FIELD NOTES

Magento Field Guide

Magento 2 indexing, cron and MSI inventory fixes.

59 fixes, each one a written note with diagrams and a small Python and Node.js script that finds the problem and repairs it. All 59 have a script in the public repo.

Before you run anything. Every script starts in a dry run: it reports what it would change and writes nothing. Read that output first, then set `DRY_RUN=false` when you are satisfied it has understood your data.

Scripts: github.com/allanninal/magento-fixes
Guides: allanninal.dev/magento
All 14 repos: github.com/allanninal

The fixes

1 **Admin or integration tokens expire and silently break automation**
A script or cron job calls POST /rest/V1/integration/admin/token once, saves the bearer token in an env var, and runs fine for hours. Then every REST call starts coming back Unauthorized, and because a 401 is a plain HTTP status rather than a business exception, the automation often swallows it instead of alerting anyone. Here is why the admin token quietly expires, and a script that detects the failure, re-authenticates safely once, and reports the rest.
`detect_token_expiry.py`
https://www.allanninal.dev/magento/admin-token-expiry-breaks-automation/
https://github.com/allanninal/magento-fixes/tree/main/admin-token-expiry-breaks-automation

---

2 **Magento anchor category shows products from disabled subcategories**
You disabled a subcategory. It is gone from the menu, gone from the sitemap, nobody can click into it. But its products are still sitting in the storefront listing of the parent anchor category, as if nothing changed. Nobody reassigned anything, and a reindex does not clear it. Here is why Magento's anchor aggregation was never built to check whether a child category is active, and a small script that finds every leaked SKU so you can review it.
`anchor_leak_check.py`
https://www.allanninal.dev/magento/anchor-category-leaks-disabled-subcategory-products/
https://github.com/allanninal/magento-fixes/tree/main/anchor-category-leaks-disabled-subcategory-products

---

3 **Catalog price rule cron failure blocks downstream indexers**

You added a store view, or a new catalog price rule went live, and now nothing seems to reindex. Prices are wrong, search results feel stale, and it is not just that one store, it is every store on the install. Here is why a single failing catalogrule_apply_all run can hold a lock that stops indexer_reindex_all_invalid and indexer_update_all_views cold, and a small script that finds the SKUs and cron rows that prove it.

`detect_stuck_catalog_rule.py`

https://www.allanninal.dev/magento/catalog-price-rule-cron-blocks-indexers/

https://github.com/allanninal/magento-fixes/tree/main/catalog-price-rule-cron-blocks-indexers

---

4 **Catalog price rule discounts base price instead of group price**

You built a catalog price rule for one customer group, a wholesale tier or a VIP segment, expecting it to shave a percentage off the price that group already negotiated. Instead the storefront shows a discount off the plain base price, or worse, shoppers in a group the rule was never meant for get the markdown too. Here is why the catalog price rule indexer discounts the wrong starting amount and a small script that finds the SKUs where it happened.

`detect_rule_price_mismatch.py`

https://www.allanninal.dev/magento/catalog-price-rule-wrong-base-price/

https://github.com/allanninal/magento-fixes/tree/main/catalog-price-rule-wrong-base-price

---

5 **Category product assignment changes do not reach the search index**

A merchandiser adds a product to a category, or an API call updates catalog_category_product, and the assignment is correct everywhere the admin looks. But the product never appears in category or fulltext search results on the storefront, even with cron running normally. Here is why the catalogsearch_fulltext changelog silently misses these edits, and a script that finds every affected category and SKU pair through the REST API.

`find_missing_category_assignments.py`

https://www.allanninal.dev/magento/category-assignment-missing-from-search-index/

https://github.com/allanninal/magento-fixes/tree/main/category-assignment-missing-from-search-index

---

6 **Magento category product count wrong or zero on large catalogs**

A category page in the admin, or the product_count attribute over the REST API, says a category has far fewer products than it actually does, sometimes zero, even though every product is still correctly assigned to it. Nobody removed anything. The catalog is fine. What broke is the index table that Magento reads the count from, usually after a reindex that got interrupted partway through. Here is why it happens, why anchor categories are hit hardest, and a small script that finds every category where the reported count disagrees with the real assignments.

`category_count_check.py`

https://www.allanninal.dev/magento/category-product-count-wrong/

https://github.com/allanninal/magento-fixes/tree/main/category-product-count-wrong

---

7 **Configurable parent has no image while children do**

A shopper browsing the storefront sees a perfectly good product photo on a configurable product page. But call the same product through the API and its media_gallery_entries array comes back empty. Nobody removed an image. The configurable parent simply never had one of its own, because Magento never copies or inherits images from the simple children up to the parent row. The storefront quietly covers for it by falling back to a child's image. An API consumer asking for the parent directly gets nothing. Here is why that gap opens up and a small script that finds every configurable where it has.

