FIELD NOTES

PrestaShop Field Guide

PrestaShop stock, order state and Webservice API fixes.

73 fixes, each one a written note with diagrams and a small Python and Node.js script that finds the problem and repairs it. All 73 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/prestashop-fixes
Guides: allanninal.dev/prestashop
All 14 repos: github.com/allanninal

The fixes

1 **Refund created via API does not update the order's refunded quantity**
A script calls the PrestaShop webservice to issue a refund, gets back a shiny new credit slip, and everyone assumes the order is now correctly marked as partially or fully refunded. Then someone opens the order and the line still shows zero units refunded. Here is why creating a credit slip through the API never touches the order line's own refund counters, and a script that finds every order where the two disagree so it can be repaired safely.
`fix_api_refund_quantity.py`
https://www.allanninal.dev/prestashop/api-refund-not-updating-order-line/
https://github.com/allanninal/prestashop-fixes/tree/main/api-refund-not-updating-order-line

---

2 **Product duplication leaves a broken partial product after an error**
Someone clicks Duplicate on a product with a few combinations, and PrestaShop throws a 500 error partway through. The admin screen looks like the whole thing failed. It did not. A new product now sits in your catalog, sometimes still active and visible on the storefront, missing some or all of its combinations, features, images, or stock rows, because nothing rolled it back when the error hit. Here is why PrestaShop leaves this half-built product behind and a small script that finds it so a human can decide what to do about it.
`find_broken_duplicates.py`
https://www.allanninal.dev/prestashop/broken-product-duplicate-after-error/
https://github.com/allanninal/prestashop-fixes/tree/main/broken-product-duplicate-after-error

---

3 **Editing a carrier assigns it a new id and orphans old references**

A staff member opens a carrier in the back office, tweaks the delivery price, and saves. Nothing about the change looks dangerous. But PrestaShop just quietly retired the carrier's old id_carrier and gave it a brand new one. Any module, integration, or old order that cached that original id is now pointing at a row marked deleted. Here is why a simple carrier edit does this on purpose, and a script that finds every order left behind.

`find_orphaned_carriers.py`

https://www.allanninal.dev/prestashop/carrier-edit-creates-new-id/

https://github.com/allanninal/prestashop-fixes/tree/main/carrier-edit-creates-new-id

---

4 **Catalog price rules silently skip products that already have a specific price**

You built a catalog price rule, it is active, the product is in the category you targeted, and the storefront still shows the old price. No error, no warning in the admin, nothing red anywhere. The rule looks like it should be working and just is not, on some products but not others. Here is why PrestaShop quietly lets a product's own specific price beat any catalog rule every time, how to find every product this is happening to, and a safe way to report it without touching a merchant's deliberate override.

`catalog_rule_skip_report.py`

https://www.allanninal.dev/prestashop/catalog-rule-skipped-by-specific-price/

https://github.com/allanninal/prestashop-fixes/tree/main/catalog-rule-skipped-by-specific-price

---

5 **Combination stock quantities do not sum to the product level total**

A product with combinations shows a total quantity that does not match what you get when you add up every size and color. Nobody typed the wrong number in. PrestaShop keeps the product-level row and the combination rows in sync with application code, not a live sum, and that sync can quietly fall apart. Here is why the two figures drift and a small script that finds every mismatch, plus the orphaned rows hiding behind it, without ever writing a number on its own.

`combination_quantity_sum_mismatch.py`

https://www.allanninal.dev/prestashop/combination-quantity-sum-mismatch/

https://github.com/allanninal/prestashop-fixes/tree/main/combination-quantity-sum-mismatch

---

6 **Cover image missing or duplicated, breaking storefront image links**

A product should always have exactly one cover image, the picture the storefront shows in listings and on the product page. Sometimes it has none, and the main image link falls back to nothing. Sometimes it has two, and an API upload that should have just added a picture throws a database error instead. Both are the same underlying gap: nothing keeps the cover flag exactly right when the write path is a CSV import, a product duplication, or an interrupted webservice call. Here is why it happens and a small script that finds and safely fixes it.

`fix_cover_image.py`

https://www.allanninal.dev/prestashop/cover-image-missing-or-duplicated/

https://github.com/allanninal/prestashop-fixes/tree/main/cover-image-missing-or-duplicated

---

7 **Credit slip amount ignores the original voucher discount**

A customer used a voucher, the order total dropped, and everyone paid the net amount. Then a refund comes in, PrestaShop writes a credit slip, and the number on it is bigger than what the customer actually paid, because the discount never made it into the math. Here is why a PrestaShop credit slip can quietly hand back the voucher as extra refund, and a script that finds every order slip this already happened to so accounting can fix it by hand.

`check_credit_slip_voucher.py`

https://www.allanninal.dev/prestashop/credit-slip-ignores-voucher-discount/

https://github.com/allanninal/prestashop-fixes/tree/main/credit-slip-ignores-voucher-discount

---

8 **Product's default category is not among its assigned categories**

You open a product in the back office and the breadcrumb, or the storefront canonical link, points at a category the product is not even checked into anymore. Nothing errored when it happened. A merchant unchecked a category box weeks ago, or a category got deleted, or an import ran with a partial category list, and PrestaShop never went back to check whether id_category_default still made sense. Here is why that drift happens and a small script that finds every product it has happened to.

`flag_default_category_drift.py`

https://www.allanninal.dev/prestashop/default-category-not-in-assigned-categories/

https://github.com/allanninal/prestashop-fixes/tree/main/default-category-not-in-assigned-categories

---

9 **Product default category silently changed or lost after a catalog import**

You run a CSV import to update prices or stock, nothing errors, and the job finishes green. Days later a merchant notices a product's breadcrumb points at Home instead of the deep category it always lived in, or the storefront canonical URL looks wrong. Nobody edited that product by hand. The import did it, quietly, because PrestaShop's importer builds every row from scratch and does not know what the default category used to be. Here is why that happens and a small reconciler that catches it before a customer does.

