Removing Unapproved SPAM Comments Using the MySQL Command Line

Problem: Getting rid of most of 94,500 comment records, most of which are unapproved bot SPAM. The site forgot to put captchas up and had no other spam protection and let the site collect spam for some time, and site bloggers were not prepared to go there every day and remove spam.

The first step was to install captchas to prevent the bleeding (although this doesn't always seem to work nowadays). The second step is to remove the giant tumor of spam comments from the database.

One site helper was asked to clean up the spam and deleted approximately 25,000 comments by hand, using the batch interface (shows 25 comment records at a time). But those were quickly replaced by comment spambots.

```
$ drush sql-query "SELECT COUNT(*) from comment"
94857  <-- number of comments
$ drush sql-query "SELECT COUNT(*) FROM comment WHERE status=0"
94528  <-- number of UNAPPROVED comments (99.9% SPAM)
$ drush sql-query "SELECT COUNT(*) from field_data_comment_body"
94857
$ drush sql-query "SELECT COUNT(*) from field_revision_comment_body"
94857
$ drush sql-query "SELECT COUNT(*) from node_comment_statistics"
2045  <--- number of nodes on the site (not all of which have comments)
```

APPROVE ANY COMMENTS YOU CAN FIND TO APPROVE

Give your site blog editors an opportunity to approve recent comments they haven't yet approved. Chances are if they haven't checked in ages, many approvable comments will be buried in a backlog of SPAM comments and they won't find them. But comments withing recent days or weeks should be viewable, since things are listed by date.

Go to Administration >> Contents >> Comments >> Unapproved Comments (n)

From this page you can also actually select batches of about 25 comments to delete. But with 94,581 unapproved comments (most of which are SPAM), deleting by this method could take days.

BACKUP the DATABASE or TABLES

Backup the database in case anything goes wrong. The whole database is not the best option, since the whole database is large, and restoring it can cause problems.

The easiest approach is probably to use PhPMyAdmin to dump/export the FOUR tables that will be altered by operations in these instructions.

comment

field_data_comment_body
field_revision_comment_body
node_comment_statistics

Using PHPMyAdmin: Make sure to check the "truncate" option in the PHPAdmin table Export if you are using PHPAdmin. It dumps current data and imports older data stored from the database table dump file. Other create options are not necessary.

Using Drush: The following Drush command will also back up the tables (to the top-level site directory, in which you start Drush):

```
$ drush sql-dump --result-file=comment-tables.sql
--tables-list=comment,field_data_comment_body,field_revisio
n_comment_body,node_comment_statistics
```

If you have to restore later, you can use:

```
$mysql -u root -p database_name < table_dumpfile.sql
```

Run Drush Script "remove_ua_comments.drush"

First change to the site root folder.

```
$ cd site_root_folder
$ drush scr remove_ua_comments.drush
```

The Drush script I have written for removing unapproved (SPAM) comments works only for Drupal 7 and for MySQL databases and does the following:

(1) Tests that the site is Drupal 7 and has a MySQL database,
(2) Backs up the four tables that will be altered to an *.SQL script that can be used for recovery,
(3) Removes unapproved comment text from field_data_comment_body table,
(4) Removes unapproved comment text from field_revision_comment_body_table,
(5) Removes unapproved comment records from comment table, and
(6) Rebuilds node_comment_statistics table to reflect the remaining approved comments.

