

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-03/msg01156.html>

- *From:* Harvey Harrison <harvey.harrison@xxxxxxxxxx>
 - *Date:* Mon, 03 Mar 2008 18:00:59 -0800
-

__FUNCTION__ is gcc-specific, use __func__

Signed-off-by: Harvey Harrison <harvey.harrison@xxxxxxxxxx>

```
---
fs/ext4/balloc.c | 14 +++++-----
fs/ext4/ext4_jbd2.c | 12 +++++-----
fs/ext4/extents.c | 2 +-
fs/ext4/ialloc.c | 10 +++-----
fs/ext4/inode.c | 8 +++-----
fs/ext4/malloc.c | 20 ++++++-----
fs/ext4/namei.c | 26 ++++++-----
fs/ext4/resize.c | 70 ++++++-----
fs/ext4/super.c | 16 +++++-----
fs/ext4/xattr.c | 16 +++++-----
10 files changed, 97 insertions(+), 97 deletions(-)
```

```
diff --git a/fs/ext4/balloc.c b/fs/ext4/balloc.c
```

```
index 0737e05..6188336 100644
```

```
--- a/fs/ext4/balloc.c
```

```
+++ b/fs/ext4/balloc.c
```

```
@@ -59,7 +59,7 @@ unsigned ext4_init_block_bitmap(struct super_block *sb, struct buffer_head *bh,
```

```
/* If checksum is bad mark all blocks used to prevent allocation
```

```
* essentially implementing a per-group read-only flag. */
```

```
if (!ext4_group_desc_csum_verify(sbi, block_group, gdp)) {
```

```
- ext4_error(sb, __FUNCTION__,
```

```
+ ext4_error(sb, __func__,
```

```
"Checksum bad for group %lu\n", block_group);
```

```
gdp->bg_free_blocks_count = 0;
```

```
gdp->bg_free_inodes_count = 0;
```

```
@@ -235,7 +235,7 @@ static int ext4_valid_block_bitmap(struct super_block *sb,
```

```
return 1;
```

```
err_out:
```

```
- ext4_error(sb, __FUNCTION__,
```

```
+ ext4_error(sb, __func__,
```

```
"Invalid block bitmap - "
```

```
"block_group = %d, block = %llu",
```

```
block_group, bitmap_blk);
```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```

@@ -264,7 +264,7 @@ read_block_bitmap(struct super_block *sb, ext4_group_t block_group)
bitmap_blk = ext4_block_bitmap(sb, desc);
bh = sb_getblk(sb, bitmap_blk);
if (unlikely(!bh)) {
- ext4_error(sb, __FUNCTION__,
+ ext4_error(sb, __func__,
"Cannot read block bitmap - "
"block_group = %d, block_bitmap = %llu",
(int)block_group, (unsigned long long)bitmap_blk);
@@ -281,7 +281,7 @@ read_block_bitmap(struct super_block *sb, ext4_group_t block_group)
}
if (bh_submit_read(bh) < 0) {
put_bh(bh);
- ext4_error(sb, __FUNCTION__,
+ ext4_error(sb, __func__,
"Cannot read block bitmap - "
"block_group = %d, block_bitmap = %llu",
(int)block_group, (unsigned long long)bitmap_blk);
@@ -360,7 +360,7 @@ restart:
BUG();
}
#define rsv_window_dump(root, verbose) \
- __rsv_window_dump((root), (verbose), __FUNCTION__)
+ __rsv_window_dump((root), (verbose), __func__)
#else
#define rsv_window_dump(root, verbose) do { } while (0)
#endif
@@ -740,7 +740,7 @@ do_more:
if (!ext4_clear_bit_atomic(sb_bgl_lock(sbi, block_group),
bit + i, bitmap_bh->b_data)) {
jbd_unlock_bh_state(bitmap_bh);
- ext4_error(sb, __FUNCTION__,
+ ext4_error(sb, __func__,
"bit already cleared for block %llu",
(ext4_fsblk_t)(block + i));
jbd_lock_bh_state(bitmap_bh);
@@ -1798,7 +1798,7 @@ allocated:
if (ext4_test_bit(grp_alloc_blk+i,
bh2jh(bitmap_bh)->b_committed_data)) {
printk("%s: block was unexpectedly set in "
- "b_committed_data\n", __FUNCTION__);
+ "b_committed_data\n", __func__);
}
}
}
diff --git a/fs/ext4/ext4_jbd2.c b/fs/ext4/ext4_jbd2.c
index d6afe4e..5a1a724 100644
--- a/fs/ext4/ext4_jbd2.c
+++ b/fs/ext4/ext4_jbd2.c
@@ -9,7 +9,7 @@ int __ext4_journal_get_undo_access(const char *where, handle_t *handle,
{

```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```
int err = jbd2_journal_get_undo_access(handle, bh);
if (err)
- ext4_journal_abort_handle(where, __FUNCTION__, bh, handle,err);
+ ext4_journal_abort_handle(where, __func__, bh, handle,err);
return err;
}

@@ -18,7 +18,7 @@ int __ext4_journal_get_write_access(const char *where, handle_t *handle,
{
int err = jbd2_journal_get_write_access(handle, bh);
if (err)
- ext4_journal_abort_handle(where, __FUNCTION__, bh, handle,err);
+ ext4_journal_abort_handle(where, __func__, bh, handle,err);
return err;
}

@@ -27,7 +27,7 @@ int __ext4_journal_forget(const char *where, handle_t *handle,
{
int err = jbd2_journal_forget(handle, bh);
if (err)
- ext4_journal_abort_handle(where, __FUNCTION__, bh, handle,err);
+ ext4_journal_abort_handle(where, __func__, bh, handle,err);
return err;
}

@@ -36,7 +36,7 @@ int __ext4_journal_revoke(const char *where, handle_t *handle,
{
int err = jbd2_journal_revoke(handle, blocknr, bh);
if (err)
- ext4_journal_abort_handle(where, __FUNCTION__, bh, handle,err);
+ ext4_journal_abort_handle(where, __func__, bh, handle,err);
return err;
}

@@ -45,7 +45,7 @@ int __ext4_journal_get_create_access(const char *where,
{
int err = jbd2_journal_get_create_access(handle, bh);
if (err)
- ext4_journal_abort_handle(where, __FUNCTION__, bh, handle,err);
+ ext4_journal_abort_handle(where, __func__, bh, handle,err);
return err;
}

@@ -54,6 +54,6 @@ int __ext4_journal_dirty_metadata(const char *where,
{
int err = jbd2_journal_dirty_metadata(handle, bh);
if (err)
- ext4_journal_abort_handle(where, __FUNCTION__, bh, handle,err);
+ ext4_journal_abort_handle(where, __func__, bh, handle,err);
return err;
}
}
```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```
diff --git a/fs/ext4/extents.c b/fs/ext4/extents.c
index 9ae6e67..00823ac 100644
--- a/fs/ext4/extents.c
+++ b/fs/ext4/extents.c
@@ -308,7 +308,7 @@ corrupted:
}
```

