Rust

Rust server settings, plugins, permissions, RCON and performance troubleshooting.

- Adjust Rust Gather Manager rates and verify the actual yield
- Change Rust Stack Size Controller without losing inventory
- Configure BetterChat titles and group priority in Rust
- Configure Rust Backpacks permissions and preserve stored items
- Connect RustAdmin to Rust RCON without exposing credentials
- Create and test a Rust Kits starter kit safely
- Diagnose Rust game, query and RCON connection ports
- Diagnose Rust server lag and entity load without wiping
- Find Rust server.cfg and the correct identity save folder
- Grant Rust owner or moderator access using Steam IDs
- Manage Oxide plugin permissions without granting everyone admin
- Repair a Rust plugin JSON configuration without losing data
- Change Rust server description and header without changing the world
- Choose Rust server browser tags that match your rules
- Restore Rust+ pairing after a server move without resetting players
- Take a Rust save checkpoint before a planned panel restart
- Preflight a Rust custom map download before the next wipe
- Diagnose Rust custom map checksum mismatches without deleting the world
- Move Rust plugin data to a test server without losing state
- Test a Rust staging branch on a separate server identity

Adjust Rust Gather Manager rates and verify the actual yield

Gather Manager can change resource yields from different activities. A tree hit and a collectible on the ground do not use the same category. Start with one measured example so you can tell whether the intended rule changed.

Record the starting economy

Confirm Gather Manager is loaded and note its installed version. Copy its current configuration and relevant data before editing. Back up the active world and any economy databases before a wider rollout. Keep a dated record of existing multipliers so restoring a previous setting does not rely on memory.

Choose a normal test player, one tool and one activity. Empty enough inventory space and record the starting item count. Repeat a small number of comparable actions and note the result. Different tools, bonuses and other plugins can change the outcome, so keep those conditions constant.

Change a specific source

The documented command form is `gather.rate TYPE "RESOURCE" MULTIPLIER`. Valid types include dispenser, pickup, quarry and survey. For example, this targets wood received from dispensers such as trees:

```
gather.rate dispenser Wood 2
```

Use the trusted server console. Multiword resource names need quotes. Changing dispenser wood does not automatically establish the same rule for loose pickups or quarry output. Keep the first test narrow rather than applying a wildcard to everything.

Distinguish yield from capacity

Gather Manager also has dispenser scaling controls. Those alter the amount available in a resource source, which is a different decision from changing the amount received for an action. Leave scaling unchanged while testing yield. Otherwise two simultaneous adjustments make it difficult to explain why a tree or ore node now behaves differently.

After the test command, repeat the original measurement with the same account and equipment. Confirm the inventory gain rather than relying solely on a floating message. Note whether the rate behaves consistently across several comparable actions and whether nearby activity causes errors in the console.

Check interactions before publishing

1. Test a second source category without changing its setting, to make sure the change is limited.
2. Check plugins that modify tools, skills, VIP bonuses or gathering. Their effects may combine or override one another.
3. Review storage and crafting consequences. A small multiplier can produce much larger stockpiles over a full wipe.
4. Tell players exactly which activities changed. A label such as “2x” is ambiguous when only selected resources are affected.

Rollback without removing player property

Restore the recorded multiplier or retained working configuration using the plugin's supported application method, then repeat the baseline test. Existing gathered resources remain a separate economy consideration; changing future yield does not safely reverse everything players already earned. Do not delete inventories or wipe the world to undo a rate experiment.

If results remain inconsistent, send support the version, command, source type, resource, tool and before-and-after counts. Include the relevant redacted errors. This gives a reproducible example without exposing player databases or requiring broad access changes.

Related guides

- Configure storage stack limits
- Validate plugin configuration

Further reading

- uMod: Gather Manager
- uMod: Oxide plugin configuration

Looking for another game? Browse game server setup and troubleshooting guides.

Change Rust Stack Size Controller without losing inventory

Stack Size Controller changes how many items fit in a stack. It does not add storage slots or guarantee compatibility with every container plugin. Trial one ordinary resource before applying large limits across a live server.

Protect existing inventories

Record the plugin version and retain its current configuration. Make a consistent backup of the active world, player state and related inventory-plugin data before changing stack behaviour. Save a second copy outside the live server. Choose a quiet test window, tell administrators what will change and avoid using valuable player items for experiments.