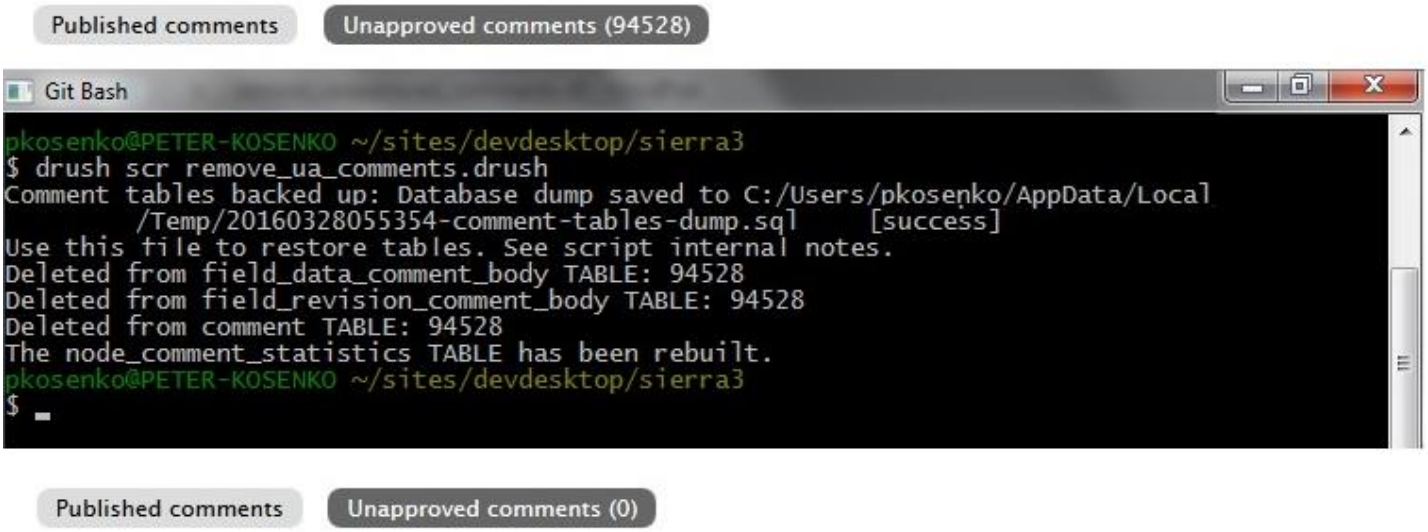
Because of the amount of data that needs to be saved and deleted, each of the queries could take a minute or so.

Results should look something like this (without the Drupal UI snippets at the top and bottom):

```
pkosenko@PETER-KOSENKO ~/sites/devdesktop/sierra3
$ drush scr remove_ua_comments.drush
Comment tables backed up: Database dump saved to
C:/Users/pkosenko/AppData/Local/Temp/20160328055354-comment-tables-dump.sql
                                [success]
Use this file to restore tables. See script internal notes.
```

Deleted from field_data_comment_body TABLE: 94528
Deleted from field_revision_comment_body TABLE: 94528
Deleted from comment TABLE: 94528
The node_comment_statistics TABLE has been rebuilt.
pkosenko@PETER-KOSENKO ~/sites/devdesktop/sierra3
$

The comment approval pane shows before and after number of unapproved comments:



APPENDIX

SQL Queries for removing unapproved comments

Note: These queries could be sent from a Drush script or Drupal 7 module. In other words, they don't have to be done from the MySQL prompt. See RUN DRUSH SCRIPT above. ***The Queries MUST be run in the following order***, since the first two queries depend on joins with the comment table. Comments cannot be deleted from the comment table until after body text is removed from field_data_comment_body and field_revision_comment_body.

```
mysql> Use database_name;
mysql> DELETE FROM field_data_comment_body
    -> USING field_data_comment_body
    -> INNER JOIN comment
    -> ON comment.cid=field_data_comment_body.entity_id
    -> AND comment.status=0;
Query OK, 94581 rows affected (1 min 5.68 sec)  <-- UNAPPROVED COMMENTS
(mostly SPAM)
```

Note that it takes quite a bit of time (computer-wise) and hence you will want to take the site offline to do it.

```
mysql> DELETE FROM field_revision_comment_body
    -> USING field_revision_comment_body
    -> INNER JOIN comment
    -> ON comment.cid=field_revision_comment_body.entity_id
    -> AND comment.status=0;
Query OK, 94581 rows affected (57.67 sec)    <-- UNAPPROVED COMMENTS (mostly
SPAM)

mysql> DELETE FROM comment WHERE status=0;
Query OK, 94581 rows affected (30.90 sec)    <-- UNAPPROVED COMMENTS (mostly
SPAM)

mysql> exit
```