```
#define ext4_ext_check_header(inode, eh, depth) \
- __ext4_ext_check_header(__FUNCTION__, inode, eh, depth)
+ __ext4_ext_check_header(__func__, inode, eh, depth)
```

```
#ifdef EXT_DEBUG
```

```
static void ext4_ext_show_path(struct inode *inode, struct ext4_ext_path *path)
```

```
diff --git a/fs/ext4/ialloc.c b/fs/ext4/ialloc.c
```

```
index 8036b9b..45f39b8 100644
```

```
--- a/fs/ext4/ialloc.c
```

```
+++ b/fs/ext4/ialloc.c
```

```
@@ -75,7 +75,7 @@ unsigned ext4_init_inode_bitmap(struct super_block *sb, struct buffer_head *bh,
```

```
/* If checksum is bad mark all blocks and inodes use to prevent
```

```
* allocation, essentially implementing a per-group read-only flag. */
```

```
if (!ext4_group_desc_csum_verify(sbi, block_group, gdp)) {
```

```
- ext4_error(sb, __FUNCTION__, "Checksum bad for group %lu\n",
```

```
+ ext4_error(sb, __func__, "Checksum bad for group %lu\n",
```

```
block_group);
```

```
gdp->bg_free_blocks_count = 0;
```

```
gdp->bg_free_inodes_count = 0;
```

```
@@ -588,7 +588,7 @@ got:
```

```
ino++;
```

```
if ((group == 0 && ino < EXT4_FIRST_INO(sb)) ||
```

```
ino > EXT4_INODES_PER_GROUP(sb)) {
```

```
- ext4_error(sb, __FUNCTION__,
```

```
+ ext4_error(sb, __func__,
```

```
"reserved inode or inode > inodes count - "
```

```
"block_group = %lu, inode=%lu", group,
```

```
ino + group * EXT4_INODES_PER_GROUP(sb));
```

```
@@ -796,7 +796,7 @@ struct inode *ext4_orphan_get(struct super_block *sb, unsigned long ino)
```

```
/* Error cases - e2fsck has already cleaned up for us */
```

```
if (ino > max_ino) {
```

```
- ext4_warning(sb, __FUNCTION__,
```

```
+ ext4_warning(sb, __func__,
```

```
"bad orphan ino %lu! e2fsck was run?", ino);
```

```
goto error;
```

```
}
```

```
@@ -805,7 +805,7 @@ struct inode *ext4_orphan_get(struct super_block *sb, unsigned long ino)
```