Count the test items and record where they are stored. Use a test container with spare slots and a normal player account. Keep special items, modified weapons and containers with custom rules out of the initial trial.

Edit the current configuration format

The author identifies `IndividualItemStackSize` in the configuration as the place for individual item sizes. Do not follow old advice to edit a vanilla-defaults data file. Check the installed version's generated configuration and current author instructions before changing a field.

Individual and category multipliers can affect the final result. The documented calculation uses the configured individual stack size as the base, not necessarily Rust's vanilla size, and individual multipliers take precedence over category multipliers. Record every applicable field for your chosen test item so an unexpected size can be traced.

Apply one bounded change

1. Copy the configuration to a local JSON-aware editor.
2. Change one non-critical resource to a modest limit suitable for your rules.
3. Validate the JSON locally and upload it to its original path.
4. Use the author's documented `o.reload StackSizeController` command from the server console.
5. Read the reload result before moving any items.

Do not jump to extreme integer limits. A configuration accepted by the plugin can still produce awkward inventory behaviour or conflict with another plugin. A larger stack does not increase the number of slots available to the container.

Test movement and persistence

Merge and split the test resource, move it between inventory and the test container, and compare the total quantity before and after. Reconnect the player and check again. In a planned maintenance window, verify persistence across a normal save and restart. Watch for errors from backpack, furnace or other storage integrations.

If only one container differs, investigate that container's plugin rules before changing the global stack setting. Keep a record of which actions work and which fail; “stacking is broken” does not identify the affected path.

Plan the downward change

Reducing a limit may leave existing stacks larger than the new value. Before rollback, provide sufficient empty slots and test how the installed plugins handle those stacks. Do not assume they will split safely by themselves. Restore the original configuration if necessary and ask support for a controlled conversion plan rather than deleting items.

Share the item short name, relevant configuration fields and a redacted console error with support. Keep full inventory data and backup archives private.

Related guides

- Measure Gather Manager resource rates

- Configure Backpacks safely

Further reading

- uMod: Stack Size Controller
- uMod: Oxide plugin configuration

Looking for another game? Browse game server setup and troubleshooting guides.

Configure BetterChat titles and group priority in Rust

BetterChat formats player messages with group colours and titles. A coloured staff label is not proof that the player has administrator authority. Keep appearance changes separate from Rust owner access and sensitive Oxide permissions.

Prepare a reversible test

Confirm BetterChat is loaded and note its version. Back up its configuration and `oxide/data/BetterChat.json`, together with the relevant Oxide permission data. The group definitions are persistent data, so do not treat that file as a disposable generated configuration. Keep the original copy outside the server before editing.

Choose a test account and record its current memberships. If the account belongs to several existing roles, note all of them before testing. Select an unobtrusive title that cannot be confused with official CloudExa support or an automated security message.

Use the supported management commands

The author documents a `chat` command family. Run the following examples in the trusted server console; in-game chat commands use the slash-prefixed form. These examples show a new group and an individual membership; replace the placeholders with a deliberate group name and verified Steam64 ID:

```
chat group add communityhelpers
chat user add STEAM64 communityhelpers
chat group list
```

Administrative use requires `betterchat.admin`. Grant that only to the person managing the plugin, not to everyone receiving a title. Use the author's supported settings and syntax for the installed

version when editing a group's formatting.

Understand priority

BetterChat uses the lowest numerical priority as the highest priority when selecting a player's primary group. A player in several groups may therefore inherit a different primary format than expected. Inspect all memberships rather than repeatedly making the same group more privileged. Document your priority order so future roles can fit into it.

Prefer command-based edits for a simple change. If editing the JSON data directly, use a controlled maintenance window and the author's storage guidance so an in-memory save cannot overwrite your changes. Validate the local copy before uploading it and preserve the previous file.

Verify real chat output

1. Send a harmless message from the test account and check the title, colour and name.
2. Send a message from an ordinary player and confirm that their format remains readable.
3. Check a player with multiple roles to confirm the intended priority.
4. Inspect whether messages appear once, and whether another chat integration changes the result.

Duplicate messages can be caused by another incompatible chat formatter. Identify the overlapping component and read its integration documentation before disabling anything. Logging, moderation and relay plugins may depend on the existing setup, so avoid removing them casually.