`reconcile_import_default_category.py`

https://www.allanninal.dev/prestashop/default-category-overwritten-by-import/

https://github.com/allanninal/prestashop-fixes/tree/main/default-category-overwritten-by-import

---

10 **Duplicate customer accounts created from the same email during guest checkout**

A shopper checks out as a guest, comes back a week later, checks out as a guest again with the same email, and support ends up staring at two customer records that both claim to be the same person. Or a returning guest finally creates a password, and PrestaShop quietly adds a brand new row instead of turning the old one into their account. Nothing crashes. Nothing warns you. Here is why PrestaShop lets this happen and a script that finds every email with more than one live customer row so a human can decide what to do about it.

`check_duplicate_customers.py`

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

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

---

11 **Duplicate friendly URL slug collides across different products or categories**

A product page 404s, or worse, opens the wrong product, and nobody touched anything in the back office. Somewhere a bulk import, a Duplicate click, or a webservice call saved a link_rewrite that another product or category already owns. PrestaShop's own admin form would have stopped it. Whatever wrote this one did not go through that form. Here is why that gap exists and a script that finds every collision and safely renames the one losing the URL.

`fix_duplicate_friendly_url_slug.py`

https://www.allanninal.dev/prestashop/duplicate-friendly-url-slug/

https://github.com/allanninal/prestashop-fixes/tree/main/duplicate-friendly-url-slug

---

12 **Duplicate invoice numbers issued across different orders**

Two customers check out within a second of each other, and both end up with an invoice carrying the exact same sequential number. Your accountant notices it first, usually when trying to reconcile a VAT report, and now you have two legal documents claiming to be the same invoice. Here is why PrestaShop can hand out one number twice and a script that finds every pair this happened to so a human can decide how to fix it.

`check_duplicate_invoice_numbers.py`

https://www.allanninal.dev/prestashop/duplicate-invoice-numbers/

https://github.com/allanninal/prestashop-fixes/tree/main/duplicate-invoice-numbers

---

13 **Duplicate order history rows created for a single status change**

You open an order's history and the same status is listed twice in a row, back to back, with two different history ids and two nearly identical timestamps. Sometimes the customer even got the same email twice. Nothing in the store changed the order's real status, it just got recorded more than once. Here is why PrestaShop lets one status change turn into two history rows, how to find every order it has happened to, and a safe way to clean it up without losing the audit trail.

`duplicate_history_cleanup.py`

https://www.allanninal.dev/prestashop/duplicate-order-history-rows/

https://github.com/allanninal/prestashop-fixes/tree/main/duplicate-order-history-rows

---

14 **Order payment row duplicated for certain order state configurations**

A customer pays by bankwire, staff move the order to Payment accepted, and the order detail page suddenly shows two payment lines for the exact same amount, seconds apart. The order is not double charged, but the record now looks like it is, and total_paid_real can be thrown off in reports. Here is why an order state can quietly write its own payment twice and a script that finds every order this happened to so a human can review it.

`check_duplicate_payments.py`

https://www.allanninal.dev/prestashop/duplicate-order-payment-row/

https://github.com/allanninal/prestashop-fixes/tree/main/duplicate-order-payment-row

---

15 **Duplicate product reference or SKU allowed across different products**

Someone creates a new product, or clicks Duplicate product on an existing one, types or copies a reference, and saves. PrestaShop takes it without complaint, even though that exact reference already sits on a completely different product somewhere else in the catalog. Now your accounting export, your marketplace feed, and your barcode scanner cannot tell the two products apart. Here is why PrestaShop lets this happen and a small script that finds every collision so a human can decide what to do about it.

`find_reference_collisions.py`

https://www.allanninal.dev/prestashop/duplicate-product-reference-sku/

https://github.com/allanninal/prestashop-fixes/tree/main/duplicate-product-reference-sku

---

16 **Duplicating a product keeps the original's friendly URL unchanged**

You click Duplicate in the product list to save yourself some typing, and PrestaShop hands you a fresh product with a new id, set to disabled, ready to edit. Everything looks right until you check its SEO tab: the friendly URL is word for word the same slug as the product you copied it from. Nothing crashed. No warning showed up. Here is why the clone keeps the exact same slug and a script that finds every one of these collisions and fixes the ones it is safe to fix.

`fix_duplicated_product_slug.py`

https://www.allanninal.dev/prestashop/duplicated-product-keeps-original-slug/

https://github.com/allanninal/prestashop-fixes/tree/main/duplicated-product-keeps-original-slug

---

17 **Currency exchange rate update in one shop overwrites another shop's rate**

You open Localization, Currencies in one shop context and update the exchange rate, meaning it for that shop only. Check a second shop that uses the same currency and its rate has changed too, to the exact number you just typed. Nothing errored. Nothing warned you. This is not a mistake in your update. PrestaShop stores a currency's exchange rate once per currency row, not once per shop, so every shop sharing that currency id inherits whatever value was written last. Here is why that happens and a small script that detects when it likely just did.

`detect_rate_overwrite.py`

https://www.allanninal.dev/prestashop/exchange-rate-overwritten-across-shops/

https://github.com/allanninal/prestashop-fixes/tree/main/exchange-rate-overwritten-across-shops

---

18 **Expired voucher still usable and left attached to orders**

A customer adds a voucher a day before it expires, takes a slow trip through checkout, and pays after the code should have died. PrestaShop lets the discount through anyway, and it rides all the way into a paid order. Nothing about the order looks broken at a glance, but the discount shown no longer matches what a fresh entry of that same code would allow, and finance ends up looking at totals that do not add up. Here is why an expired voucher can stay attached through checkout and a script that finds every cart and order still carrying one.