```
bit = (ino - 1) % EXT4_INODES_PER_GROUP(sb);
```

```
bitmap_bh = read_inode_bitmap(sb, block_group);
```

```
if (!bitmap_bh) {
```

```
- ext4_warning(sb, __FUNCTION__,
```

```
+ ext4_warning(sb, __func__,
```

```
"inode bitmap error for orphan %lu", ino);
```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```
goto error;
}
@@ -830,7 +830,7 @@ iget_failed:
err = PTR_ERR(inode);
inode = NULL;
bad_orphan:
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"bad orphan inode %lu! e2fsck was run?", ino);
printk(KERN_NOTICE "ext4_test_bit(bit=%d, block=%llu) = %d\n",
bit, (unsigned long long)bitmap_bh->b_blocknr,
diff --git a/fs/ext4/inode.c b/fs/ext4/inode.c
index 945cbf6..7351a45 100644
--- a/fs/ext4/inode.c
+++ b/fs/ext4/inode.c
@@ -93,7 +93,7 @@ int ext4_forget(handle_t *handle, int is_metadata, struct inode *inode,
BUFFER_TRACE(bh, "call ext4_journal_revoke");
err = ext4_journal_revoke(handle, blocknr, bh);
if (err)
- ext4_abort(inode->i_sb, __FUNCTION__,
+ ext4_abort(inode->i_sb, __func__,
"error %d when attempting revoke", err);
BUFFER_TRACE(bh, "exit");
return err;
@@ -1230,7 +1230,7 @@ int ext4_journal_dirty_data(handle_t *handle, struct buffer_head *bh)
{
int err = jbd2_journal_dirty_data(handle, bh);
if (err)
- ext4_journal_abort_handle(__FUNCTION__, __FUNCTION__,
+ ext4_journal_abort_handle(__func__, __func__,
bh, handle, err);
return err;
}
@@ -3374,7 +3374,7 @@ int ext4_mark_inode_dirty(handle_t *handle, struct inode *inode)
EXT4_I(inode)->i_state |= EXT4_STATE_NO_EXPAND;
if (mnt_count !=
le16_to_cpu(sbi->s_es->s_mnt_count)) {
- ext4_warning(inode->i_sb, __FUNCTION__,
+ ext4_warning(inode->i_sb, __func__,
"Unable to expand inode %lu. Delete"
" some EAs or run e2fsck.",
inode->i_ino);
@@ -3415,7 +3415,7 @@ void ext4_dirty_inode(struct inode *inode)
current_handle->h_transaction != handle->h_transaction) {
/* This task has a transaction open against a different fs */
printk(KERN_EMERG "%s: transactions do not match!\n",
- __FUNCTION__);
+ __func__);
} else {
jbd_debug(5, "marking dirty. outer handle=%p\n",
current_handle);
```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```
diff --git a/fs/ext4/malloc.c b/fs/ext4/malloc.c
index ef97f19..e61cc49 100644
--- a/fs/ext4/malloc.c
+++ b/fs/ext4/malloc.c
@@ -736,7 +736,7 @@ static void mb_free_blocks_double(struct inode *inode, struct ext4_buddy *e4b,
blocknr +=
le32_to_cpu(EXT4_SB(sb)->s_es->s_first_data_block);

- ext4_error(sb, __FUNCTION__, "double-free of inode"
+ ext4_error(sb, __func__, "double-free of inode"
" %lu's block %llu(bit %u in group %lu)\n",
inode ? inode->i_ino : 0, blocknr,
first + i, e4b->bd_group);
@@ -908,7 +908,7 @@ static int __mb_check_buddy(struct ext4_buddy *e4b, char *file,
}
#undef MB_CHECK_ASSERT
#define mb_check_buddy(e4b) __mb_check_buddy(e4b, \
- __FILE__, __FUNCTION__, __LINE__)
+ __FILE__, __func__, __LINE__)
#else
#define mb_check_buddy(e4b)
#endif
@@ -982,7 +982,7 @@ static void ext4_mb_generate_buddy(struct super_block *sb,
grp->bb_fragments = fragments;

if (free != grp->bb_free) {
- ext4_error(sb, __FUNCTION__,
+ ext4_error(sb, __func__,
"EXT4-fs: group %lu: %u blocks in bitmap, %u in gd\n",
group, free, grp->bb_free);
/*
@@ -1367,7 +1367,7 @@ static int mb_free_blocks(struct inode *inode, struct ext4_buddy *e4b,
blocknr +=
le32_to_cpu(EXT4_SB(sb)->s_es->s_first_data_block);