Database comment tables structure

```
(1) comment TABLE

cid = comment id
pid = parent id (the comment to which this is a reply -- 0 = original blog
node)
nid = node id   (the original blog node on which this reply comments)
uid = user id   (the user id -- if not anonymous -- who made this comment)

--
-- Table structure for table `comment`
--

CREATE TABLE IF NOT EXISTS `comment` (
  `cid` int(11) NOT NULL COMMENT 'Primary Key: Unique comment ID.',
  `pid` int(11) NOT NULL DEFAULT '0' COMMENT 'The comment.cid to which this
comment is a reply. If set to 0, this comment is not a reply to an existing
comment.',
  `nid` int(11) NOT NULL DEFAULT '0' COMMENT 'The node.nid to which this
comment is a reply.',
  `uid` int(11) NOT NULL DEFAULT '0' COMMENT 'The users.uid who authored
the comment. If set to 0, this comment was created by an anonymous user.',
  `subject` varchar(64) NOT NULL DEFAULT '' COMMENT 'The comment title.',
  `hostname` varchar(128) NOT NULL DEFAULT '' COMMENT 'The authorâ€™s host
name.',
  `created` int(11) NOT NULL DEFAULT '0' COMMENT 'The time that the comment
was created, as a Unix timestamp.',
  `changed` int(11) NOT NULL DEFAULT '0' COMMENT 'The time that the comment
was last edited, as a Unix timestamp.',
  `status` tinyint(3) unsigned NOT NULL DEFAULT '1' COMMENT 'The published
status of a comment. (0 = Not Published, 1 = Published)',
  `thread` varchar(255) NOT NULL COMMENT 'The vancode representation of
the commentâ€™s place in a thread.',
  `name` varchar(60) DEFAULT NULL COMMENT 'The comment authorâ€™s name.
Uses users.name if the user is logged in, otherwise uses the value typed
into the comment form.',
```

```
  `mail` varchar(64) DEFAULT NULL COMMENT 'The comment authorâ€™s e-mail
address from the comment form, if user is anonymous, and the â€™Anonymous
users may/must leave their contact informationâ€™ setting is turned on.',
  `homepage` varchar(255) DEFAULT NULL COMMENT 'The comment authorâ€™s
home page address from the comment form, if user is anonymous, and the
â€™Anonymous users may/must leave their contact informationâ€™ setting is
turned on.',
  `language` varchar(12) NOT NULL DEFAULT '' COMMENT 'The
languages.language of this comment.'
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8 COMMENT='Stores
comments and associated data.';
```

(2) field_data_comment_body TABLE

```
CREATE TABLE IF NOT EXISTS `field_data_comment_body` (
  `entity_type` varchar(128) NOT NULL DEFAULT '' COMMENT 'The entity type
this data is attached to',
  `bundle` varchar(128) NOT NULL DEFAULT '' COMMENT 'The field instance
bundle to which this row belongs, used when deleting a field instance',
  `deleted` tinyint(4) NOT NULL DEFAULT '0' COMMENT 'A boolean indicating
whether this data item has been deleted',
  `entity_id` int(10) unsigned NOT NULL COMMENT 'The entity id this data
is attached to',
  `revision_id` int(10) unsigned DEFAULT NULL COMMENT 'The entity revision
id this data is attached to, or NULL if the entity type is not versioned',
  `language` varchar(32) NOT NULL DEFAULT '' COMMENT 'The language for this
data item.',
  `delta` int(10) unsigned NOT NULL COMMENT 'The sequence number for this
data item, used for multi-value fields',
  `comment_body_value` longtext,
  `comment_body_format` varchar(255) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Data storage for field 1
(comment_body)';
```