`check_expired_voucher.py`

https://www.allanninal.dev/prestashop/expired-voucher-still-applied/

https://github.com/allanninal/prestashop-fixes/tree/main/expired-voucher-still-applied

---

19 **Free gift line quantity doubles when unrelated cart items are removed**

A customer qualifies for a free gift, the rule adds one free unit, everything looks right. Then they remove some completely different item from the cart, and the gift line quietly becomes quantity 2, still marked free, still costing nothing, but now handing out twice the product the rule was ever meant to give away. Nobody touched the gift. Nobody edited the rule. Here is why removing an unrelated item is exactly what triggers it, how to find every cart carrying the extra unit, and a script that reports the damage without silently rewriting a customer's cart.

`find_doubled_gift_lines.py`

https://www.allanninal.dev/prestashop/free-gift-quantity-doubles/

https://github.com/allanninal/prestashop-fixes/tree/main/free-gift-quantity-doubles

---

20 **Free shipping voucher fails to zero out the shipping cost**

The customer enters a free shipping code, the cart summary even shows the discount line, checkout goes through, and the confirmation email still lists a shipping charge. Nothing looks broken from the storefront, and the voucher shows as applied. But the order's shipping total never actually dropped to zero. Here is why PrestaShop's free_shipping flag can be honored on screen and ignored on the order, and a script that finds every order where this happened.

`check_free_shipping.py`

https://www.allanninal.dev/prestashop/free-shipping-voucher-not-applied/

https://github.com/allanninal/prestashop-fixes/tree/main/free-shipping-voucher-not-applied

---

21 **Guest order becomes untrackable after registering an account with the same email**

Someone checks out as a guest, gets their confirmation email, then a week later decides to create a real account using that exact same email address. They expect their old order to be sitting right there in their order history. It is not. Nothing crashes, nothing warns anyone, the order is simply gone from the account's point of view, even though the email matches perfectly. Here is why PrestaShop leaves that order behind and a script that finds every one of these orphaned orders so a human can decide what to do with them.

`find_orphaned_orders.py`

https://www.allanninal.dev/prestashop/guest-order-untrackable-after-registration/

https://github.com/allanninal/prestashop-fixes/tree/main/guest-order-untrackable-after-registration

---

22 **Friendly URL field accepts invalid values instead of a proper slug**

Someone posted a product through the webservice, and the row saved fine. Weeks later the product page 404s, or opening it in the back office throws a validation error the moment you hit save. Whatever wrote the link_rewrite field never went through PrestaShop's own slug check, and the API let it through anyway. Here is why that gap exists and a script that finds every slug that is not actually a slug and safely repairs it.

`fix_invalid_friendly_url_format.py`

https://www.allanninal.dev/prestashop/invalid-friendly-url-format-accepted/

https://github.com/allanninal/prestashop-fixes/tree/main/invalid-friendly-url-format-accepted

---

23 **Invoice number stays empty after an order is marked shipped via API**

You POST a new row to order_histories to move the order to Shipped, the call succeeds, and the order's state updates in the back office right away. But the invoice number field stays blank, the customer portal has nothing to download, and accounting cannot reconcile the order to an invoice line. Nothing errored. PrestaShop simply never created the order_invoice row in the first place, because the specific state change your integration posted was never the one that triggers it. Here is why that gap opens up on the webservice path and a script that finds every order stuck this way and backfills the invoice for the ones it is actually safe to fix.

`backfill_missing_invoice.py`

https://www.allanninal.dev/prestashop/invoice-number-missing-after-api-status-change/

https://github.com/allanninal/prestashop-fixes/tree/main/invoice-number-missing-after-api-status-change

---

24 **Product has no default combination, causing price to display as zero**

A product with variants looks fine in the list of products, but the price column reads 0.00, and the storefront shows the same. Nothing is wrong with the combinations themselves, they have real prices and real stock. The product row just lost track of which one is the default. Here is why that pointer goes missing and a small script that finds the products stuck like this and repairs the pointer safely.

`fix_default_combination.py`

https://www.allanninal.dev/prestashop/missing-default-combination-zero-price/

https://github.com/allanninal/prestashop-fixes/tree/main/missing-default-combination-zero-price

---

25 **Duplicate key error creating a default combination per shop in multistore**

A merchant with two or more shops tries to set a default combination on the second shop, and PrestaShop hands back a SQL error: a duplicate entry for key product_default. The write can leave the wrong shop holding the default, or leave a shop with no default at all. Here is why the per shop scoping breaks down and a diagnostic script that lists every product and shop where the default combination state is actually broken, before you touch anything.

`diagnose_multistore_default_combination.py`

https://www.allanninal.dev/prestashop/multistore-default-combination-duplicate-key/

https://github.com/allanninal/prestashop-fixes/tree/main/multistore-default-combination-duplicate-key

---

26 **Catalog listing shows the wrong price when currency differs per shop**

A merchant with two or more shops opens the backoffice Catalog list, or a storefront category page, and the price looks off. Click into the single product and the price shown there is different, and it is the correct one for that shop. The two views of the exact same id_product disagree, and nobody edited anything in between. Here is why listings and single product pages can resolve price differently in multistore, and a diagnostic script that finds every product and shop where the two views actually disagree, before you touch anything.

`diagnose_multistore_listing_price.py`

https://www.allanninal.dev/prestashop/multistore-listing-price-mismatch/

https://github.com/allanninal/prestashop-fixes/tree/main/multistore-listing-price-mismatch

---

27 **Tax calculation uses the wrong shop's default country rate**

A customer in one country places an order on a shop whose own default country carries a different tax rate, and the invoice comes out taxed at the shop's rate instead of the customer's. The order still looks fine at a glance, the totals add up internally, but the VAT or sales tax owed is simply wrong. Here is why the tax engine can silently fall back to the shop's own country and a diagnostic script that finds every order where that happened, without touching a single total on its own.

`audit_multistore_tax_rate.py`

https://www.allanninal.dev/prestashop/multistore-wrong-country-tax-rate/

https://github.com/allanninal/prestashop-fixes/tree/main/multistore-wrong-country-tax-rate

---

28 **Negative quantities recorded on backorder paid orders**

A product's stock_available row shows a quantity like -3 or -12, and it has sat that way for weeks. Nobody sold stock they did not have on purpose, but the number will not come back on its own. Here is why PrestaShop can drive stock below zero on backorder paid orders and race conditions, and a small script that tells real oversell demand apart from drift and repairs only the drift.