- ext4_error(sb, __FUNCTION__, "double-free of inode"
+ ext4_error(sb, __func__, "double-free of inode"
" %lu's block %llu(bit %u in group %lu)\n",
inode ? inode->i_ino : 0, blocknr, block,
e4b->bd_group);
@@ -1848,7 +1848,7 @@ static void ext4_mb_complex_scan_group(struct ext4_allocation_context *ac,
* free blocks even though group info says we
* we have free blocks
*/
- ext4_error(sb, __FUNCTION__, "%d free blocks as per "
+ ext4_error(sb, __func__, "%d free blocks as per "
"group info. But bitmap says 0\n",
free);
break;
@@ -1857,7 +1857,7 @@ static void ext4_mb_complex_scan_group(struct ext4_allocation_context *ac,
mb_find_extent(e4b, 0, i, ac->ac_g_ex.fe_len, &ex);
```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```

BUG_ON(ex.fe_len <= 0);
if (free < ex.fe_len) {
- ext4_error(sb, __FUNCTION__, "%d free blocks as per "
+ ext4_error(sb, __func__, "%d free blocks as per "
"group info. But got %d blocks\n",
free, ex.fe_len);
/*
@@ -3078,7 +3078,7 @@ static int ext4_mb_mark_diskspace_used(struct ext4_allocation_context *ac,
in_range(block, ext4_inode_table(sb, gdp),
EXT4_SB(sb)->s_itb_per_group)) {

- ext4_error(sb, __FUNCTION__,
+ ext4_error(sb, __func__,
"Allocating block in system zone - block = %llu",
block);
}
@@ -3797,7 +3797,7 @@ static int ext4_mb_release_inode_pa(struct ext4_buddy *e4b,
pa, (unsigned long) pa->pa_lstart,
(unsigned long) pa->pa_pstart,
(unsigned long) pa->pa_len);
- ext4_error(sb, __FUNCTION__, "free %u, pa_free %u\n",
+ ext4_error(sb, __func__, "free %u, pa_free %u\n",
free, pa->pa_free);
/*
* pa is already deleted so we use the value obtained
@@ -4497,7 +4497,7 @@ void ext4_mb_free_blocks(handle_t *handle, struct inode *inode,
if (block < le32_to_cpu(es->s_first_data_block) ||
block + count < block ||
block + count > ext4_blocks_count(es)) {
- ext4_error(sb, __FUNCTION__,
+ ext4_error(sb, __func__,
"Freeing blocks not in datazone - "
"block = %lu, count = %lu", block, count);
goto error_return;
@@ -4538,7 +4538,7 @@ do_more:
in_range(block + count - 1, ext4_inode_table(sb, gdp),
EXT4_SB(sb)->s_itb_per_group)) {

- ext4_error(sb, __FUNCTION__,
+ ext4_error(sb, __func__,
"Freeing blocks in system zone - "
"Block = %lu, count = %lu", block, count);
}
diff --git a/fs/ext4/namei.c b/fs/ext4/namei.c
index 28aa2ed..5b37712 100644
--- a/fs/ext4/namei.c
+++ b/fs/ext4/namei.c
@@ -348,7 +348,7 @@ dx_probe(struct dentry *dentry, struct inode *dir,
if (root->info.hash_version != DX_HASH_TEA &&
root->info.hash_version != DX_HASH_HALF_MD4 &&
root->info.hash_version != DX_HASH_LEGACY) {

```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```
- ext4_warning(dir->i_sb, __FUNCTION__,
+ ext4_warning(dir->i_sb, __func__,
"Unrecognised inode hash code %d",
root->info.hash_version);
brelse(bh);
@@ -362,7 +362,7 @@ dx_probe(struct dentry *dentry, struct inode *dir,
hash = hinfo->hash;

if (root->info.unused_flags & 1) {
- ext4_warning(dir->i_sb, __FUNCTION__,
+ ext4_warning(dir->i_sb, __func__,
"Unimplemented inode hash flags: %#06x",
root->info.unused_flags);
brelse(bh);
@@ -371,7 +371,7 @@ dx_probe(struct dentry *dentry, struct inode *dir,
}

if ((indirect = root->info.indirect_levels) > 1) {
- ext4_warning(dir->i_sb, __FUNCTION__,
+ ext4_warning(dir->i_sb, __func__,
"Unimplemented inode hash depth: %#06x",
root->info.indirect_levels);
brelse(bh);
@@ -384,7 +384,7 @@ dx_probe(struct dentry *dentry, struct inode *dir,

if (dx_get_limit(entries) != dx_root_limit(dir,
root->info.info_length)) {
- ext4_warning(dir->i_sb, __FUNCTION__,
+ ext4_warning(dir->i_sb, __func__,
"dx entry: limit != root limit");
brelse(bh);
*err = ERR_BAD_DX_DIR;
@@ -396,7 +396,7 @@ dx_probe(struct dentry *dentry, struct inode *dir,
{
count = dx_get_count(entries);
if (!count || count > dx_get_limit(entries)) {
- ext4_warning(dir->i_sb, __FUNCTION__,
+ ext4_warning(dir->i_sb, __func__,
"dx entry: no count or count > limit");
brelse(bh);
*err = ERR_BAD_DX_DIR;
@@ -441,7 +441,7 @@ dx_probe(struct dentry *dentry, struct inode *dir,
goto fail2;
at = entries = ((struct dx_node *) bh->b_data)->entries;
if (dx_get_limit(entries) != dx_node_limit (dir)) {
- ext4_warning(dir->i_sb, __FUNCTION__,
+ ext4_warning(dir->i_sb, __func__,
"dx entry: limit != node limit");
brelse(bh);
*err = ERR_BAD_DX_DIR;
@@ -457,7 +457,7 @@ fail2:
```