Finish or restore

If the role is only a trial, remove that test membership with the documented `chat user remove STEAM64 communityhelpers` form. Restore the retained group definition if formatting is wrong. Do not reset all permission groups to fix a cosmetic problem.

For support, provide the version, redacted group definitions, memberships and a screenshot with private messages hidden. Explain the expected primary group and actual output. Never include RCON secrets or complete player chat histories in a public report.

Related guides

- Manage Oxide permission groups
- Validate JSON before uploading

Further reading

- uMod: Better Chat
- uMod: plugin permissions
- uMod: Oxide plugin configuration

Looking for another game? Browse game server setup and troubleshooting guides.

Configure Rust Backpacks permissions and preserve stored items

The uMod Backpacks plugin adds persistent inventory space with its own permissions and rules. Treat its stored contents as player property. Enabling a button, changing capacity and deciding what happens on death are separate configuration choices.

Protect the existing storage

Record the Backpacks version and preserve its configuration and complete plugin data directory. Include the active world and related inventory integrations in a consistent backup before major changes. Arrange a normal stop if necessary to copy state reliably. Keep the backup outside the live server, with its version and date recorded.

Inspect the current configuration rather than assuming an older tutorial matches. Modern capacity permissions and profiles differ from some legacy row-based examples. Do not delete a data directory to force the plugin to recreate it; that can remove the items you are trying to protect.

Enable a limited trial

Grant the base use permission to one verified account through the trusted server console:

```
oxide.grant user STEAM64 backpacks.use
```

The player can then try `/backpack`. The optional GUI button uses the separate `backpacks.gui` permission. Enable it only if that is part of your intended interface. A missing button does not by itself mean backpack storage is unavailable.

Check capacity and retention rules

Read the configured default size and registered permission sizes for the installed version. Grant only a capacity permission that actually exists in that configuration. Inspect all the player's groups if the resulting capacity differs from your expectation. Avoid experimental dynamic-size features while establishing a reliable baseline.

Review the configured death and wipe policies before placing valuable items inside. A keep-on-death exception is not the same as keeping items across a map wipe. Do not change those policies globally just to debug one player's access. Communicate the actual rules before players rely on the storage.

Verify with disposable items

1. Place a small, counted set of inexpensive items in the test backpack.
2. Close and reopen it, then reconnect and check the quantity.
3. Move the items back to ordinary inventory and confirm nothing duplicated or disappeared.
4. During a planned maintenance window, verify persistence after a normal save and restart.
5. If testing death behaviour, use a consenting test account and disposable inventory under controlled conditions.

Do not trigger a map wipe to test retention. Keep the live world's lifecycle unchanged. Features such as automatic gathering and retrieval add other requirements and interactions; leave them disabled until basic storage works.

Recover carefully

If a capacity reduction would leave occupied slots outside the new limit, stop and plan a supported migration with sufficient ordinary storage. Do not assume the plugin will safely dispose of excess contents. Restore the retained configuration if needed while preserving current player data.

Provide support with the plugin version, affected Steam64 ID privately, assigned permissions and relevant redacted errors. Never run erase or wipe commands to investigate a display problem. Where recovery requires restoring inventory data, coordinate it to avoid overwriting legitimate items acquired since the backup.

Related guides

- Plan safe stack-size changes
- Trace direct and inherited permissions

Further reading

- uMod: Backpacks
- uMod: plugin permissions
- uMod: Oxide plugin configuration

Looking for another game? Browse game server setup and troubleshooting guides.

Connect RustAdmin to Rust RCON without exposing credentials

RCON provides remote control of your Rust server. Treat its password like an administrator credential. The game connection port and the remote-console port serve different purposes; entering the game port in RustAdmin will not establish an administrative connection.

Collect the correct endpoint

Check your service's assigned public address and allocated RCON TCP port. Confirm the running startup configuration uses that allocation. In a hosted container, the public endpoint can differ from the address visible inside the container. Do not replace the bind address with your home IP or assume localhost reaches the hosted server.

Record the existing RCON settings privately. Back up the relevant configuration before changing anything. If RCON is already used by monitoring or administration tools, plan the password change with those administrators so their clients can be updated together.

