It will happen.

• Database corruption will happen to you.
• Sooner or later.
• Fortunately, it's super easy to recover!
Step 1: Restore last-good backup
Step 2: Receive the praise of a grateful nation.
Lunchtime yet?
Oh.

- You don’t have a known-good backup?
- That’s a shame.
- Sadly, even good backups can…
  - Have hidden long-term corruption.
  - be too old.
  - *(whisper it)* or PostgreSQL bugs.
For example...

- PostgreSQL 9.1 streaming replication bugs.
- Secondary could have silent corruption.
- Fail over, promote secondary, and...
  - … horrible things happened.
Let’s talk about…

• Preventing corruption.
• Finding corruption.
• Fixing corruption…
  • … if you can.
Hi.

- Christophe Pettus
- Consultant with PostgreSQL Experts, Inc.
- Working with PostgreSQL since 1997.
Preventing Corruption
PostgreSQL is very trusting.

- PostgreSQL assumes the file system is perfect.
- It cannot recover from any silent bad data write (unless you are very lucky).
- With 9.3 checksums, you at least get a warning.
- So use them.
Hardware is cheap. Data is expensive.

- Use good-quality hardware.
- Be sure your hardware properly honors fsync, end to end.
- This is more common than you think.
- Avoid (if at all possible) network-attached devices for $PGDATA.
Backup, backup, backup.

- What only exists on one drive you do not truly possess.
- Be sure you follow the right backup protocol for your technique.
  - `pg_start_backup()`, etc.
- Test your backups.
  - An untested backup strategy isn’t one.
Prophylactic pg_dump

• pg_dump to /dev/null.
• Reads every single row in the database.
• Great for finding lurking corruption.
• Of course, if you can save the dump file, do so!
What causes corruption?
#1: Hardware failures.

- Underlying storage failure.
- Bad disk, bad controller.
- Garbage writes during power loss.
- Battery backup that didn’t.
- Bad RAM.
- Especially non-ECC RAM.
#2: Hardware “features.”

- Deferred or entirely missing fsync behavior.
- Often done to flatter benchmark results.
- Network-attached-storage that does not handle detach gracefully.
- Soft-RAID edge conditions.
#3: PostgreSQL bugs.

- 9.x had a series of unfortunate replication bugs.
- Used to be extremely rare.
- With luck, will become extremely rare again.
#4: Operator error.

- Backups that do not include critical files.
- Backups that do not follow protocol.
- Backups that forget external tablespaces.
- `rm -rf` in the wrong directory.
- Bungled attempts at problem recovery.
- Delete the wrong files to free space.
What to do?

• Buy good hardware, demand your cloud provider do so, or have multi-tier redundancy.
• Make backups, and test them.
• Stay up on PostgreSQL releases, and read the release notes.
PostgreSQL Disk Format
A quick overview.

- Full details are in the documentation…
- … or in the code.
- Enough to understand what the problems might be.
Everything’s under \$PGDATA.

- The heap and indexes (actual data).
- The write-ahead log (at least via symlink)
- The commit log.
base/

- Contains all “real” database data.
- Tables, indexes.
- Subdirectories, one per database.
- Named with OID for the database.
total 0

drwx------ 14 postgres postgres  476 Sep 17 12:59 .
drwx------ 22 postgres postgres  748 Oct 18 14:51 ..
drwx------ 238 postgres postgres  8092 Nov 5 2013 1
drwx------ 302 postgres postgres 10268 Oct 18 14:52 108290
drwx------ 238 postgres postgres  8092 Sep 17 2013 12292
drwx------ 296 postgres postgres 10064 Oct 18 14:52 1653531
drwx------ 315 postgres postgres 10710 Sep 8 16:09 1653924
drwx------ 491 postgres postgres 16694 Oct 18 14:52 1724452
drwx------ 630 postgres postgres 21420 Oct 18 14:52 1781788
drwx------ 697 postgres postgres 23698 Oct 18 14:52 1783503
drwx------ 242 postgres postgres  8228 Oct 18 14:51 1785736
drwx------ 248 postgres postgres  8432 Oct 18 14:52 271548
drwx------ 368 postgres postgres 12512 Jul 10 13:46 90822
drwx------  2 postgres postgres   68 Sep 14 15:35 psql_tmp
postgres=# select oid from pg_database where datname='silverandgold';
    oid
---------
  1653924
(1 row)
In each database directory...