`configurable_missing_image.py`

https://www.allanninal.dev/magento/configurable-parent-missing-image/

https://github.com/allanninal/magento-fixes/tree/main/configurable-parent-missing-image

---

8 **Configurable parent stock status not derived from children**

A shopper opens a configurable product, picks a size, and Magento says In Stock right up until they try to add it to the cart. Every single child is actually out of stock. Or the opposite happens: a configurable sits hidden as Out of Stock while two of its children are perfectly salable. Nobody touched the parent. An import updated a child's quantity, or a source-level edit landed through the API, and nothing told the parent to recompute its own cached stock flag from its children. Here is why that cache goes stale and a small script that finds every configurable where it has.

`configurable_stock_sync.py`

https://www.allanninal.dev/magento/configurable-parent-stock-status-not-synced/

https://github.com/allanninal/magento-fixes/tree/main/configurable-parent-stock-status-not-synced

---

9 **Coupon marked used despite the order never completing**

A shopper enters a limited use coupon, the cart looks fine, and then checkout fails, maybe the cart no longer meets a minimum order amount after a shipping method changed the total. No order gets created. But the coupon's usage count went up anyway, and now a one time code says it is spent, or a customer's per person limit is burned, for a purchase that does not exist. Here is why Magento does this and a small script that finds every coupon this happened to.

`reconcile_coupon_usage.py`

https://www.allanninal.dev/magento/coupon-marked-used-without-order/

https://github.com/allanninal/magento-fixes/tree/main/coupon-marked-used-without-order

---

10 **Coupon usage limit not enforced, allowing unlimited reuse**

You set Uses per Coupon to one, or Uses per Customer to one, and the storefront still accepts the same code on the tenth order. Nothing looks wrong in the rule configuration. But the counters Magento uses to enforce that limit are updated after the fact, on a queue, and when that queue lags or the consumer is not running, the limit simply stops being checked. Here is why that happens and a small script that finds every coupon being reused past its real limit without touching a single order.

`evaluate_coupon_usage.py`

https://www.allanninal.dev/magento/coupon-usage-limit-not-enforced/

https://github.com/allanninal/magento-fixes/tree/main/coupon-usage-limit-not-enforced

---

11 **Credit memo grand total not refreshed after adjustment edit**

A refund needs a little extra for return shipping, or a small deduction for a restocking fee, so someone opens the credit memo screen and types a number into Refund Shipping or Adjustment Fee. The grand total on screen does not move. They save anyway, and now there is a credit memo whose total does not match what its own fields add up to. Here is why only quantity changes ever trigger a recalculation, and a small script that finds every credit memo where the math does not check out.

`flag_creditmemo_total_drift.py`

https://www.allanninal.dev/magento/credit-memo-total-not-refreshed/

https://github.com/allanninal/magento-fixes/tree/main/credit-memo-total-not-refreshed

---

12 **Credit memo total wrong on multi invoice orders**

A customer ordered three items, one shipped a week before the other two, so the order carries two invoices. A partial refund goes out against the second invoice, and the credit memo total, or just its tax amount, does not add up to what was actually invoiced and refunded. Finance flags it, and now you are staring at a credit memo that is a real financial record, cannot be edited, and still looks wrong. Here is why Magento's total collectors get this wrong on split orders and a small script that finds exactly which credit memos do not add up.

`flag_creditmemo_discrepancy.py`

https://www.allanninal.dev/magento/credit-memo-total-wrong-multi-invoice/

https://github.com/allanninal/magento-fixes/tree/main/credit-memo-total-wrong-multi-invoice

---

13 **Cron generation halts entirely after a crash**

One cron job died weeks ago, an out of memory kill, a fatal timeout, a deploy that restarted the container mid run, and Magento never noticed it was gone. Every other job code still fires on schedule. But that one job code has not run since. Nobody killed it on purpose, and nothing in the admin says it stopped. Here is why Magento's own scheduler treats a dead process as still busy forever, and a small script that finds the exact job code stuck behind it.

`flag_stalled_cron.py`

https://www.allanninal.dev/magento/cron-generation-halts-after-crash/

https://github.com/allanninal/magento-fixes/tree/main/cron-generation-halts-after-crash

---

14 **cron_schedule fills with duplicate pending jobs**

You open the cron_schedule table expecting a handful of pending rows per job and instead find dozens, sometimes hundreds, all pending, all for the same job_code. Real cron work gets delayed or starves entirely behind the pile. Here is why Magento's cron generator keeps stacking duplicates and a small script that finds the surplus and prunes it safely, keeping the one job that still needs to run.

`prune_cron_schedule.py`

https://www.allanninal.dev/magento/cron-schedule-duplicate-pending-jobs/

https://github.com/allanninal/magento-fixes/tree/main/cron-schedule-duplicate-pending-jobs

---

15 **Cron jobs stuck in running state block all future runs**

The admin grid stops updating, orders sit unprocessed, and nothing in Magento's own logs says why. Somewhere in cron_schedule a single row has been sitting on running for days, and every later attempt to run that same job code refuses to queue a new one. Nobody killed the job on purpose. A cron process crashed weeks ago, maybe an out of memory kill, a deploy restart, or an infinite loop, and Magento has believed ever since that the job is still in progress. Here is why that row never clears itself and a small script that finds it before it starves an entire job group.

