Chapter 4

Experience

Once again, we had a good experience working as a team. We split the work in three main tasks: write documentation, improve script to run tests, and write the drivers for the remaining 20 test cases. Writing the tests cases was time consuming because it was hard to find functions that were neither too complicated or static (they can only be called inside the module where they were defined). We also changed our test cases output layout to a table view, so it's easier to read the information. By using the table view, the 25 test cases are shown in a cleaner view.

Test Cases

ID	Module	Function	Requirement	Driver	Arguments	Expected	Result
1	credential.c	credential_match	returns 1 credentials have the same protocol, otherwise 0	test-credentialMatch Driver	"https example.com foo.git bob http example.com foo.git bob"	0	0
2	credential.c	credential_match	returns 1 credentials have the same host, otherwise 0	test-credentialMatch Driver	"https example.com foo.git bob https otherExample.com foo.git bob"	0	0
3	credential.c	credential_match	returns 1 credentials have the same path, otherwise 0	test-credentialMatch Driver	"https example.com foo.git bob https example.com bar.git bob"	0	0
4	credential.c	credential_match	returns 1 credentials have the same username, otherwise 0	test-credentialMatch Driver	"https example.com foo.git bob https example.co foo.git mary"	0	0
5	credential.c	credential_match	returns 1 when every field of the credentials are equal	test-credentialMatch Driver	"https example.com foo.git bob https example.com foo.git bob"	1	1
6	url.c	is_urlschemechar	return 0 if '+' is the first character, otherwise 1	test-urlSchemecharDriver	"1 +"	1	0
7	url.c	is_urlschemechar	returns 0 if '-' as the first character, otherwise 1	test-urlSchemecharDriver	"1 -"	0	0

8	url.c	is_urlschemechar	returns 0 if '.' as the first character, otherwise 1	test-urlSchemecharDriver	"1 ."	0	0
9	url.c	is_urlschemechar	returns 1 if the special character is in other positions that is not the first one, otherwise 0	test-urlSchemecharDriver	"0 +"	1	1
10	url.c	is_urlschemechar	returns 1 if the character in the first position is a number	test-urlSchemecharDriver	"1/8/2015"	1	1
11	url.c	is_urlschemechar	allows numbers in the other positions of the url	test-urlSchemecharDriver	"0 8"	1	1
12	url.c	is_url	verifies if url is not empty	test-isUrlDriver	"" ""	0	0
13	url.c	is_url	return 0 if the first character of the url is a special character(+ - .)	test-isUrlDriver	"+https://myurl.com"	0	0
14	url.c	is_url	verifies if url has the pattern '://'	test-isUrlDriver	"https:myurl.com"	0	0
15	bisect.c	estimate_bisect_steps	returns 0 if n is less than 3	test-estimateBisectStepsDriver	"2"	0	0
16	bisect.c	estimate_bisect_steps	returns the integer log e of n minus 1 ((logi n)-1) if (2^e < 3*(n - 2^e))	test-estimateBisectStepsDriver	"10"	2	2
17	bisect.c	estimate_bisect_steps	returns integer log e of n if (2^e < 3*(n - 2^e))	test-estimateBisectStepsDriver	"100"	6	6
18	color.c	git_config_colorbool	returns 0 if value is equal to 'never'	test-gitConfigColorboolDriver	"core.color never"	0	0
19	color.c	git_config_colorbool	returns 1 if value is equal to 'always'	test-gitConfigColorboolDriver	"core.color always"	1	1
20	color.c	git_config_colorbool	returns the automatic color(2) if the value is equal to 'auto'	test-gitConfigColorboolDriver	"core.color auto"	2	2
21	color.c	git_config_colorbool	returns -1 if var and value are empty	test-gitConfigColorboolDriver	"returns -1 if var and value are empty"	-1	-1
22	color.c	git_config_colorbool	returns the automatic color(2) if var is not empty, but value is empty	test-gitConfigColorboolDriver	"core.color" ""	2	2

23	color.c	color_is_nil	returns 1 if color is nil	test-colorIsNil	"NIL"	1	1
24	color.c	color_is_nil	returns 0 if color is not nil	test-colorIsNil	"green"	0	0
25	commit.c	commit_list_count	returns 0 if commit list is empty	test-commitListCount	"0"	0	0
26	commit.c	commit_list_count	returns the number of items in the commit list if it is not empty	test-commitListCount	"4"	4	4

Methods being tested

```
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
}

int is_urlschemechar(int first_flag, int ch) {
        /*
         * The set of valid URL schemes, as per STD66 (RFC3986) is
         * '[A-Za-z][A-Za-z0-9+.-]*'. But use sightly looser check
         * of '[A-Za-z0-9][A-Za-z0-9+.-]*' because earlier version
         * of check used '[A-Za-z0-9]+' so not to break any remote
         * helpers.
         */
        int alphanumeric, special;
        alphanumeric = ch > 0 && isalnum(ch);
        // alphanumeric = ch > 0 && isalpha(ch); //code for fail injection 2
        special = ch == '+' || ch == '-' || ch == '.';
        return alphanumeric || (!first_flag && special);
}

int is_url(const char *url) {
        /* Is "scheme" part reasonable? */
        if (!url || !is_urlschemechar(1, *url++))
        // if (!url) //code for fail injection 3
                return 0;
        while (*url && *url != ':') {
                if (!is_urlschemechar(0, *url++))
                        return 0;
        }
        /* We've seen "scheme"; we want colon-slash-slash */
        return (url[0] == ':' && url[1] == '/' && url[2] == '/');
}

int estimate_bisect_steps(int all) {
        int n, x, e;
        if (all < 3)
                return 0;
        n = log2i(all);
        e = exp2i(n);
        x = all - e;
        return (e < 3 * x) ? n : n - 1;
        // return n; //code for fail injection 4
}

int git_config_colorbool(const char *var, const char *value) {
        if (value) {
                if (!strcasecmp(value, "never"))
                        return 0;
                if (!strcasecmp(value, "always"))
                        return 1;
                if (!strcasecmp(value, "auto"))
                        return GIT_COLOR_AUTO;
        }
        if (!var)
                return -1;
        /* Missing or explicit false to turn off colorization */
        if (!git_config_bool(var, value))
                return 0;

        /* any normal truth value defaults to 'auto' */
        return GIT_COLOR_AUTO;
        // return GIT_COLOR_RED; //code for fail injection 5
}

static int check_auto_color(void) {
        if (color_stdout_is_tty < 0)
                color_stdout_is_tty = isatty(1);
        if (color_stdout_is_tty || (pager_in_use() && pager_use_color)) {
                char *term = getenv("TERM");
                if (term && strcmp(term, "dumb"))
                        return 1;
        }
        return 0;
}

int want_color(int var) {
        static int want_auto = -1;
        if (var < 0)
                var = git_use_color_default;
        if (var == GIT_COLOR_AUTO) {
                if (want_auto < 0)
                        want_auto = check_auto_color();
                return want_auto;
        }
        return var;
}

int git_color_config(const char *var, const char *value, void *cb) {
        if (!strcmp(var, "color.ui")) {
                git_use_color_default = git_config_colorbool(var, value);
                return 0;
        }
        return 0;
}
```

```
int git_color_default_config(const char *var, const char *value, void *cb) {
	if (git_color_config(var, value, cb) < 0)
		return -1;
	return git_default_config(var, value, cb);
}

int color_is_nil(const char *c) {
	return !strcmp(c, "NIL");
}
unsigned commit_list_count(const struct commit_list *l) {
	unsigned c = 0;
	for (; l; l = l->next )
		c++;
	return c;
}
```