

---

# GlusterFS Coding Standards

**Z** Research

July 14, 2008

---

## • Structure definitions should have a comment per member

Every member in a structure definition must have a comment about its purpose. The comment should be descriptive without being overly verbose.

*Bad:*

```
gf_lock_t    lock;           /* lock */
```

*Good:*

```
DBTYPE      access_mode;    /* access mode for accessing
                             * the databases, can be
                             * DB_HASH, DB_BTREE
                             * (option access-mode <mode>)
                             */
```

## • Declare all variables at the beginning of the function

All local variables in a function must be declared immediately after the opening brace. This makes it easy to keep track of memory that needs to be freed during exit. It also helps debugging, since gdb cannot handle variables declared inside loops or other such blocks.

## • Always initialize local variables

Every local variable should be initialized to a sensible default value at the point of its declaration. All pointers should be initialized to NULL, and all integers should be zero or (if it makes sense) an error value.

*Good:*

```
int ret      = 0;
char *databuf = NULL;
int _fd      = -1;
```

- **Initialization should always be done with a constant value**

Never use a non-constant expression as the initialization value for a variable.

*Bad:*

```
pid_t pid      = frame->root->pid;
char *databuf = malloc (1024);
```

- **Validate all arguments to a function**

All pointer arguments to a function must be checked for NULL. A macro named `VALIDATE` (in `common-utils.h`) takes one argument, and if it is NULL, writes a log message and jumps to a label called `err` after setting `op_ret` and `op_errno` appropriately. It is recommended to use this template.

*Good:*

```
VALIDATE(frame);
VALIDATE(this);
VALIDATE(inode);
```

- **Never rely on precedence of operators**

Never write code that relies on the precedence of operators to execute correctly. Such code can be hard to read and someone else might not know the precedence of operators as accurately as you do.

*Bad:*

```
if (op_ret == -1 && errno != ENOENT)
```

*Good:*

```
if ((op_ret == -1) && (errno != ENOENT))
```

- **Use exactly matching types**

Use a variable of the exact type declared in the manual to hold the return value of a function. Do not use an “equivalent” type.

*Bad:*

```
int len = strlen (path);
```

*Good:*

```
size_t len = strlen (path);
```

- **Never write code such as `foo->bar->baz`; check every pointer**

Do not write code that blindly follows a chain of pointer references. Any pointer in the chain may be NULL and thus cause a crash. Verify that each pointer is non-null before following it.

- **Check return value of all functions and system calls**

The return value of all system calls and API functions must be checked for success or failure.

*Bad:*

```
close (fd);
```

*Good:*

```
op_ret = close (_fd);
if (op_ret == -1) {
    gf_log (this->name, GF_LOG_ERROR,
            "close on file %s failed (%s)", real_path,
            strerror (errno));
    op_errno = errno;
    goto err;
}
```

- **Gracefully handle failure of malloc**

GlusterFS should never crash or exit due to lack of memory. If a memory allocation fails, the call should be unwound and an error returned to the user.

- **Use result args and reserve the return value to indicate success or failure**

The return value of every functions must indicate success or failure (unless it is impossible for the function to fail — e.g., boolean functions). If the function needs to return additional data, it must be returned using a result (pointer) argument.

*Bad:*

```
int32_t dict_get_int32 (dict_t *this, char *key);
```

*Good:*

```
int dict_get_int32 (dict_t *this, char *key, int32_t *val);
```

- **Always use the ‘n’ versions of string functions**

Unless impossible, use the length-limited versions of the string functions.

*Bad:*

```
strcpy (entry_path, real_path);
```

*Good:*

```
strncpy (entry_path, real_path, entry_path_len);
```

- **No dead or commented code**

There must be no dead code (code to which control can never be passed) or commented out code in the codebase.

- **Only one unwind and return per function**

There must be only one exit out of a function. UNWIND and return should happen at only point in the function.

- **Keep functions small**

Try to keep functions small. Two to three screenfulls (80 lines per screen) is considered a reasonable limit. If a function is very long, try splitting it into many little helper functions.

*Example for a helper function:*

```
static int
same_owner (posix_lock_t *l1, posix_lock_t *l2)
{
    return ((l1->client_pid == l2->client_pid) &&
            (l1->transport == l2->transport));
}
```

## A skeleton fop function

This is the recommended template for any fop. In the beginning come the initializations. After that, the ‘success’ control flow should be linear. Any error conditions should cause a `goto` to a single point, `err`. At that point, the code should detect the error that has occurred and do appropriate cleanup.

```
int32_t
sample_fop (call_frame_t *frame,
            xlator_t *this,
            ...)
{
    char *var1 = NULL;
    int32_t op_ret = 0;
    int32_t op_errno = 0;
    DIR *dir = NULL;
    struct posix_fd *pfd = NULL;

    VALIDATE(frame);
    VALIDATE(this);

    /* other validations */

    dir = opendir (...);

    if (dir == NULL) {
        op_ret = -1;
        op_errno = errno;
        gf_log (this->name, GF_LOG_ERROR,
                "opendir failed on %s (%s)", loc->path, strerror (op_errno));
        goto err;
    }

    /* another system call */
    if (...) {
        op_ret = -1;
        op_errno = ENOMEM;
        gf_log (this->name, GF_LOG_ERROR,
                "out of memory :(");
        goto err;
    }

    /* ... */

err:
    if (op_ret == -1) {
```

```

    /* check for all the cleanup that needs to be
       done */

    if (dir) {
        closedir (dir);
        dir = NULL;
    }
    if (pfd) {
        if (pfd->path)
            FREE (pfd->path);
        FREE (pfd);
        pfd = NULL;
    }
}

STACK_UNWIND (frame, op_ret, op_errno, fd);
return 0;
}

```

---

I hereby agree, upon the pain of termination of employment, to abide by the coding standards above.

Signed,

1. Anand V. Avati
2. Basavanagowda Kanur
3. Krishna Srinivas
4. Raghavendra G
5. Vikas Gorur