`flag_stuck_cron.py`

https://www.allanninal.dev/magento/cron-stuck-running-blocks-jobs/

https://github.com/allanninal/magento-fixes/tree/main/cron-stuck-running-blocks-jobs

---

16 **Product URL fails when rewrite generation is disabled with empty suffix**

A merchant turns off the trailing .html on every URL, sets category and product suffixes to nothing, and also has category/product rewrite generation switched off to save on reindex time. Individually each setting is fine. Together, on a store using categories in the product path, they turn ordinary product pages into 404s or 500s. Nothing in the catalog changed. Here is why the empty suffix breaks Magento's on-the-fly path resolution, and a small script that finds the stores and products actually affected.

`url_suffix_risk_check.py`

https://www.allanninal.dev/magento/disabled-rewrite-empty-suffix-error/

https://github.com/allanninal/magento-fixes/tree/main/disabled-rewrite-empty-suffix-error

---

17 **Disabling rewrite setting still creates duplicate rewrites**

You went into Catalog, Search Engine Optimization and set Generate category/product URL Rewrites to No, expecting products to keep one clean URL each. Instead a product assigned to a category still ends up with two rows in url_rewrite: a plain product URL and a category-prefixed one, as if the setting did nothing. Here is why the toggle only stops half the story, and a small script that finds the duplicate pairs over REST and reports them for repair.

`disabled_rewrite_setting_still_duplicates.py`

https://www.allanninal.dev/magento/disabled-rewrite-setting-still-duplicates/

https://github.com/allanninal/magento-fixes/tree/main/disabled-rewrite-setting-still-duplicates

---

18 **Duplicate credit memo created for one refund action**

Someone clicks Refund once, the page is slow, and they click it again just to be sure. Or a payment gateway retries a webhook it never got an acknowledgment for. Either way, two credit memos appear against the same order, seconds apart, for the same amount, and now the refunded total is double what it should be. Here is why Magento never stops the second one, and a small script that finds every near-duplicate pair so a human can review it.

`flag_duplicate_creditmemos.py`

https://www.allanninal.dev/magento/duplicate-credit-memo-created/

https://github.com/allanninal/magento-fixes/tree/main/duplicate-credit-memo-created

---

19 **Duplicate customer accounts share one email across websites**

The same shopper registers on two websites in the same Magento instance, using the exact same email address both times, and Magento lets it happen without a single error. Two separate customer records exist now, with two separate entity_ids, two order histories, and one email address that your ERP or CRM thinks belongs to a single person. Here is why Magento allows this, and a script that finds every email address split across more than one website so a human can decide which account is canonical.

`find_duplicate_email_clusters.py`

https://www.allanninal.dev/magento/duplicate-customer-accounts-same-email/

https://github.com/allanninal/magento-fixes/tree/main/duplicate-customer-accounts-same-email

---

20 **Duplicate or colliding order increment id**

Two customers call about the same order number, or your payment gateway return URL lands on the wrong receipt, or your ERP sync starts overwriting one order's data with another's. Magento's internal entity_id primary key never collided. But the human-facing increment_id, the one printed on invoices, used in gateway callbacks, and typed into your ERP, quietly points at two different rows. Here is why the sequence tables behind increment_id can hand out the same number twice, and a small script that finds every collision and flags it without touching the number itself.

`find_duplicate_increment_ids.py`

https://www.allanninal.dev/magento/duplicate-order-increment-id/

https://github.com/allanninal/magento-fixes/tree/main/duplicate-order-increment-id

---

21 **Duplicate SKU created through concurrent API or import saves**

Two REST POST /V1/products calls land within the same instant, or an import bunch and an async bulk save race each other while touching the same SKU, and Magento lets both of them believe the SKU does not exist yet. One wins the unique key check outright, the other either errors out or, on some code paths, quietly commits a second entity_id under the same SKU string. Search and the storefront start showing two rows for what should be one product. Here is why the unique index does not prevent this and a small script that finds every collision and reports it without touching your data.

`find_sku_collisions.py`

https://www.allanninal.dev/magento/duplicate-sku-race-condition/

https://github.com/allanninal/magento-fixes/tree/main/duplicate-sku-race-condition

---

22 **Duplicate url_key blocks product resave after migration**

You open a migrated product, change nothing important, hit save, and Magento throws back "URL key for specified store already exists." The product looks fine. The url_key field looks fine. But somewhere else in the catalog sits another product with the same url_key, or one that normalizes to the same request_path once the suffix and category path are applied, and Magento 2 will not let either of them save until that collision is resolved. Here is why Magento 1 never caught this, why Magento 2 refuses to, and a small script that finds every collision over REST and reports a safe rename.