`negative_quantity_backorder.py`

https://www.allanninal.dev/prestashop/negative-quantity-on-backorder-orders/

https://github.com/allanninal/prestashop-fixes/tree/main/negative-quantity-on-backorder-orders

---

29 **Negative stock quantities appear even with backorders disabled**

The product page says deny backorders. The out_of_stock setting is correct. And yet the stock_available row for that product sits at minus two, minus five, sometimes worse. Nobody hand edited it, or so everyone insists. Here is why PrestaShop can let stock go negative even when it is configured to refuse every one of those sales, and a small script that finds every row where that promise was broken.

`reconcile_negative_stock.py`

https://www.allanninal.dev/prestashop/negative-stock-despite-backorder-denied/

https://github.com/allanninal/prestashop-fixes/tree/main/negative-stock-despite-backorder-denied

---

30 **Stock quantity goes negative after ordering an out of stock product**

A product shows zero stock, a customer orders it anyway, and the order goes through. Nothing in PrestaShop stops it, and once the order is validated the stock row quietly drops to -1, -2, or further. Here is why PrestaShop lets this happen, how to find every row it already happened to, and a script that reports the damage and clamps it back to zero once you say so.

`reconcile_negative_stock.py`

https://www.allanninal.dev/prestashop/negative-stock-quantity-after-order/

https://github.com/allanninal/prestashop-fixes/tree/main/negative-stock-quantity-after-order

---

31 **Order's carrier reference becomes invalid after editing order lines**

A staff member edits the quantity on an order line to fix a picking mistake. PrestaShop recalculates the shipping, and the order screen throws back "The order carrier ID is invalid." The order was fine an hour ago. Nothing about the customer's shipment actually changed. Here is why a carrier that was deleted or edited months ago can leave an order's carrier reference dead, and a script that finds every order this has already happened to.

`check_order_carrier.py`

https://www.allanninal.dev/prestashop/order-carrier-invalid-after-line-edit/

https://github.com/allanninal/prestashop-fixes/tree/main/order-carrier-invalid-after-line-edit

---

32 **Order current_state field goes stale after order history is edited**

An admin removes a wrongly added status line from an order, or a cleanup script deletes some old order_history rows, and the order list keeps showing a status that no longer matches what the history actually says. PrestaShop only ever updates orders.current_state as a side effect of adding a new history row, so editing or deleting a history row directly leaves that column pointing at a status the order no longer supports. Here is why that gap opens up and a script that finds every order where the pointer disagrees with its own history and repairs it safely.

`fix_stale_current_state.py`

https://www.allanninal.dev/prestashop/order-current-state-stale-after-history-edit/

https://github.com/allanninal/prestashop-fixes/tree/main/order-current-state-stale-after-history-edit

---

33 **Order detail stock quantity fields disagree with the real order quantity**

Every order line in PrestaShop carries two quantity fields that sound like they should always match: product_quantity, the amount actually ordered, and product_quantity_in_stock, a snapshot of whether that item was in stock when the order was placed. They come from separate code paths, and in real stores they quietly disagree, showing a line with one unit ordered and zero units in stock on the very same row. Here is why that happens and a script that finds every mismatched row and reports it for a human to review.

`check_order_detail_stock.py`

https://www.allanninal.dev/prestashop/order-detail-stock-quantity-inconsistent/

https://github.com/allanninal/prestashop-fixes/tree/main/order-detail-stock-quantity-inconsistent

---

34 **Order history record is missing for the order's current state**

Open an order in PrestaShop and its current_state column says one thing, while the order_history timeline underneath tells a different story, or no story at all. PrestaShop keeps these two records in sync by convention, not by a database constraint, so a crash, a module that writes the state directly, or a broken upgrade can leave an order pointing at a state that never got its own history row. Here is why that gap opens up and a script that finds every order where the two disagree and reports it for a human to review.

`check_order_history.py`

https://www.allanninal.dev/prestashop/order-history-missing-for-current-state/

https://github.com/allanninal/prestashop-fixes/tree/main/order-history-missing-for-current-state

---

35 **Order history entries appear out of chronological order**

You pull an order's history to see how it got to its current state, and the rows do not read top to bottom the way the story actually happened. Two states landed in the same second. The row PrestaShop calls the latest is not the one the order's own current_state field points to. Here is why the history table can drift out of order, how to detect it without touching anything, and a safe way to correct it once a human confirms what really happened.

`chronology_audit.py`

https://www.allanninal.dev/prestashop/order-history-out-of-chronological-order/

https://github.com/allanninal/prestashop-fixes/tree/main/order-history-out-of-chronological-order

---

36 **Order stored with no carrier and free shipping the customer was not entitled to**

An order lands in the back office with no shipping method on file and a shipping total of zero, but nobody ever earned that free shipping. The customer's cart never qualified for a voucher, and yet the order acts as if it did. Here is why PrestaShop's check-out can save an order like this, and a script that finds every order this has already happened to.

`check_missing_carrier.py`

https://www.allanninal.dev/prestashop/order-missing-carrier-unearned-free-shipping/

https://github.com/allanninal/prestashop-fixes/tree/main/order-missing-carrier-unearned-free-shipping

---

37 **Order created in the wrong state when the amount paid does not match the total**

A customer pays part of an order, or a payment module reports success on the wrong amount, and the order still lands on a normal, paid-looking state in PrestaShop. Nobody sees a warning. The order simply looks fine, current_state says "Payment accepted" or similar, and total_paid_real quietly disagrees with total_paid underneath it. Here is why PrestaShop lets that gap through during validation and a script that finds every order where the two amounts do not match, so a human can decide what to do about it.