Configure a compatible connection

1. Check that the Rust server has finished starting successfully.
2. For a WebRCON-compatible client, confirm `rcon.web 1` is configured. This selects the protocol; it does not by itself encrypt the connection.
3. Set a long unique `rcon.password` through the supported startup or configuration method. Avoid passwords reused for billing, Steam or email.
4. Apply changes through a normal scheduled restart if required. Keep the previous configuration available for rollback.
5. In your trusted RustAdmin installation, choose the connection mode matching the server, then enter the public address, allocated RCON port and password in the appropriate fields. Client layouts differ, so use the help for your installed edition.

Test with a harmless command

Request a player list or server status first. Confirm the response belongs to your server before issuing any command that changes it. Check that console output remains available from the game panel as a recovery route. Close duplicate client sessions during diagnosis so you can identify the attempt being tested.

Interpret the failure

A timeout suggests the endpoint, allocation, firewall path or listener needs checking. Immediate refusal can mean the process has not opened that port. A successful socket connection followed by authentication failure points towards credentials or protocol. These are clues rather than definitive diagnoses; save the exact error and time for support.

Ask for a narrowly scoped RCON allowlist or a managed private connection where available. Do not open every port or disable the host firewall to test one listener. Some browser-based remote consoles forward secrets through another service; use only a client you trust and avoid putting the password in shared links or screenshots.

Finish safely

Remove stale saved credentials from unused tools and rotate the password if it was exposed. A connection working from one location does not prove access is restricted elsewhere. Tell support which client and protocol worked, with the address and time, while keeping secrets out of the ticket body. Never upload an unredacted configuration file publicly.

Related guides

- Diagnose Rust game, query and RCON ports
- Assign owner and moderator access

Further reading

- Facepunch: creating a Rust server

- RustAdmin official client website

Looking for another game? Browse game server setup and troubleshooting guides.

Create and test a Rust Kits starter kit safely

A starter kit affects your server's economy as well as its first impression. Test a small bundle before making it available to everyone. This guide refers to the uMod Rust Kits plugin maintained by k1lly0u; similarly named plugins can use different commands and data formats.

Confirm the plugin and preserve state

Record the installed plugin name and version. Retain a copy of its configuration and complete kit data, including player usage records. Back up world and player inventory state before a major kit change. Do not clear cooldown or redemption data to make a test pass; that would also change existing players' entitlements.

Check the author's dependency and compatibility notes for your installed release. If the menu is missing images or fails to open, inspect the first console error rather than repeatedly importing another kit plugin's files.

Create a harmless trial

The documented administrator permission is `kits.admin`. Limit it to the administrator building the kit. On a dedicated test account, store personal items safely and arrange only the inexpensive test items intended for the bundle. Do not discard valuable inventory to prepare the sample.

Use `/kit new` or `/kit add` in game to enter the supported creator. The author's workflow lets you copy the prepared inventory into the kit, then set its details. Choose a clear unique name and inspect quantities, equipment and attachments before saving.

Choose restrictions deliberately

Define whether the kit is public, limited to a role or intended for a particular event. Set appropriate cooldown and usage limits using the creator's supported fields. Review any costs, required authorisation and configured commands. Avoid including command rewards unless you understand exactly which console action they execute.

Do not enable an automatic spawn kit merely to simplify a one-time test. Repeated deaths, respawns or reconnects can make an unintended autokit much more influential than an ordinary redeemable bundle.

Test as an ordinary player

1. Use a separate normal account without the kit administrator permission.
2. Open `/kit` and verify which kits are visible.
3. Redeem the trial by its configured name and count every received item.
4. Try again to confirm the intended cooldown or maximum-use response.
5. Check inventory capacity behaviour and inspect the console for errors.

An administrator may bypass restrictions depending on configuration. A successful admin redemption is therefore insufficient evidence that ordinary players face the intended rules. Never test with unlimited redemptions of valuable items on the live economy.

Retain a narrow rollback

If the kit is wrong, withdraw that trial from availability using the author's supported editing workflow and restore its previous definition if needed. Preserve player usage data. Changing kit contents does not reclaim items already distributed; plan a fair response before a broad rollout.

Send support the plugin version, kit name, expected restriction and redacted error. Do not run global usage-reset commands as a troubleshooting shortcut, and do not replace the entire kit database with an unrelated example.