`duplicate_url_key.py`

https://www.allanninal.dev/magento/duplicate-url-key-blocks-resave/

https://github.com/allanninal/magento-fixes/tree/main/duplicate-url-key-blocks-resave

---

23 **Duplicate or orphaned url_rewrite rows cause 404s**

A product or category URL that used to work now 404s, or the storefront quietly serves the wrong page for a path that two rows both claim. Nobody removed the product. Nothing looks wrong in the catalog. What broke is the url_rewrite table itself, usually after a bulk import, a Magento 1 to 2 migration, or a CSV re-import that did not carry full category data. Here is why Magento leaves these rows behind, and a small script that finds the duplicates and orphans over REST and repairs what it safely can.

`url_rewrite_check.py`

https://www.allanninal.dev/magento/duplicate-url-rewrite-rows-404/

https://github.com/allanninal/magento-fixes/tree/main/duplicate-url-rewrite-rows-404

---

24 **Enabled product missing from the Magento 2 storefront**

The admin grid says Enabled. You open the product, everything looks fine, and yet the storefront acts like it does not exist. No search result, no category listing, a 404 on the direct URL. Status is only one of several conditions Magento checks before it will show a product, and any one of the others failing is enough to hide it completely. Here is what actually decides storefront visibility and a small script that checks each condition in order so you know exactly which one is broken.

`diagnose_missing_product.py`

https://www.allanninal.dev/magento/enabled-product-missing-from-storefront/

https://github.com/allanninal/magento-fixes/tree/main/enabled-product-missing-from-storefront

---

25 **Grid refresh by schedule caps at batch size, backlog grows**

The admin order grid is missing rows that clearly exist in the orders table. Same story on the invoice, shipment, and credit memo grids. Nothing crashed, nothing errored, and the cron job that syncs them is running on schedule exactly as configured. The problem is quieter than that: the sync query itself never asks for more than 100 rows at a time, so when more than 100 orders fall out of sync between ticks, the extra rows are simply invisible to that run. Here is why the batch cap leaks a backlog instead of draining it, and a small script that watches the symptom from the REST API and reports it.

`flag_grid_sync_backlog.py`

https://www.allanninal.dev/magento/grid-refresh-batch-size-backlog/

https://github.com/allanninal/magento-fixes/tree/main/grid-refresh-batch-size-backlog

---

26 **Imported products missing website assignment**

A CSV import runs clean, or an integration posts a new product through the REST API, and every field looks right. Price, stock, description, all there. But the product never shows up on the storefront, not in category pages, not in search, not even by direct URL. Here is why Magento can save a fully valid product with zero rows in catalog_product_website, and a script that finds every SKU stuck in that state through the REST API.

`find_missing_website_assignment.py`

https://www.allanninal.dev/magento/imported-products-missing-website-assignment/

https://github.com/allanninal/magento-fixes/tree/main/imported-products-missing-website-assignment

---

27 **Product shows in stock while salable quantity is zero**

The product detail page says In Stock, the Add to Cart button is live, and a shopper can drop it in their cart. But every unit is already spoken for. Salable quantity has been at zero for a while, and the shopper will only find out at checkout when Magento rejects the order. Here is why the in stock flag and the salable quantity number tell two different stories in MSI, and a small script that finds every SKU where they disagree.

`flag_phantom_in_stock.py`

https://www.allanninal.dev/magento/in-stock-flag-disagrees-with-zero-salable-qty/

https://github.com/allanninal/magento-fixes/tree/main/in-stock-flag-disagrees-with-zero-salable-qty

---

28 **Magento indexer stuck at reindex required or processing**

The admin grid says Reindex required. The storefront shows a stale price, or a product that should be out of stock is still buyable. You run bin/magento indexer:reindex and nothing happens, or it just says the indexer is already running. Nobody killed it on purpose. A cron run crashed partway through weeks ago, and Magento has believed ever since that the job is still in progress. Here is why that lock never clears itself and a small script that finds it before the changelog backlog makes the eventual catch-up worse.

`flag_stuck_indexers.py`

https://www.allanninal.dev/magento/indexer-stuck-reindex-required/

https://github.com/allanninal/magento-fixes/tree/main/indexer-stuck-reindex-required

---

29 **Listing page and product page disagree on stock status**

A customer emails a screenshot: the category grid clearly says In Stock, but the product page they clicked through to says Out of Stock and will not let them add it to the cart. Nobody edited the product. Nothing is corrupted. A sale, or even just a pending unpaid order, quietly zeroed out the real time salable quantity the moment it was placed, while the grid keeps showing whatever the stock status index last calculated. Here is why those two numbers can disagree and a small script that finds every SKU where they do.