`check_amount_mismatch.py`

https://www.allanninal.dev/prestashop/order-state-mismatch-on-amount-paid/

https://github.com/allanninal/prestashop-fixes/tree/main/order-state-mismatch-on-amount-paid

---

38 **Order gets stuck permanently on one status with no history advancing**

An order has been sitting on "Awaiting payment" or "Processing" for two weeks, and nobody moved it. The order looks untouched, but so does its history. Here is why PrestaShop can leave orders.current_state and the order_history table disagreeing with each other, and a small script that tells apart a truly stuck order from a hidden desync, then repairs it the one safe way.

`order_stuck_on_stale_status.py`

https://www.allanninal.dev/prestashop/order-stuck-on-stale-status/

https://github.com/allanninal/prestashop-fixes/tree/main/order-stuck-on-stale-status

---

39 **Order total does not match the sum of its order line totals**

Someone reconciles an order against its invoice and the numbers do not add up. The order header says one total, and when you add up every line on the order, plus shipping, minus discounts, you get a different number. Nothing on the order screen calls this out. Here is why PrestaShop lets its cached order total drift away from what the lines actually add up to, and a script that finds every order where the two disagree so a human can decide what to do about it.

`check_order_total.py`

https://www.allanninal.dev/prestashop/order-total-mismatch-with-line-sum/

https://github.com/allanninal/prestashop-fixes/tree/main/order-total-mismatch-with-line-sum

---

40 **Order total wrong after cancelling a line item on an order with a voucher**

A customer had a percent-off voucher applied at checkout, staff cancel one line from the order later on, and now the numbers on the order do not add up. The remaining product lines look right, shipping looks right, but total_paid and total_paid_tax_incl are off by an amount that traces straight back to the voucher. The discount was fixed against a cart total that no longer exists once the line is gone, and PrestaShop never re-derives it. Here is why that happens and a script that finds every order where it did.

`check_order_total_after_cancel.py`

https://www.allanninal.dev/prestashop/order-total-wrong-after-line-cancel-with-voucher/

https://github.com/allanninal/prestashop-fixes/tree/main/order-total-wrong-after-line-cancel-with-voucher

---

41 **Products or categories orphaned outside the root category tree**

A category still sits in the database, active, with products attached, but it never shows up in navigation. Nothing was deleted on purpose. This usually means its id_parent chain no longer leads back to the shop's root category, either because the Home category was removed directly instead of through the shop's reassignment flow, or an import set id_parent to an id that does not exist or belongs to a different shop. PrestaShop only renders what it can walk from the root, so the row is alive but unreachable. Here is why that happens and a small script that walks the tree and reports every orphaned category and product id.

`find_orphaned_categories.py`

https://www.allanninal.dev/prestashop/orphaned-categories-outside-root-tree/

https://github.com/allanninal/prestashop-fixes/tree/main/orphaned-categories-outside-root-tree

---

42 **System generated vouchers without a code cannot be deleted and pile up**

A free-shipping-over-X promo. A loyalty discount that attaches itself the moment a cart qualifies. No code, no typing, nothing for the customer to enter. These codeless cart rules work fine right up until their uses run out or the promo window closes, and then they just sit there. No delete button ever shows up for them, so the cart_rule table quietly fills with dead rows that clutter every admin listing and report. Here is why PrestaShop never wired up a way to remove them, how to find the ones that are actually dead, and a script that reports them for review without ever guessing which ones are safe to remove on its own.

`report_orphaned_vouchers.py`

https://www.allanninal.dev/prestashop/orphaned-codeless-vouchers-accumulate/

https://github.com/allanninal/prestashop-fixes/tree/main/orphaned-codeless-vouchers-accumulate

---

43 **Orphaned combinations remain linked after a product is removed from a shop**

You uncheck a shop in a product's Shops association panel, or remove it through Product V2, and the product itself looks clean everywhere in that shop. But its combinations do not always follow. A stale row can keep a combination linked to the shop you just removed, still able to resolve, price, or stock-check against a shop context the parent product no longer belongs to. Here is why PrestaShop's core does not cascade that cleanup, and a small script that finds and reports every orphaned tuple.

`orphaned_combination_shops.py`

https://www.allanninal.dev/prestashop/orphaned-combinations-after-shop-removal/

https://github.com/allanninal/prestashop-fixes/tree/main/orphaned-combinations-after-shop-removal

---

44 **Orphaned stock rows remain after a combination is deleted**

You delete a product combination in the Back Office, or through the combinations webservice resource, and everything looks clean. But the stock_available row that belonged to it can stay behind, still holding a quantity, still counted in the product's reported total stock, even though no combination points to it anymore. Here is why PrestaShop does not clean that row up on its own, and a small script that finds every orphan and removes it safely.

`orphaned_stock_rows.py`

https://www.allanninal.dev/prestashop/orphaned-stock-rows-after-combination-delete/

https://github.com/allanninal/prestashop-fixes/tree/main/orphaned-stock-rows-after-combination-delete

---

45 **Order reaches a paid state despite the product being out of stock**

A customer checks out, the payment module confirms the charge, and the order lands on Payment accepted. Only the product sold out a few seconds earlier, from a different order, and nobody re-checked. The order is financially paid and logically broken at the same time. Here is why PrestaShop lets this slip through, how to find every order it already happened to, and a script that flags the damage without touching a single euro that was already captured.

`audit_paid_out_of_stock.py`

https://www.allanninal.dev/prestashop/paid-order-despite-out-of-stock-product/

https://github.com/allanninal/prestashop-fixes/tree/main/paid-order-despite-out-of-stock-product

---

46 **Product left without a valid default category after category deletion**

Someone tidies up the catalog and deletes a category that is no longer needed. Everything looks fine in the back office category tree. Then a product that used to live under that category starts throwing errors on the storefront, or its default category shows up blank in the admin. The product still has other, perfectly valid categories. But its id_category_default field is still pointing at the category id you just deleted, and that id does not exist anymore. Here is why PrestaShop leaves that dangling reference behind and a small script that finds and repairs every product it happened to.