- One big list of files.
- One or more per relation (table, index).
- The name is the relfilenode of the relation.
drwx------   14 postgres  postgres     476 Sep 17 12:59 ..
-rw-------    1 postgres  postgres  172032 Jun 29 10:04 12030
-rw-------    1 postgres  postgres  24576 Jun 29 10:04 12030_fsm
-rw-------    1 postgres  postgres   8192 Jun 29 10:04 12030_vm
-rw-------    1 postgres  postgres   16384 Jun 24 12:48 12032
-rw-------    1 postgres  postgres  24576 May 31 14:36 12032_fsm
-rw-------    1 postgres  postgres   8192 May 31 15:07 12032_vm
-rw-------    1 postgres  postgres  16384 Jun 24 12:48 12034
-rw-------    1 postgres  postgres  40960 Jun 29 10:04 12035
-rw-------    1 postgres  postgres  73728 Jun 29 10:04 12036
-rw-------    1 postgres  postgres  24576 May 31 15:07 12036_fsm
-rw-------    1 postgres  postgres   8192 May 31 14:36 12036_vm
-rw-------    1 postgres  postgres  32768 Jun 29 10:04 12038
-rw-------    1 postgres  postgres  40960 Jun 29 10:04 12039
-rw-------    1 postgres  postgres   0 May 31 14:36 12044
-rw-------    1 postgres  postgres   8192 May 31 14:36 12046
-rw-------    1 postgres  postgres   8192 May 31 14:36 12047
-rw-------    1 postgres  postgres  450560 Jun 29 10:04 12048
-rw-------    1 postgres  postgres  24576 Jun 24 12:48 12048_fsm
-rw-------    1 postgres  postgres   8192 May 31 14:36 12048_vm
-rw-------    1 postgres  postgres  155648 Jun 29 10:04 12050
silverandgold=# select relfilenode from pg_class where relname='comic_chapter';

relfilenode
-----------
  1654288
(1 row)
Relation files.

• If >1 GB, divided into 1 GB segments.
• .1, .2, .3…

• _fsm is the free space map.
• _vm is the visibility map.
• Small / new tables might be missing one or both of these.
Inside the table files.

- Table (and index) files are divided into 8KB pages.
- Can change this at compile time; no one does.
- Each page contains 0+ tuples.
- Tuples do not span pages; TOAST is used for larger attributes.
Page Format.
Items.

- Each item is variable length.
- Large attributes are (usually) compressed, and then spilled to TOAST if they won’t fit.
- Each item begins with a bitmap of NULL fields (if any are NULLable), then the columns in order.
A note about items.

- You cannot decode an item without access to its schema definition.
- Thus, corruption of the system catalogs can render a table unreadable.
Every row in a table has a ctid.

(block number, item offset) pair.

Can be used to select a precise row in absence of a primary key...

... or if corruption has rendered the primary key unusable.
silverandgold=# select ctid from comic_issue;
  ctid
--------
(0,2)
(0,4)
(0,6)
(0,10)
(4 rows)
silverandgold=# select id from comic_issue where ctid='(0,6)';
   id
----
   4
(1 row)
Indexes.

• For B-tree indexes, the same general structure as tuple pages.
• “Items” are heap pointers (for leaf nodes), index pointers (for internal nodes).
• Other index types have their own formats, but generally follow this one.
Write-Ahead Log

• Stored in pg_xlog/

• Can be symlinked elsewhere

• Contains a sequence of 16MB segments.

• Files are recycled and renamed when no longer required for crash protection or replication.
Yes, these files are important.

- pg_xlog bloating is a common failure condition.
- … especially with an archive_command that is failing for some reason.
- The “log” name implies something unfortunate.
Commit Log.

- Stored in pg_clog/
- Series of files containing status of transactions.
- Stored as bitmaps, two bits per transaction.
- Be sure your backups include it!
Basic Techniques.
Save all the parts!

• Stop PostgreSQL.
• Do a full file-system level backup.
• Keep that backup safe.
• Make changes methodically, and document each step.
Index Corruption.

- The most common kind of corruption.
- Drop the index in a transaction, and confirm that solves the problem.
- If so, rebuild the index.
- If not, it’s probably not index corruption.
Take a `pg_dump`.

- `pg_dump` reads every row, and...
- ... creates a logically-good snapshot.
- Restore that into a clean database.
Bad Data Page.

- Checksum failures, complaints about bad headers, etc.
- Can you do a pg_dump of the table?
- zero_damaged_pages = on.
Really Bad Data Pages.

- Can you SELECT around them?
- Do a COPY out of the good data, drop table, COPY back in.
- Or do a CREATE TABLE from the SELECT, rename appropriately.
- DELETE just the bad rows by ctid, if you can isolate them.
Finding bad data pages.