`diagnose_stock_mismatch.py`

https://www.allanninal.dev/magento/listing-vs-detail-stock-status-mismatch/

https://github.com/allanninal/magento-fixes/tree/main/listing-vs-detail-stock-status-mismatch

---

30 **Manual invoice missing tax leaves a false amount due**

An admin invoices an order in two passes, say the simple products first and a virtual product on its own, using Sales, Orders, Invoice in the Admin. Every item ends up invoiced, the merchant is sure the order is fully paid, but the order still shows an amount due. Nobody touched a discount or shipping. The gap is tax, and it is missing from the second invoice because of a real, reproduced core bug, not a store misconfiguration. Here is why it happens and a script that finds every order it hit.

`detect_invoice_tax_shortfall.py`

https://www.allanninal.dev/magento/manual-invoice-missing-tax/

https://github.com/allanninal/magento-fixes/tree/main/manual-invoice-missing-tax

---

31 **Magento 2 negative source item quantity still counted as positive stock**

One source has 2 units, another has 3, and a third has been driven to minus 29 by a drop-ship deficit or an oversell tracking rule. Add them the way a spreadsheet would and the stock is clearly wiped out. But Magento's own indexer can compute a combined salable quantity for that SKU that reads as positive and available, because a negative source is only forced to zero when that source is explicitly marked out of stock. Here is why the sign gets lost in the sum and a small script that catches the impossible total before a customer buys stock that does not exist.

`flag_negative_source_masked.py`

https://www.allanninal.dev/magento/negative-source-item-counted-as-positive/

https://github.com/allanninal/magento-fixes/tree/main/negative-source-item-counted-as-positive

---

32 **Automatic no coupon rule stops applying once a coupon rule exists**

You built a No Coupon cart price rule so every shopper gets a small automatic discount. It worked fine on its own. Then you launched a seasonal coupon rule, and the automatic discount quietly stopped showing up for anyone who enters a code. Nothing in either rule's conditions changed. Here is why Magento drops the automatic rule from consideration the moment a coupon is entered, and a small script that finds every rule pair where this is happening.

`find_shadowed_no_coupon_rules.py`

https://www.allanninal.dev/magento/no-coupon-rule-disabled-by-coupon-rule/

https://github.com/allanninal/magento-fixes/tree/main/no-coupon-rule-disabled-by-coupon-rule

---

33 **Online refund silently falls back to offline**

A support agent opens an order, clicks Credit Memo, and expects Magento to hand the money back through the same card gateway the customer paid with. Instead the form only shows an offline refund option, or the agent does not notice the toggle and submits an offline credit memo anyway. Magento marks the order Refunded. The customer never sees the money. Here is why the gateway call gets skipped without any error, and a small script that finds every credit memo where that gap is hiding.

`flag_offline_refund_fallback.py`

https://www.allanninal.dev/magento/online-refund-falls-back-offline/

https://github.com/allanninal/magento-fixes/tree/main/online-refund-falls-back-offline

---

34 **Order closed prematurely while invoice is still pending**

The order is fully shipped, the admin grid says Closed, and everyone moves on. Except nobody got paid. The invoice was created with Not Capture, or left pending on purpose for a later capture, and Magento's own automatic close logic never checked that. It only looked at whether there was anything left to ship or invoice, saw nothing, and closed the order with total_due still greater than zero. Here is why that check misses an unpaid invoice and a small script that finds every order this happened to.

`flag_premature_closure.py`

https://www.allanninal.dev/magento/order-closed-with-pending-invoice/

https://github.com/allanninal/magento-fixes/tree/main/order-closed-with-pending-invoice

---

35 **Order confirmation emails silently stop sending**

Checkout works. Payment captures. Inventory deducts. The order lands in the admin grid looking completely normal. But the customer never gets a confirmation email, the invoice email never goes out, and nobody notices until a customer asks where their receipt is, sometimes days later. The order was never broken. The email queue behind it was never drained, because the Magento cron scheduler that drains it quietly stopped running. Here is why that dependency is so easy to miss and a small script that surfaces the backlog before customers do.

`flag_cron_email_backlog.py`

https://www.allanninal.dev/magento/order-emails-not-sent-cron-dependency/

https://github.com/allanninal/magento-fixes/tree/main/order-emails-not-sent-cron-dependency

---

36 **Order sequence drifts from auto increment after migration**

A store just moved off Magento 1, or a DBA restored a backup, and a week later two customers are staring at the same order number, or the numbers jump from 100004521 straight to 100009000 with nothing in between. Nothing about sales_order looks wrong. The rows are all there, entity_id ticks up cleanly. The part that broke lives somewhere the migration script never touched: a separate sequence table that hands out the next human facing order number. Here is why that table falls behind and a small script that finds exactly which stores drifted.

`detect_sequence_drift.py`

https://www.allanninal.dev/magento/order-sequence-drift-after-migration/

https://github.com/allanninal/magento-fixes/tree/main/order-sequence-drift-after-migration

---

37 **Order status wrong after partial or zero total refund**

A customer got a store-credit-only refund, or you refunded just the shipping, or a bundle item was partially credited back. The credit memo saved fine. But the order still says Processing days later, even though nothing is left to pay back. Or the opposite happens, one partial refund and the order jumps straight to Closed while most of the balance is still owed. Here is why Magento's own refund logic misreads these totals and a small script that finds every order it happened to.