(3) field_revision_comment_body TABLE

```
CREATE TABLE IF NOT EXISTS `field_revision_comment_body` (
  `entity_type` varchar(128) NOT NULL DEFAULT '' COMMENT 'The entity type
this data is attached to',
  `bundle` varchar(128) NOT NULL DEFAULT '' COMMENT 'The field instance
bundle to which this row belongs, used when deleting a field instance',
  `deleted` tinyint(4) NOT NULL DEFAULT '0' COMMENT 'A boolean indicating
whether this data item has been deleted',
  `entity_id` int(10) unsigned NOT NULL COMMENT 'The entity id this data
is attached to',
  `revision_id` int(10) unsigned NOT NULL COMMENT 'The entity revision id
this data is attached to',
  `language` varchar(32) NOT NULL DEFAULT '' COMMENT 'The language for this
data item.',
```

```
  `delta` int(10) unsigned NOT NULL COMMENT 'The sequence number for this
data item, used for multi-value fields',
  `comment_body_value` longtext,
  `comment_body_format` varchar(255) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Revision archive storage for
field 1 (comment_body)';
```

(4) node_comment_statistics TABLE

```
CREATE TABLE IF NOT EXISTS `node_comment_statistics` (
  `nid` int(10) unsigned NOT NULL DEFAULT '0' COMMENT 'The node.nid for
which the statistics are compiled.',
  `cid` int(11) NOT NULL DEFAULT '0' COMMENT 'The comment.cid of the last
comment.',
  `last_comment_timestamp` int(11) NOT NULL DEFAULT '0' COMMENT 'The Unix
timestamp of the last comment that was posted within this node, from
comment.changed.',
  `last_comment_name` varchar(60) DEFAULT NULL COMMENT 'The name of the
latest author to post a comment on this node, from comment.name.',
  `last_comment_uid` int(11) NOT NULL DEFAULT '0' COMMENT 'The user ID of
the latest author to post a comment on this node, from comment.uid.',
  `comment_count` int(10) unsigned NOT NULL DEFAULT '0' COMMENT 'The total
number of comments on this node.'
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Maintains statistics of node
and comments posts to show ...';
```

DRUSH SCRIPT EXPLANATION

I debated using the Drupal 7 delete_comment_multiple() api function, but calling that with a 95,000 element $cids array wasn't going to work (memory crash). And calling it 95 times with a 1,000 element array each time would also end up taking an exorbitant amount of time. (The manual deletion pane actually limits deletions to 25 at a time.) Deleting all the unapproved rows of all three tables by going directly into the database with three queries solves that problem, since mysql has its own memory management for large row deletes, is very fast, and it wouldn't crash the site.

A couple commenters on Drupal Groups questioned whether or not the database queries might leave "orphaned" data in the database. If "orphaned" means inaccessible data that cannot be removed, the answer is NO. I tested.

If parent comments have been approved, then unplublished (unapproved), but their replies have NOT been unpublished, the script will leave the replies (child comments). However, these continue to be visible in the comment pane and can be checked and removed manually if necessary (the red "Zorro child of Doug" below remains visible in the comment listings and can be deleted).

But there would normally not be child comments added to spam, since spam is normally never approved in the first place, and hence unavailable for replies.

Hence it is very unlike that there will be many child comments without parent comments.

It is more likely that legitimate parent comments will be approved and then have unapproved SPAM replies attached to them. But anything unapproved will be automatically removed by the script.

EXAMPLE -- with a short data set

Notice, after bulk delete, only status 1 (approved) comments are left.

Notice that cid 11 has a pid of 10, which is missing from the cid list. In other words, cid 11 is an orphaned child comment. The original parent comment was unpublished and deleted because ALL unapproved (status=0) comments were deleted.

Notice that there are 4 comments for node 4 and 1 comment for node 2. These will show up in the node_comments_statistics table when it is rebuilt.