`fix_default_category.py`

https://www.allanninal.dev/prestashop/product-missing-default-category-after-deletion/

https://github.com/allanninal/prestashop-fixes/tree/main/product-missing-default-category-after-deletion

---

47 **Product visibility setting silently reverts after being changed via API**

You hide a product in one shop, or set it to catalog only, and it sticks for a while. Then a scheduled sync runs and the product is back to both, visible everywhere, like your change never happened. Nothing in your logs complains. This is not a fluke. A recurring sync is almost certainly overwriting the field, and in multistore setups a long standing webservice bug can make it worse. Here is why visibility keeps reverting and a small reconciler that finds and safely repairs the drift.

`reconcile_visibility.py`

https://www.allanninal.dev/prestashop/product-visibility-reverts/

https://github.com/allanninal/prestashop-fixes/tree/main/product-visibility-reverts

---

48 **Products disappear from the storefront days after being added by API**

You create a product through the webservice API, look it up right after, and it looks perfect. A few days later a customer says they cannot find it. You search for it, browse the category, check "related products," and it is nowhere. But it is still in the database, nothing was deleted, and no error was ever logged. Here is why API-created products quietly fall out of navigation and a small script that catches it before your customers do.

`detect_and_repair_delisted_products.py`

https://www.allanninal.dev/prestashop/products-disappear-days-after-api-creation/

https://github.com/allanninal/prestashop-fixes/tree/main/products-disappear-days-after-api-creation

---

49 **Partial refund accepted for more than the original line amount**

Someone processes a partial refund in the back office, or a script calls the webservice directly, and PrestaShop accepts it without a fight, even though the amount is bigger than what the line was ever worth. Nothing on the order screen stops it. Here is why PrestaShop lets a refund overshoot its own order line, and a script that finds every order where that already happened so a human can look before the books close.

`check_refund_overage.py`

https://www.allanninal.dev/prestashop/refund-amount-exceeds-line-total/

https://github.com/allanninal/prestashop-fixes/tree/main/refund-amount-exceeds-line-total

---

50 **Refunded quantity exceeds the originally ordered quantity**

Someone opens an order to check its refund history, and one line shows five units refunded on an order that only ever had three. Nothing on the order screen stops this from happening, and nothing warns you when it has. Here is why PrestaShop lets a line's refunded quantity climb past what was actually ordered, and a script that finds every line where the two disagree so a human can reconcile it against the real credit slips.

`check_refund_overage.py`

https://www.allanninal.dev/prestashop/refunded-quantity-exceeds-ordered/

https://github.com/allanninal/prestashop-fixes/tree/main/refunded-quantity-exceeds-ordered

---

51 **Reserved stock quantity drifts from actual pending orders**

A product still shows units held in reserved_quantity long after the order that reserved them was cancelled or refunded. Nobody edited stock by hand, but the count never lets go. Here is why PrestaShop's reserved quantity is a counter, not a live answer, and a small script that finds every drifted row and repairs it the way PrestaShop's own core would.

`reserved_quantity_drift.py`

https://www.allanninal.dev/prestashop/reserved-quantity-drift/

https://github.com/allanninal/prestashop-fixes/tree/main/reserved-quantity-drift

---

52 **Wrong price selected when multiple pricing rules overlap for a customer group**

A merchant sets an 89 price scoped to a specific customer group, and a broader 90 price sitting on All Groups. The customer belongs to the narrower group and should legitimately see 89. PrestaShop shows 90 instead. Nothing throws an error, nothing looks wrong in the back office, the store simply picked the wrong row out of several that all technically matched. Here is why the priority order is not the same thing as the lowest price, and a script that recomputes the price the customer actually qualifies for and flags every case where the store got it wrong.

`check_specific_price_priority.py`

https://www.allanninal.dev/prestashop/specific-price-priority-wrong/

https://github.com/allanninal/prestashop-fixes/tree/main/specific-price-priority-wrong

---

53 **Split orders show mismatched totals and wrong shipping cost**

A customer buys three products in one cart. Two ship by courier, one ships by a freight carrier, so PrestaShop quietly splits the cart into two orders behind the same reference. Days later someone reconciling the order finds one split order with a shipping cost of zero and no carrier, and the other carrying a shipping charge that does not match any carrier it actually used. Here is why PrestaShop's split logic mixes up which order gets which carrier, and a script that finds every split order where the carrier and the shipping cost do not agree.

`check_split_shipping.py`

https://www.allanninal.dev/prestashop/split-order-mismatched-shipping/

https://github.com/allanninal/prestashop-fixes/tree/main/split-order-mismatched-shipping

---

54 **Duplicate key error when updating stock quantity concurrently**

Two stock updates land at almost the same moment, maybe a webservice PUT racing a checkout, and PrestaShop throws a duplicate entry error for key product_sqlstock. The row you expected to update never gets touched, and somewhere there is now an orphan stock row fighting the real one. Here is why the check-then-act pattern in StockAvailable lets this happen and a script that finds the duplicate rows so you can review and merge them safely.

`find_duplicate_stock.py`

https://www.allanninal.dev/prestashop/stock-available-duplicate-key-error/

https://github.com/allanninal/prestashop-fixes/tree/main/stock-available-duplicate-key-error

---

55 **PATCH request to the stock endpoint is silently redirected and dropped**

You PATCH a new quantity to /api/stock_availables/{id}, the server answers with a plain 200, and your integration marks the write a success. Then you look at the actual stock and it never moved. Nothing threw an error. Here is why PrestaShop quietly turns that PATCH into a GET before it reaches the stock handler, and a small script that catches the dropped write and repairs it the safe way.