`flag_status_after_refund.py`

https://www.allanninal.dev/magento/order-status-wrong-after-refund/

https://github.com/allanninal/magento-fixes/tree/main/order-status-wrong-after-refund

---

38 **Order stuck in payment review with no way out**

A PayPal fraud filter, an Adyen or Braintree risk check, or a custom gateway adapter flagged the transaction for manual review, and Magento put the order in payment_review to wait for the gateway's own callback. Then the callback never came. There is no invoice, the Cancel button is missing from the admin order view, and the order just sits there holding an inventory reservation forever. Here is why Magento leaves it that way and a small script that finds the stuck ones and safely reports, flags, or cancels them.

`reconcile_payment_review.py`

https://www.allanninal.dev/magento/order-stuck-in-payment-review/

https://github.com/allanninal/magento-fixes/tree/main/order-stuck-in-payment-review

---

39 **Order stuck on pending payment after invoice is paid**

The invoice is right there, its state is Paid, and total_paid matches grand_total. But the order itself still reads new or pending_payment, so it does not enter your processing queue, does not trigger shipment, and shows up on every report as unpaid revenue that was, in fact, already collected. Here is why order state and invoice state can disagree, and a small script that finds every order stuck like this without touching a single record on its own.

`detect_pending_payment_mismatch.py`

https://www.allanninal.dev/magento/order-stuck-pending-payment-after-invoice/

https://github.com/allanninal/magento-fixes/tree/main/order-stuck-pending-payment-after-invoice

---

40 **Partial refund tax computed from the full order instead of the refunded items**

A customer returns two of the five items they ordered, and the credit memo should refund tax on those two items only. Instead the credit memo's tax total matches the order's entire tax, as if all five had been returned. Nobody notices until accounting reconciles refunds against sales tax filings and the numbers do not add up. This is a real, repeatedly reported defect in Magento's credit memo tax collector, not a one off store bug, and here is why it happens and a script that finds every credit memo it hit.

`flag_creditmemo_tax_mismatch.py`

https://www.allanninal.dev/magento/partial-refund-tax-miscalculated/

https://github.com/allanninal/magento-fixes/tree/main/partial-refund-tax-miscalculated

---

41 **Product silently reassigned to the wrong store on save**

A product that has always lived on three websites gets saved by a CLI import script, a cron job, or a REST call, and comes back visible on only one. Nothing in the admin UI complained, no error was logged, and the merchandiser who set up the multi-website assignment never touched it. Here is why Magento can quietly force a product onto only its default website during save, and a script that finds every product this happened to before a customer notices it missing.

`repair_website_drift.py`

https://www.allanninal.dev/magento/product-force-assigned-wrong-store/

https://github.com/allanninal/magento-fixes/tree/main/product-force-assigned-wrong-store

---

42 **Product images duplicated on repeated import or product duplication**

Re-run the same CSV import, or click Save and Duplicate on a product, and the exact same picture shows up again in the gallery under a new name, image_1.jpg, image_2.jpg, and so on. Do it a few more times and one product ends up with a dozen entries that are all the same photo. Here is why Magento never checks whether the image is already attached before writing another gallery row, and a script that finds the true byte-identical duplicates and removes only those, safely.

`find_duplicate_gallery_entries.py`

https://www.allanninal.dev/magento/product-images-duplicated-on-import/

https://github.com/allanninal/magento-fixes/tree/main/product-images-duplicated-on-import

---

43 **Products flap out of category or search during scheduled indexing**

A merchandiser swears a product was in the category five minutes ago. A shopper searches for a SKU that was findable this morning and gets zero results. Nobody touched the product. What actually happened is a scheduled reindex cycle briefly deleted the search or category index entries for that product before rebuilding them, and the storefront was queried in that gap. Here is why Magento's own scheduled indexing causes this, and a small script that detects and reports the flapping instead of guessing.

`flag_flapping_products.py`

https://www.allanninal.dev/magento/products-flap-during-scheduled-indexing/

https://github.com/allanninal/magento-fixes/tree/main/products-flap-during-scheduled-indexing

---

44 **Products vanish mid reindex on delete then insert cycle**

A support ticket says a product was on the site an hour ago and is gone now, but nobody touched it in the admin. The product is still enabled, still in stock, still sitting in the catalog. What changed is the price index. Magento's price indexer rebuilds its table by deleting a batch of rows and then inserting them again, and for a moment in between, that batch of products simply is not there. Here is why that gap exists and a small script that tells the harmless version of this apart from a real disappearance.