```

}
fail:
if (*err == ERR_BAD_DX_DIR)
- ext4_warning(dir->i_sb, __FUNCTION__,
+ ext4_warning(dir->i_sb, __func__,
"Corrupt dir inode %ld, running e2fsck is "
"recommended.", dir->i_ino);
return NULL;
@@ -914,7 +914,7 @@ restart:
wait_on_buffer(bh);
if (!buffer_uptodate(bh)) {
/* read error, skip block & hope for the best */
- ext4_error(sb, __FUNCTION__, "reading directory #%lu "
+ ext4_error(sb, __func__, "reading directory #%lu "
"offset %lu", dir->i_ino,
(unsigned long)block);
brelse(bh);
@@ -1007,7 +1007,7 @@ static struct buffer_head * ext4_dx_find_entry(struct dentry *dentry,
retval = ext4_htree_next_block(dir, hash, frame,
frames, NULL);
if (retval < 0) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"error reading index page in directory #%lu",
dir->i_ino);
*err = retval;
@@ -1532,7 +1532,7 @@ static int ext4_dx_add_entry(handle_t *handle, struct dentry *dentry,

if (levels && (dx_get_count(frames->entries) ==
dx_get_limit(frames->entries))) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Directory index full!");
err = -ENOSPC;
goto cleanup;
@@ -1860,11 +1860,11 @@ static int empty_dir (struct inode * inode)
if (inode->i_size < EXT4_DIR_REC_LEN(1) + EXT4_DIR_REC_LEN(2) ||
!(bh = ext4_bread (NULL, inode, 0, 0, &err))) {
if (err)
- ext4_error(inode->i_sb, __FUNCTION__,
+ ext4_error(inode->i_sb, __func__,
"error %d reading directory #%lu offset 0",
err, inode->i_ino);
else
- ext4_warning(inode->i_sb, __FUNCTION__,
+ ext4_warning(inode->i_sb, __func__,
"bad directory (dir #%lu) - no data block",
inode->i_ino);
return 1;
@@ -1893,7 +1893,7 @@ static int empty_dir (struct inode * inode)
offset >> EXT4_BLOCK_SIZE_BITS(sb), 0, &err);

```

```

if (!bh) {
if (err)
- ext4_error(sb, __FUNCTION__,
+ ext4_error(sb, __func__,
"error %d reading directory"
" #%lu offset %lu",
err, inode->i_ino, offset);
diff --git a/fs/ext4/resize.c b/fs/ext4/resize.c
index e29efa0..b79300a 100644
--- a/fs/ext4/resize.c
+++ b/fs/ext4/resize.c
@@ -50,63 +50,63 @@ static int verify_group_input(struct super_block *sb,

ext4_get_group_no_and_offset(sb, start, NULL, &offset);
if (group != sbi->s_groups_count)
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Cannot add at group %u (only %lu groups)",
input->group, sbi->s_groups_count);
else if (offset != 0)
- ext4_warning(sb, __FUNCTION__, "Last group not full");
+ ext4_warning(sb, __func__, "Last group not full");
else if (input->reserved_blocks > input->blocks_count / 5)
- ext4_warning(sb, __FUNCTION__, "Reserved blocks too high (%u)",
+ ext4_warning(sb, __func__, "Reserved blocks too high (%u)",
input->reserved_blocks);
else if (free_blocks_count < 0)
- ext4_warning(sb, __FUNCTION__, "Bad blocks count %u",
+ ext4_warning(sb, __func__, "Bad blocks count %u",
input->blocks_count);
else if (!(bh = sb_bread(sb, end - 1)))
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Cannot read last block (%llu)",
end - 1);
else if (outside(input->block_bitmap, start, end))
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Block bitmap not in group (block %llu)",
(unsigned long long)input->block_bitmap);
else if (outside(input->inode_bitmap, start, end))
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Inode bitmap not in group (block %llu)",
(unsigned long long)input->inode_bitmap);
else if (outside(input->inode_table, start, end) ||
outside(itend - 1, start, end))
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Inode table not in group (blocks %llu-%llu)",
(unsigned long long)input->inode_table, itend - 1);

```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```

else if (input->inode_bitmap == input->block_bitmap)
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Block bitmap same as inode bitmap (%llu)",
(unsigned long long)input->block_bitmap);
else if (inside(input->block_bitmap, input->inode_table, itend))
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Block bitmap (%llu) in inode table (%llu-%llu)",
(unsigned long long)input->block_bitmap,
(unsigned long long)input->inode_table, itend - 1);
else if (inside(input->inode_bitmap, input->inode_table, itend))
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Inode bitmap (%llu) in inode table (%llu-%llu)",
(unsigned long long)input->inode_bitmap,
(unsigned long long)input->inode_table, itend - 1);
else if (inside(input->block_bitmap, start, metaend))
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Block bitmap (%llu) in GDT table"
" (%llu-%llu)",
(unsigned long long)input->block_bitmap,
start, metaend - 1);
else if (inside(input->inode_bitmap, start, metaend))
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Inode bitmap (%llu) in GDT table"
" (%llu-%llu)",
(unsigned long long)input->inode_bitmap,
start, metaend - 1);
else if (inside(input->inode_table, start, metaend) ||
inside(itend - 1, start, metaend))
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Inode table (%llu-%llu) overlaps"
"GDT table (%llu-%llu)",
(unsigned long long)input->inode_table,
@@ -368,7 +368,7 @@ static int verify_reserved_gdb(struct super_block *sb,
while ((grp = ext4_list_backups(sb, &three, &five, &seven)) < end) {
if (le32_to_cpu(*p++) !=
grp * EXT4_BLOCKS_PER_GROUP(sb) + blk){
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"reserved GDT %llu"
" missing grp %d (%llu)",
blk, grp,
@@ -424,7 +424,7 @@ static int add_new_gdb(handle_t *handle, struct inode *inode,
*/
if (EXT4_SB(sb)->s_sbh->b_blocknr !=
le32_to_cpu(EXT4_SB(sb)->s_es->s_first_data_block)) {

