * not traverse its real parents even when we unhide them.
         */
        if (graft && (graft->nr_parent < 0 || grafts_replace_parents))
            continue;
        new_parent = lookup_commit(parent.hash);
```

Fault Injections

- 5 faults injected in different functions



```c

int credential_match(const struct credential *want,
                     const struct credential *have)
{
#define CHECK(x) (!want->x || (have->x && !strcmp(want->x, have->x)))
    return CHECK(protocol) && CHECK(host) && CHECK(path) && CHECK(username);
    // return CHECK(protocol) && CHECK(path) && CHECK(username); //code for fail injection 1
#undef CHECK
}

```

Summary

- Create a test framework to Git
- Challenges in testing using a non-mature test framework
  - Lack of mock and stub
  - C is hard language to test

Experience

Q&A

Thanks