`reindex_anomaly.py`

https://www.allanninal.dev/magento/products-vanish-during-price-reindex/

https://github.com/allanninal/magento-fixes/tree/main/products-vanish-during-price-reindex

---

45 **Reserved order ids create unexplained numbering gaps**

Finance pulls the order list, notices number 000000512 is missing, and asks what happened to it. Nothing happened to it. Magento reserved that number the moment a shopper reached checkout, then the shopper closed the tab, or the card was declined, or the order-place transaction rolled back, and the number was never attached to a real order. It will never be reused. Here is why Magento leaves these gaps on purpose, and a small script that finds every one of them and reports it instead of guessing.

`reconcile_reserved_order_ids.py`

https://www.allanninal.dev/magento/reserved-order-id-numbering-gaps/

https://github.com/allanninal/magento-fixes/tree/main/reserved-order-id-numbering-gaps

---

46 **REST created orders skip reservation placement**

An ERP or marketplace connector posts historical or external orders straight to POST /V1/orders, and every one of them looks fine. Real order items, a decremented product quantity, a normal looking order in the grid. But MSI's salable quantity never moves, because the only thing that reduces it, a row in inventory_reservation, is only ever written by a plugin that hooks the checkout flow those orders never touch. Here is why that gap opens and a small script that finds exactly which order lines were never reserved.

`flag_unreserved_orders.py`

https://www.allanninal.dev/magento/rest-orders-skip-reservation-placement/

https://github.com/allanninal/magento-fixes/tree/main/rest-orders-skip-reservation-placement

---

47 **Salable quantity corrupted by bad reservation compensation**

The stock is on the shelf, the source item quantity looks right, and the product still will not sell, or worse, it keeps selling past zero. MSI does not store salable quantity anywhere. It calculates it every time from the source quantity minus every reservation row ever written for that SKU. When one order event fails to write its compensating reservation, that running total is wrong forever, and nothing about placing a new order fixes the old damage. Here is why the gap opens and a small script that finds the exact SKUs it hit.

`flag_salable_qty_corruption.py`

https://www.allanninal.dev/magento/salable-quantity-corrupted-by-reservations/

https://github.com/allanninal/magento-fixes/tree/main/salable-quantity-corrupted-by-reservations

---

48 **Magento 2 salable quantity goes negative or allows oversell**

A product page shows the item in stock, checkout lets a customer buy it, and yet the salable quantity Magento reports for that SKU is a negative number. Physical stock in the warehouse is fine. Nobody touched the qty column directly. What actually moved is the reservation ledger, a separate append-only log that MSI subtracts from physical stock to compute what is left to sell, and when a compensating entry on that ledger goes missing, the computed number can drift below zero forever. Here is why that invariant breaks and a small script that catches it before it turns into a real oversold order.

`flag_salable_qty_oversell.py`

https://www.allanninal.dev/magento/salable-quantity-negative-oversell/

https://github.com/allanninal/magento-fixes/tree/main/salable-quantity-negative-oversell

---

49 **Sales order grid permanently out of sync with sales_order**

Support asks about an order that the customer swears they placed. You look it up with the REST API and there it is, entity_id, correct status, correct total, sitting right there in sales_order. But the admin order grid does not show it, or shows it stuck on an old status with a stale total. The order was never lost. The grid, a separate table that a cron job is supposed to keep in sync, just stopped picking it up, and a documented race condition means it may never pick it up again on its own. Here is why that happens and a small script that finds exactly which orders drifted.

`flag_grid_sync_drift.py`

https://www.allanninal.dev/magento/sales-order-grid-out-of-sync/

https://github.com/allanninal/magento-fixes/tree/main/sales-order-grid-out-of-sync

---

50 **Shared catalog price cached and served to the wrong company**

Company A's buyer opens a category page and sees their negotiated shared catalog price. Ten minutes later, a guest, or worse, Company B's buyer, loads the same category and sees Company A's discounted price instead of their own. Nobody edited a price. Nothing looks broken in the admin. Here is why Magento's full page cache and block cache can carry a shared catalog price across companies, and a small script that detects exactly which SKU, category, and customer group combination is showing the wrong number.

`flag_shared_catalog_price_mismatch.py`

https://www.allanninal.dev/magento/shared-catalog-price-cached-wrong-company/

https://github.com/allanninal/magento-fixes/tree/main/shared-catalog-price-cached-wrong-company

---

51 **Shared stock not synced across websites**

Two storefronts were supposed to draw from one warehouse. On paper they still are, since nobody touched the product data or the source quantity. But one website keeps selling units the other one already sold, and the totals never line up no matter how many times someone reindexes. The stock was never actually shared. Here is why that mapping quietly drifts and a small script that proves it, website by website, before you touch anything.

`detect_stock_desync.py`

https://www.allanninal.dev/magento/shared-stock-not-synced-across-websites/

https://github.com/allanninal/magento-fixes/tree/main/shared-stock-not-synced-across-websites

---

52 **Shipment tracking number dropped when created via API**

You call the ship endpoint with a carrier and a tracking number attached, Magento answers with a 200 and a brand new shipment ID, and everything looks fine. Then you open the shipment and the tracking number is not there. No error was ever raised. The customer gets a shipped email with nothing to track, and support has to go dig through the original request to figure out what carrier was even used. Here is why the tracks array can vanish between your request and the database, and a small script that finds every shipment this happened to and reports or repairs it safely.