`stock_patch_guard.py`

https://www.allanninal.dev/prestashop/stock-patch-silently-dropped/

https://github.com/allanninal/prestashop-fixes/tree/main/stock-patch-silently-dropped

---

56 **Product quantity wrongly doubled or changed when order status changes**

An order moves from Awaiting payment to Payment accepted, and the stock drops by twice the line quantity. Or a Cancelled order gets reverted, and instead of restoring the stock it had before, PrestaShop adds even more. Nothing about your catalog changed, but the numbers in stock_available no longer match what actually shipped. Here is why PrestaShop's own status engine does this and a small script that finds the affected products and repairs them safely.

`reconcile_stock.py`

https://www.allanninal.dev/prestashop/stock-quantity-corrupted-on-status-change/

https://github.com/allanninal/prestashop-fixes/tree/main/stock-quantity-corrupted-on-status-change

---

57 **Physical quantity, reserved quantity, and virtual quantity fall out of sync**

A PrestaShop product carries three related stock numbers: physical_quantity on the shelf, reserved_quantity held for unshipped or unpaid orders, and quantity, the sellable number shown to customers. They are supposed to always add up. In real stores they quietly stop adding up, and nothing in the core ever notices or repairs it. Here is why that happens and a script that recomputes the true reserved figure from open orders and flags every row where the numbers disagree.

`check_stock_invariant.py`

https://www.allanninal.dev/prestashop/stock-quantity-formula-drift/

https://github.com/allanninal/prestashop-fixes/tree/main/stock-quantity-formula-drift

---

58 **Updating stock via the API can delete the linked product combination**

You send a normal stock update through the webservice, and a combination that was selling fine a minute ago now shows zero quantity everywhere, as if someone deleted it. Nothing was deleted. On multistore installs that share stock across a shop group, a single stock write can knock the row out of the shared scope that every shop in the group is reading from. Here is why that happens and a small script that finds every combination this has quietly happened to.

`stock_update_deletes_combination.py`

https://www.allanninal.dev/prestashop/stock-update-deletes-combination/

https://github.com/allanninal/prestashop-fixes/tree/main/stock-update-deletes-combination

---

59 **Total paid real field doubles after a partial payment update**

A partial payment comes in, someone or something confirms it a second time, and the order's total_paid_real quietly ends up at exactly twice the amount that was actually collected. The order_payment table may carry a duplicate row, the stored total no longer matches the sum of the real payments behind it, and nobody notices until finance tries to reconcile against the bank. Here is why PrestaShop's own payment recording code can fire twice for one payment, and a script that finds every order where the stored total disagrees with the real payment rows.

`reconcile_total_paid_real.py`

https://www.allanninal.dev/prestashop/total-paid-real-doubled/

https://github.com/allanninal/prestashop-fixes/tree/main/total-paid-real-doubled

---

60 **Per customer voucher limit ignored for guest checkouts sharing an email**

The cart rule says "total available for each user: 1." One code, one customer, ever. Then the same email address shows up on the back office three, four, six times, each order redeeming the exact same voucher. Nobody shared the code and nobody found a workaround. The gap is identity, not configuration. Here is why PrestaShop never recognizes that a repeat guest is the same person, how to find every voucher this happened to, and a script that reports the damage without touching a single already-placed order.

`audit_guest_voucher_reuse.py`

https://www.allanninal.dev/prestashop/voucher-per-user-limit-ignored-for-guests/

https://github.com/allanninal/prestashop-fixes/tree/main/voucher-per-user-limit-ignored-for-guests

---

61 **Single use voucher redeemed more than its allowed quantity**

The cart rule says quantity 1. One code, one use, ever. Then two, three, sometimes a dozen paid orders show up in the back office all discounted by the same "single use" voucher. Nobody edited the rule and nobody found a second code. The gap is timing, not configuration. Here is why PrestaShop lets a single-use voucher slip past its own limit under concurrent checkouts, how to find every order it happened to, and a script that reports the damage without touching a single already-paid order.

`audit_voucher_overuse.py`

https://www.allanninal.dev/prestashop/voucher-redeemed-beyond-quantity-limit/

https://github.com/allanninal/prestashop-fixes/tree/main/voucher-redeemed-beyond-quantity-limit

---

62 **Category created via webservice ignores shop scoping in multistore**

Your integration posts a new category to the PrestaShop webservice, meaning it for a single shop in a multistore setup. Check the back office and the category is sitting in every shop, not just the one you meant. Nothing errored. Nothing warned you. This is not a mistake in your JSON body. The webservice categories resource has no visible field for scoping shops, so unless you explicitly add id_shop to the query string, PrestaShop associates the category with the entire shop context by default. Here is why that happens and a small script that flags every category this has likely happened to.

`flag_category_shop_scope.py`

https://www.allanninal.dev/prestashop/webservice-category-ignores-shop-scope/

https://github.com/allanninal/prestashop-fixes/tree/main/webservice-category-ignores-shop-scope

---

63 **Setting a default combination via API throws a duplicate key error**

You PUT a combination with default_on set to 1, expecting it to become the product's default variant, and PrestaShop hands back a SQL error instead: a duplicate entry for key product_default. Nothing looks wrong in the payload. The problem is that another combination on the same product is still flagged default, and the Webservice API will not clear that flag for you the way the back office does. Here is why the unique key rejects the write and a script that swaps the default combination in the one order that never collides.

`swap_default_combination.py`

https://www.allanninal.dev/prestashop/webservice-default-combination-duplicate-key/

https://github.com/allanninal/prestashop-fixes/tree/main/webservice-default-combination-duplicate-key

---

64 **Product images updated via API do not create per shop associations in multistore**

You PUT a product image through the webservice, the response comes back 200, and the binary is sitting on disk right where it should be. Then you check the second shop in your multistore setup and the old image, or no image at all, is still showing. Nothing errored. Nothing in the response hinted anything was wrong. Here is why the update path can store the file but never wire it to the shop you asked for, and a small script that finds and reports every image missing its shop association.