- Iterate through rows in PL/pgSQL…
- … with an exception block around the SELECT.
- Catch and log any rows that throw an exception.
- Very helpful for finding TOAST corruption.
Great and Desperate Cures.
Known unknown knowns.

- All corruption is, by its nature, a one-off situation.
- Be sure to determine the extent of it before continuing.
- Be sure you can step backwards!
There are no recipes.
REMEMBER.
WORK ON A COPY.
First things first.

- Are there errors in demsg indicating a hardware or OS problem?
- Is the OOM killer terminating backends?
- Disk I/O errors?
- Can you cp -R the data directory to /dev/null?
Very, very bad data pages.

- As in, the backend crashes when it touches them.
- Isolate pages, use `dd` to zero out those blocks.
- Be sure to drop and recreate all indexes on the table!
WAL files corrupt or missing.

- You went on vacation…
- The system ran out of disk space…
- And they called you to say that it won’t start now.
- “We just deleted some log files.”
- “Which ones?”
“pg_xlog”
“Is that bad?”
“Yes.”
pg_resetxlog

- Tells PostgreSQL that WAL files it needs for crash recovery... it doesn't need.
- Can get the server to start with missing log files.
- Read the instructions carefully!
- High risk of inconsistent data! Check the database very thoroughly!
pg_clog
PG::InternalError: ERROR: could not access status of transaction 2924295225
DETAIL: Could not read from file "pg_clog/0AE4" at offset 212992: Success.
pg_clog corruption.

- Good news: Rarely subtle.
- Missing file.
- Truncated file.
Patching.

- Replace a missing file with all-zeros file.
- 00 for a transaction means “in progress.”
- Previously-committed transactions can thus disappear.
- Be prepared to do more cleanup in this case.
System catalog corruption.

• The nightmare scenario.

• Very hard to recover from, unless the corruption is very small.

• Probably requires expert attention to do recovery or scavenging.
War Stories.
PG::InternalError: ERROR: could not access status of transaction 2924295225
DETAIL: Could not read from file "pg_clog/0AE4" at offset 212992: Success.
So, how did we get here?

- Network connectivity issue caused secondary to be promoted to primary.
- New secondary couldn’t handle load.
- Beefier primary was initialized from secondary, but…
- … on startup, these errors popped out.
What happened?

• Same network problems that began the situation caused the rsync building the primary to abort.

• No one noticed in the rush to get the primary back on line.
The fix...

• ... the missing clog file could be copied from the secondary and dropped into place.

• Problem solved!

• Very lucky that clog file was available.
The moral?

• No matter how bad a disaster is…
• … rushing can make it worse.
• Make sure that you are not introducing new problems as you are repairing old ones.
ERROR: missing chunk number 0 for toast value 968442 in pg_toast_263610
So, how did we get here?

- New primary provisioned by promoting from secondary.
- Errors started appearing almost immediately.
- But only one row, on one table...
Spooky!

• … and only on some queries.
• Isolating the record using primary key found nothing; the record retrieved just fine.
• Reindexing the TOAST table? No help.
• Iterating through the table did find it, however.
What happened?

- Two levels of corruption.
- Bad TOAST entry, and...
- ... two rows with the same primary key.
  - One referring to the “bad” row.
- Index scans only found the “good” row.
- Seq scans found both.
The fix...

- Delete the missing row by ctid.
- Iterate through all other tables to confirm no other corrupt rows.
The moral?

- Read the release notes.
- This was directly related to an existing bug in PostgreSQL.
- But the hosting provider* hadn’t upgraded PostgreSQL promptly.
- * cough AWS cough
Uh, Christophe?
About that upgrade...
So, how did we get here?

- New primary provisioned by promoting from secondary.
- New primary put into service, old primary decommissioned.
- Everything looks fine for a few hours, until…
Spooky!

- Some existing rows are missing.
- Some existing rows are duplicated, as if both old and new from an UPDATE had been committed.
- No error messages.
What happened?

- PostgreSQL bug.
- Since fixed.
- clog values were not properly being transmitted from primary to secondary under high-write-load conditions.
- So, some rolled back, etc.
The fix...

• Enough information in the database (date/time stamps) to delete the bad rows, and copy the missing ones from the old database.

• Hand-crafted scripts to do so.

• Never, ever want to do that again.
The moral?

- Do not exclude that it could be a PostgreSQL problem.
- Do thorough sanity checks on promoted primaries.
- Have a client or employer who understands open source software.
Thank you!
Christophe Pettus
PostgreSQL Experts, Inc.

pgexperts.com

thebuild.com

christophe.pettus@pgexperts.com

Twitter @xof