```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"won't resize using backup superblock at %llu",
(unsigned long long)EXT4_SB(sb)->s_sbh->b_blocknr);
return -EPERM;
@@ -448,7 +448,7 @@ static int add_new_gdb(handle_t *handle, struct inode *inode,

data = (__le32 *)dind->b_data;
if (le32_to_cpu(data[gdb_num % EXT4_ADDR_PER_BLOCK(sb)]) != gdblock) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"new group %u GDT block %llu not reserved",
input->group, gdblock);
err = -EINVAL;
@@ -472,7 +472,7 @@ static int add_new_gdb(handle_t *handle, struct inode *inode,
GFP_KERNEL);
if (!n_group_desc) {
err = -ENOMEM;
- ext4_warning (sb, __FUNCTION__,
+ ext4_warning (sb, __func__,
"not enough memory for %lu groups", gdb_num + 1);
goto exit_inode;
}
@@ -571,7 +571,7 @@ static int reserve_backup_gdb(handle_t *handle, struct inode *inode,
/* Get each reserved primary GDT block and verify it holds backups */
for (res = 0; res < reserved_gdb; res++, blk++) {
if (le32_to_cpu(*data) != blk) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"reserved block %llu"
" not at offset %ld",
blk,
@@ -715,7 +715,7 @@ static void update_backups(struct super_block *sb,
*/
exit_err:
if (err) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"can't update backup for group %lu (err %d), "
"forcing fsck on next reboot", group, err);
sbi->s_mount_state &= ~EXT4_VALID_FS;
@@ -755,33 +755,33 @@ int ext4_group_add(struct super_block *sb, struct ext4_new_group_data *input)

if (gdb_off == 0 && !EXT4_HAS_RO_COMPAT_FEATURE(sb,
EXT4_FEATURE_RO_COMPAT_SPARSE_SUPER)) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Can't resize non-sparse filesystem further");
return -EPERM;
}
}
```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```
if (ext4_blocks_count(es) + input->blocks_count <
ext4_blocks_count(es)) {
- ext4_warning(sb, __FUNCTION__, "blocks_count overflow\n");
+ ext4_warning(sb, __func__, "blocks_count overflow\n");
return -EINVAL;
}

if (le32_to_cpu(es->s_inodes_count) + EXT4_INODES_PER_GROUP(sb) <
le32_to_cpu(es->s_inodes_count)) {
- ext4_warning(sb, __FUNCTION__, "inodes_count overflow\n");
+ ext4_warning(sb, __func__, "inodes_count overflow\n");
return -EINVAL;
}

if (reserved_gdb || gdb_off == 0) {
if (!EXT4_HAS_COMPAT_FEATURE(sb,
EXT4_FEATURE_COMPAT_RESIZE_INODE)){
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"No reserved GDT blocks, can't resize");
return -EPERM;
}
inode = ext4_iget(sb, EXT4_RESIZE_INO);
if (IS_ERR(inode)) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"Error opening resize inode");
return PTR_ERR(inode);
}
@@ -810,7 +810,7 @@ int ext4_group_add(struct super_block *sb, struct ext4_new_group_data *input)

lock_super(sb);
if (input->group != sbi->s_groups_count) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"multiple resizers run on filesystem!");
err = -EBUSY;
goto exit_journal;
@@ -977,13 +977,13 @@ int ext4_group_extend(struct super_block *sb, struct ext4_super_block *es,
" too large to resize to %llu blocks safely\n",
sb->s_id, n_blocks_count);
if (sizeof(sector_t) < 8)
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"CONFIG_LBD not enabled\n");
return -EINVAL;
}

if (n_blocks_count < o_blocks_count) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```
"can't shrink FS – resize aborted");
return -EBUSY;
}
@@ -992,7 +992,7 @@ int ext4_group_extend(struct super_block *sb, struct ext4_super_block *es,
ext4_get_group_no_and_offset(sb, o_blocks_count, NULL, &last);

if (last == 0) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"need to use ext2online to resize further");
return -EPERM;
}
@@ -1000,7 +1000,7 @@ int ext4_group_extend(struct super_block *sb, struct ext4_super_block *es,
add = EXT4_BLOCKS_PER_GROUP(sb) - last;

if (o_blocks_count + add < o_blocks_count) {
- ext4_warning(sb, __FUNCTION__, "blocks_count overflow");
+ ext4_warning(sb, __func__, "blocks_count overflow");
return -EINVAL;
}

@@ -1008,7 +1008,7 @@ int ext4_group_extend(struct super_block *sb, struct ext4_super_block *es,
add = n_blocks_count - o_blocks_count;

if (o_blocks_count + add < n_blocks_count)
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"will only finish group (%llu"
" blocks, %u new)",
o_blocks_count + add, add);
@@ -1016,7 +1016,7 @@ int ext4_group_extend(struct super_block *sb, struct ext4_super_block *es,
/* See if the device is actually as big as what was requested */
bh = sb_bread(sb, o_blocks_count + add - 1);
if (!bh) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"can't read last block, resize aborted");
return -ENOSPC;
}
@@ -1028,13 +1028,13 @@ int ext4_group_extend(struct super_block *sb, struct ext4_super_block *es,
handle = ext4_journal_start_sb(sb, 3);
if (IS_ERR(handle)) {
err = PTR_ERR(handle);
- ext4_warning(sb, __FUNCTION__, "error %d on journal start",err);
+ ext4_warning(sb, __func__, "error %d on journal start",err);
goto exit_put;
}

lock_super(sb);
if (o_blocks_count != ext4_blocks_count(es)) {
- ext4_warning(sb, __FUNCTION__,
```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```