`detect_dropped_tracking.py`

https://www.allanninal.dev/magento/shipment-tracking-dropped-via-api/

https://github.com/allanninal/magento-fixes/tree/main/shipment-tracking-dropped-via-api

---

53 **Stale price index shows wrong or old prices**

You changed a price in the admin, or a catalog price rule kicked in, and the admin agrees with you. But the storefront still shows the old number. Nothing is broken in the sense of an error message, the site just keeps quoting yesterday's price. Here is why Magento's price index falls behind and a small script that finds the SKUs where admin and storefront disagree.

`flag_stale_price_index.py`

https://www.allanninal.dev/magento/stale-price-index-wrong-prices/

https://github.com/allanninal/magento-fixes/tree/main/stale-price-index-wrong-prices

---

54 **Tax rate wrong when shipping address differs from customer default**

A logged-in customer with more than one saved address checks out, picks a shipping address that is not their default, and the order still gets taxed as if it shipped to the default one. Sometimes that means the wrong region's rate. Sometimes, across countries, it means 0% tax on an order that should have carried VAT. The tax rules are not broken. The address Magento resolves before it asks the rules is. Here is why it happens and a script that finds every order it hit.

`detect_tax_address_mismatch.py`

https://www.allanninal.dev/magento/tax-rate-wrong-for-shipping-address/

https://github.com/allanninal/magento-fixes/tree/main/tax-rate-wrong-for-shipping-address

---

55 **Order tax off by rounding between calculation methods**

A script recomputes an order's expected tax with a plain "price times qty times rate" formula, and it does not match the order's actual tax_amount by a cent or two. Nothing is broken. Magento lets a merchant choose where in the math it rounds, per unit, per row, or once on the total, and each choice legitimately produces a slightly different number from the same catalog prices and the same tax rate. Here is why that happens and a script that replicates the configured algorithm before it calls anything drift.

`flag_tax_rounding_drift.py`

https://www.allanninal.dev/magento/tax-rounding-drift/

https://github.com/allanninal/magento-fixes/tree/main/tax-rounding-drift

---

56 **Tax recalculated incorrectly after coupon applied**

A shopper applies a coupon, the discount looks right on the order, and the tax line still looks plausible on its own. But add it up: subtotal minus discount plus tax does not equal the grand total. Nobody notices at checkout, because every individual number looks reasonable. It only shows up when finance reconciles orders against what they collected in tax, and by then dozens of orders already carry the wrong number. Here is why Magento's total collectors disagree with each other and a script that finds every order where they do.

`reconcile_coupon_tax.py`

https://www.allanninal.dev/magento/tax-wrong-after-coupon-applied/

https://github.com/allanninal/magento-fixes/tree/main/tax-wrong-after-coupon-applied

---

57 **Out of stock threshold change not applied to existing items**

Someone lowers the Out-of-Stock Threshold to sell down the last few units of everything, or raises it to stop selling on razor thin stock. The setting saves without an error. But a pile of products that should have flipped in stock or out of stock never move, the admin grid disagrees with what the storefront shows, and the MSI salable quantity API tells a third story entirely. The threshold changed. The existing inventory did not catch up. Here is why, and a small script that finds and repairs the exact source items the change left behind.

`repair_threshold_source_items.py`

https://www.allanninal.dev/magento/threshold-change-not-applied-existing-items/

https://github.com/allanninal/magento-fixes/tree/main/threshold-change-not-applied-existing-items

---

58 **URL rewrites not generated on product edit or duplicate**

You save a product through the REST API, or you duplicate one in the admin grid, and everything looks fine. No error, a clean 200 OK. Then someone clicks the product link and gets a 404. Nothing wrote a url_rewrite row for it. This is not a misconfiguration. It is a long-standing core bug in how Magento resolves the store scope for a saved product, and it fails silently every time. Here is why it happens and a small script that finds the affected SKUs before your customers do.

`url_rewrite_missing.py`

https://www.allanninal.dev/magento/url-rewrite-not-generated-on-edit/

https://github.com/allanninal/magento-fixes/tree/main/url-rewrite-not-generated-on-edit

---

59 **Wrong tax or price shown per customer group at checkout**

Two customers, same product, same tier price, and yet one of them sees a different tax amount and a different final total at checkout. Nothing throws an error. The numbers just quietly disagree between customer groups. Here is why Magento 2 lets a customer group drift onto the wrong tax class or rate, and a small script that computes the expected tax for every group and flags the SKU/group pairs where the storefront disagrees.

`flag_tax_price_mismatch.py`

`https://www.allanninal.dev/magento/wrong-tax-price-per-customer-group/`

`https://github.com/allanninal/magento-fixes/tree/main/wrong-tax-price-per-customer-group`