`find_missing_image_shops.py`

https://www.allanninal.dev/prestashop/webservice-images-missing-shop-association/

https://github.com/allanninal/prestashop-fixes/tree/main/webservice-images-missing-shop-association

---

65 **Order created via webservice with no current state set at all**

An order comes in through the webservice, and it looks fine at a glance: a real id, a real reference, real order lines. But open it and current_state reads 0, or something that never actually happened, and the order_history timeline underneath is completely empty. Not stale, not mismatched, just missing from the start. Here is why POSTing to /api/orders can create an order that never really entered the state machine, and a script that finds every one of these and backfills it the safe way.

`backfill_stateless_orders.py`

https://www.allanninal.dev/prestashop/webservice-order-created-without-state/

https://github.com/allanninal/prestashop-fixes/tree/main/webservice-order-created-without-state

---

66 **Order created via webservice drops the carrier and shipping cost fields**

You POST a new order to the webservice with a real id_carrier and a real total_shipping, the call returns 201, and everything looks fine. Then you fetch the order back and the carrier is 0, or it is the shop's default carrier instead of the one the customer chose, and the shipping cost is recalculated to something else, or it is 0.00 outright. The invoice under- or overstates shipping, and sometimes the order even lands in a payment error state because the totals no longer add up. Here is why PrestaShop's order creation quietly ignores what you submitted and a script that finds and repairs every order this happened to.

`repair_carrier_shipping.py`

https://www.allanninal.dev/prestashop/webservice-order-drops-carrier-shipping/

https://github.com/allanninal/prestashop-fixes/tree/main/webservice-order-drops-carrier-shipping

---

67 **Orders created via webservice land in a payment error state**

You POST an order to the PrestaShop webservice, the call returns fine, and the order shows up, sitting on "Payment error" instead of the state you expected. Nobody declined a card. Nothing actually failed. PrestaShop simply compared the amount your integration sent against what the cart really adds up to, found the two did not agree, and forced the order into the error state on the spot. Here is why that comparison trips so easily on webservice-created orders and a reconciler that finds every affected order and repairs the safe cases.

`reconcile_payment_error.py`

https://www.allanninal.dev/prestashop/webservice-order-payment-error-mismatch/

https://github.com/allanninal/prestashop-fixes/tree/main/webservice-order-payment-error-mismatch

---

68 **Product created via webservice is invisible on the storefront**

You POST a new product through the webservice, the response comes back with an id, and the back office grid shows it active. Then you check the storefront and it is nowhere: not in its category, not in search, not anywhere in the catalog. Nothing errored. Nothing in the response hinted anything was wrong. Here is why the webservice can create a product that is active but structurally invisible, and a small script that finds and repairs the missing links.

`repair_invisible_product.py`

https://www.allanninal.dev/prestashop/webservice-product-invisible-on-storefront/

https://github.com/allanninal/prestashop-fixes/tree/main/webservice-product-invisible-on-storefront

---

69 **Product quantity field via webservice always reads or writes as zero**

You call the products resource, read the quantity field, and it is always 0, even for a product that is very much in stock. You try writing a new value back to it and nothing complains, but nothing changes either. This is not a bug in your script. PrestaShop moved real stock out of the product table years ago, and the webservice never caught up. Here is why products.quantity always lies and a small script that reads and repairs the real number instead.

`sync_stock_quantity.py`

https://www.allanninal.dev/prestashop/webservice-product-quantity-always-zero/

https://github.com/allanninal/prestashop-fixes/tree/main/webservice-product-quantity-always-zero

---

70 **Stock update via webservice does not sync back to the product quantity field**

You PUT a new quantity to stock_availables, the back office shows the right number, and then you GET the product back and its own quantity field is stale, or stuck at zero. Nothing is actually broken with your stock. Here is why PrestaShop keeps two numbers for the same thing, and a small script that finds the mismatch and repairs it the safe way.

`webservice_stock_resync.py`

https://www.allanninal.dev/prestashop/webservice-stock-update-not-synced-to-product/

https://github.com/allanninal/prestashop-fixes/tree/main/webservice-stock-update-not-synced-to-product

---

71 **Stock hooks and back in stock alerts never fire on webservice quantity updates**

You restock a product through the webservice, the quantity in stock_availables updates correctly, and everything looks fine in the database. Then a customer who asked to be notified when it came back never gets an email, and any module or custom code that reacts to a restock never runs. Nothing errored, nothing logged a failure. Here is why writing to the webservice never runs PrestaShop's stock hooks, and a small script that catches the restock yourself and drives the notification safely.

`detect_restock_alerts.py`

https://www.allanninal.dev/prestashop/webservice-stock-update-skips-hooks-and-alerts/

https://github.com/allanninal/prestashop-fixes/tree/main/webservice-stock-update-skips-hooks-and-alerts

---

72 **Combination resolved from the wrong shop context in multistore**

A product in your multistore install has a combination that looks broken in one shop only. Price shows 0, or the minimal quantity is not what you set, but the same product looks fine in a sibling shop. Nothing in your data changed. This is a known gap in how PrestaShop's assembler resolves the id_product_attribute for a product: it can pick a combination row that has a product_attribute_shop association for a different shop, not the one being served. Here is why that happens and a small script that finds every product where this has likely occurred.

`find_shop_mismatch.py`

https://www.allanninal.dev/prestashop/wrong-shop-combination-resolved/

https://github.com/allanninal/prestashop-fixes/tree/main/wrong-shop-combination-resolved

---

73 **Wrong tax rule applied on the product page for the delivery country**

A shopper in one country loads a product page and the tax included in the price does not match their own country's rate at all. It looks like a random number, or worse, it looks like a rate from a country nobody in the order is connected to. Here is why PrestaShop's tax lookup can quietly drift away from the customer's real delivery address, and a script that cross-checks the address, the tax rules group, and the displayed price so a mismatch gets reported instead of shipped to a buyer.

`find_tax_rule_mismatch.py`

https://www.allanninal.dev/prestashop/wrong-tax-rule-for-delivery-country/

https://github.com/allanninal/prestashop-fixes/tree/main/wrong-tax-rule-for-delivery-country

---