comment TABLE

```
cid   pid   nid   uid   subject    . . . .    status     thread
      name
1     0     2     3     Wait a minute!        1          01/
      matthew
2     0     4     1     Hey . . .             1          01/
      peter
3     2     4     1     Let's get deeper 1    0          1.00/
      peter
4     3     4     1     How deep will it go?  1          01.00.00/
      peter
11    10    4     0     Zorro child of Doug   1          01.00.00.01.00/
Zorro
```

field_data_comment_body TABLE

```
THE FOLLOWING FIELDS HAVE IDENTICAL VALUES
entity_type = comment
bundle = comment_node_blog
deleted = 0
language = und
delta = 0
comment_body_format = filtered_html

THESE FIELDS DIFFER
entity_id  revision_id      comment_body_value
1          1                Wait a minute! . . .
2          2                Hey . . . this is some spam for your spam.
3          3                This is a reply to your spam to your spam.
4          4                I wonder how many comments . . .
11         11               Zorro the child of Doug. . .
```

field_revision_comment_body TABLE

```
THE FOLLOWING FIELDS HAVE IDENTICAL VALUES
```

```
entity_type = comment
bundle = comment_node_blog
deleted = 0
language = und
delta = 0
comment_body_format = filtered_html

THESE FIELDS DIFFER
entity_id  revision_id     comment_body_value
1          1               Wait a minute! . . .
2          2               Hey . . . this is some spam for your spam.
3          3               This is a reply to your spam to your spam.
4          4               I wonder how many comments . . .
11         11              Zorro the child of Doug. . .
```

REBUILD node_comment_statistics table

It is important to rebuild the node_comment_statistics_table after bulk deletion. Fortunately, a function for doing that is available in the Devel Module. But because I didn't want to require that the Devel be installed on the site, I borrowed the function and slightly renamed it (underscore) and added an output comment.

```
<?php
#!/usr/bin/env drush
/**
* Rebuild node_comment_statistics table
* The function in this script is borrowed verbatim from the Devel module.
* Copying it here means that Devel does not need to be enabled on your site.
*
* Run the script with the following command line if the script is in your site root.
*
*        $drush scr rebuild_node_comment_statistics.sh
*/

// check if we can bootstrap
$self = drush_sitealias_get_record('@self');
if (empty($self)) {
  drush_die("I can't bootstrap from the current location.", 0);
}

_devel_rebuild_node_comment_statistics();

/**
 * Regenerates the data in node_comment_statistics table.
 * Technique - http://www.artfulsoftware.com/infotree/queries.php?&bw=1280#101
 *
 * @return void
 */
function _devel_rebuild_node_comment_statistics() {
  // Empty table.
  db_truncate('node_comment_statistics')->execute();  // <-- saves previous data in a table dump

  // TODO: DBTNG. Ignore keyword is Mysql only? Is only used in the rare case
  // when two comments on the same node share same timestamp.
  $sql = "
    INSERT IGNORE INTO {node_comment_statistics} (nid, cid, last_comment_timestamp,
last_comment_name, last_comment_uid, comment_count) (
      SELECT c.nid, c.cid, c.created, c.name, c.uid, c2.comment_count FROM {comment} c
      JOIN (
        SELECT c.nid, MAX(c.created) AS created, COUNT(*) AS comment_count FROM {comment} c WHERE
```

```
status = 1 GROUP BY c.nid
      ) AS c2 ON c.nid = c2.nid AND c.created = c2.created
    )";
  db_query($sql, array(':published' => COMMENT_PUBLISHED));

  // Insert records into the node_comment_statistics for nodes that are missing.
  $query = db_select('node', 'n');
  $query->leftJoin('node_comment_statistics', 'ncs', 'ncs.nid = n.nid');
  $query->addField('n', 'changed', 'last_comment_timestamp');
  $query->addField('n', 'uid', 'last_comment_uid');
  $query->addField('n', 'nid');
  $query->addExpression('0', 'comment_count');
  $query->addExpression('NULL', 'last_comment_name');
  $query->isNull('ncs.comment_count');

  db_insert('node_comment_statistics', array('return' => Database::RETURN_NULL))
    ->from($query)
    ->execute();
}
```

****

When you run the script, it updates node_comment_statistics table data:

node_comment_statistics TABLE

nid	cid	last_comment_timestamp	last_comment_name	last_comment_uid	comment_count
1	0	1455232987	NULL	1	0
2	1	1455497538		3	1
3	0	1455676852	NULL	1	0
4	11	1458604710	Zorro	0	4