+ ext4_warning(sb, __func__,
"multiple resizers run on filesystem!");
unlock_super(sb);
ext4_journal_stop(handle);
@@ -1044,7 +1044,7 @@ int ext4_group_extend(struct super_block *sb, struct ext4_super_block *es,

if ((err = ext4_journal_get_write_access(handle,
EXT4_SB(sb)->s_sb))) {
- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"error %d on journal write access", err);
unlock_super(sb);
ext4_journal_stop(handle);
diff --git a/fs/ext4/super.c b/fs/ext4/super.c
index 13383ba..0baf5a1 100644
--- a/fs/ext4/super.c
+++ b/fs/ext4/super.c
@@ -135,7 +135,7 @@ handle_t *ext4_journal_start_sb(struct super_block *sb, int nblocks)
* take the FS itself readonly cleanly. */
journal = EXT4_SB(sb)->s_journal;
if (is_journal_aborted(journal)) {
- ext4_abort(sb, __FUNCTION__,
+ ext4_abort(sb, __func__,
"Detected aborted journal");
return ERR_PTR(-EROFS);
}
@@ -355,7 +355,7 @@ void ext4_update_dynamic_rev(struct super_block *sb)
if (le32_to_cpu(es->s_rev_level) > EXT4_GOOD_OLD_REV)
return;

- ext4_warning(sb, __FUNCTION__,
+ ext4_warning(sb, __func__,
"updating to rev %d because of new feature flag, "
"running e2fsck is recommended",
EXT4_DYNAMIC_REV);
@@ -1510,7 +1510,7 @@ static int ext4_check_descriptors(struct super_block *sb)
return 0;
}
if (!ext4_group_desc_csum_verify(sbi, i, gdp)) {
- ext4_error(sb, __FUNCTION__,
+ ext4_error(sb, __func__,
"Checksum for group %lu failed (%u!=%u)\n",
i, le16_to_cpu(ext4_group_desc_csum(sbi, i,
gdp)), le16_to_cpu(gdp->bg_checksum));
@@ -1604,7 +1604,7 @@ static void ext4_orphan_cleanup (struct super_block * sb,
if (inode->i_nlink) {
printk(KERN_DEBUG
"%s: truncating inode %lu to %Ld bytes\n",
- __FUNCTION__, inode->i_ino, inode->i_size);
+ __func__, inode->i_ino, inode->i_size);
jbd_debug(2, "truncating inode %lu to %Ld bytes\n",

```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```

inode->i_ino, inode->i_size);
ext4_truncate(inode);
@@ -1612,7 +1612,7 @@ static void ext4_orphan_cleanup (struct super_block * sb,
} else {
printk(KERN_DEBUG
"%s: deleting unreferenced inode %lu\n",
- __FUNCTION__, inode->i_ino);
+ __func__, inode->i_ino);
jbd_debug(2, "deleting unreferenced inode %lu\n",
inode->i_ino);
nr_orphans++;
@@ -2698,9 +2698,9 @@ static void ext4_clear_journal_err(struct super_block * sb,
char nbuf[16];

errstr = ext4_decode_error(sb, j_errno, nbuf);
- ext4_warning(sb, __FUNCTION__, "Filesystem error recorded "
+ ext4_warning(sb, __func__, "Filesystem error recorded "
"from previous mount: %s", errstr);
- ext4_warning(sb, __FUNCTION__, "Marking fs in need of "
+ ext4_warning(sb, __func__, "Marking fs in need of "
"filesystem check.");

EXT4_SB(sb)->s_mount_state |= EXT4_ERROR_FS;
@@ -2827,7 +2827,7 @@ static int ext4_remount (struct super_block * sb, int * flags, char * data)
}

if (sbi->s_mount_opt & EXT4_MOUNT_ABORT)
- ext4_abort(sb, __FUNCTION__, "Abort forced by user");
+ ext4_abort(sb, __func__, "Abort forced by user");

sb->s_flags = (sb->s_flags & ~MS_POSIXACL) |
((sbi->s_mount_opt & EXT4_MOUNT_POSIX_ACL) ? MS_POSIXACL : 0);
diff --git a/fs/ext4/xattr.c b/fs/ext4/xattr.c
index d796213..f7bd0cf 100644
--- a/fs/ext4/xattr.c
+++ b/fs/ext4/xattr.c
@@ -225,7 +225,7 @@ ext4_xattr_block_get(struct inode *inode, int name_index, const char *name,
ea_bdebug(bh, "b_count=%d, refcount=%d",
atomic_read(&(bh->b_count)), le32_to_cpu(BHDR(bh)->h_refcount));
if (ext4_xattr_check_block(bh)) {
- bad_block: ext4_error(inode->i_sb, __FUNCTION__,
+ bad_block: ext4_error(inode->i_sb, __func__,
"inode %lu: bad block %llu", inode->i_ino,
EXT4_I(inode)->i_file_acl);
error = -EIO;
@@ -367,7 +367,7 @@ ext4_xattr_block_list(struct inode *inode, char *buffer, size_t buffer_size)
ea_bdebug(bh, "b_count=%d, refcount=%d",
atomic_read(&(bh->b_count)), le32_to_cpu(BHDR(bh)->h_refcount));
if (ext4_xattr_check_block(bh)) {
- ext4_error(inode->i_sb, __FUNCTION__,
+ ext4_error(inode->i_sb, __func__,

```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```
"inode %lu: bad block %llu", inode->i_ino,
EXT4_I(inode)->i_file_acl);
error = -EIO;
@@ -660,7 +660,7 @@ ext4_xattr_block_find(struct inode *inode, struct ext4_xattr_info *i,
atomic_read(&(bs->bh->b_count)),
le32_to_cpu(BHDR(bs->bh)->h_refcount));
if (ext4_xattr_check_block(bs->bh)) {
- ext4_error(sb, __FUNCTION__,
+ ext4_error(sb, __func__,
"inode %lu: bad block %llu", inode->i_ino,
EXT4_I(inode)->i_file_acl);
error = -EIO;
@@ -863,7 +863,7 @@ cleanup_dquot:
goto cleanup;

bad_block:
- ext4_error(inode->i_sb, __FUNCTION__,
+ ext4_error(inode->i_sb, __func__,
"inode %lu: bad block %llu", inode->i_ino,
EXT4_I(inode)->i_file_acl);
goto cleanup;
@@ -1166,7 +1166,7 @@ retry:
if (!bh)
goto cleanup;
if (ext4_xattr_check_block(bh)) {
- ext4_error(inode->i_sb, __FUNCTION__,
+ ext4_error(inode->i_sb, __func__,
"inode %lu: bad block %llu", inode->i_ino,
EXT4_I(inode)->i_file_acl);
error = -EIO;
@@ -1341,14 +1341,14 @@ ext4_xattr_delete_inode(handle_t *handle, struct inode *inode)
goto cleanup;
bh = sb_bread(inode->i_sb, EXT4_I(inode)->i_file_acl);
if (!bh) {
- ext4_error(inode->i_sb, __FUNCTION__,
+ ext4_error(inode->i_sb, __func__,
"inode %lu: block %llu read error", inode->i_ino,
EXT4_I(inode)->i_file_acl);
goto cleanup;
}
if (BHDR(bh)->h_magic != cpu_to_le32(EXT4_XATTR_MAGIC) ||
BHDR(bh)->h_blocks != cpu_to_le32(1)) {
- ext4_error(inode->i_sb, __FUNCTION__,
+ ext4_error(inode->i_sb, __func__,
"inode %lu: bad block %llu", inode->i_ino,
EXT4_I(inode)->i_file_acl);
goto cleanup;
@@ -1475,7 +1475,7 @@ again:
}
bh = sb_bread(inode->i_sb, ce->e_block);
if (!bh) {
```

[PATCH 03/13] ext4: replace remaining __FUNCTION__ occurrences

```
- ext4_error(inode->i_sb, __FUNCTION__,
+ ext4_error(inode->i_sb, __func__,
"inode %lu: block %lu read error",
inode->i_ino, (unsigned long) ce->e_block);
} else if (le32_to_cpu(BHDR(bh)->h_refcount) >=
```

1.5.4.3.500.g83a2c

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>