Related guides

- Grant only the required permission
- Recover a faulty plugin configuration

Further reading

- uMod: Rust Kits
- uMod: plugin permissions
- uMod: Oxide plugin configuration

Looking for another game? Browse game server setup and troubleshooting guides.

Diagnose Rust game, query and RCON connection ports

A Rust server can accept players while missing from the browser, or appear online while its remote console fails. Diagnose each connection separately. A single successful port test does not prove that all three functions work.

Map the assigned ports

Record the public service address and each allocation. Rust uses UDP for `server.port`, UDP for `server.queryport` and TCP for `rcon.port`. Query and game ports must be distinct. Do not choose an adjacent port just because an online example uses it; the hosting platform must allocate and route that endpoint to your container.

Keep a private copy of the current startup settings and configuration. If changing network variables requires a restart, arrange a short maintenance window. World files, player databases and Oxide data do not need to be deleted to investigate reachability.

Test the game connection first

1. Wait for startup to complete and check for repeated crashes or bind errors.
2. From a Rust client outside the hosted container, try connecting directly to the assigned game address and port.
3. Record whether it times out, rejects the connection or reaches the loading screen. Note the exact time and the matching server-console output.
4. If another player can connect, compare locations and endpoints before changing the server. A local routing or client problem remains possible.

A generic TCP port checker cannot validate a UDP game listener. Likewise, a UDP probe with no reply is not conclusive unless it sends the expected protocol. Use actual Rust connection behaviour as the primary client test.

Then check discovery

If direct joining works but discovery fails, compare the configured query endpoint with the assigned query allocation. Allow time for listing updates and check the selected browser filters. Avoid changing the world name, identity or seed while testing browser visibility. Ask support to verify query traffic routing when the allocated values match but queries still fail.

Check RCON independently

Use a compatible trusted RCON client against the assigned TCP port. Confirm WebRCON mode and the password privately. A working game connection does not establish that the RCON listener is configured or permitted. Do not share credentials when reporting a timeout; support can investigate the network path without receiving the password in a public message.

Escalate with a small evidence set

Send the service ID, public endpoints, configured port values, UTC timestamp, client location and exact result for each test. Redact authentication strings and unrelated player details. Request checks for the specific protocol and allocation, not a blanket firewall exception. If a recent port change caused the issue, restore the recorded values and restart normally before attempting further changes.

Rust+ connectivity is another service with its own requirements. Keep it separate from game, discovery and RCON diagnosis so changing one feature does not obscure the original fault.

Related guides

- Set up a secure RCON connection
- Find the active identity configuration

Further reading

- Facepunch: creating a Rust server
- Facepunch: Rust+ server requirements

Looking for another game? Browse game server setup and troubleshooting guides.

Diagnose Rust server lag and entity load without wiping

Rubber-banding, slow inventory actions and low client frame rate can have different causes. Collect a short, timed example before changing world settings. A large entity count can be relevant, but it is not sufficient evidence that deleting players' buildings will fix the problem.

Separate the symptoms

Ask two players in different locations to report the same event and time. Note whether everyone sees delayed actions, only one connection suffers, or one player's rendering slows down. Compare the game panel's existing CPU, memory and network history with that period. A brief CPU average can conceal a main-thread stall, so retain the timeline rather than only a current screenshot.

Record player count, map size, world age, plugin versions and recent configuration changes. Include any entity-count information already available through trusted server telemetry. Do not run unverified commands from forum posts to enumerate or remove entities.

Look for correlated activity

1. Check console and Oxide logs around the reported time for repeated plugin exceptions.
2. Compare scheduled saves, backups and plugin tasks with the lag window.
3. Check whether storage is nearly full or memory repeatedly approaches its limit.
4. Look for a recent plugin or settings change that matches the first occurrence.
5. Repeat a short observation during a similar player load before drawing a conclusion.

Keep the original logs and redact account details before sharing them. Avoid mass plugin unloading during peak play; inventory, protection and economy plugins may provide essential rules. A temporarily quiet server is not necessarily proof that a particular plugin caused the issue.

Test one reversible change

Back up world, player data, plugin data and any external databases before changing behaviour. Record the current configuration and choose a quiet maintenance window. If evidence points to one plugin, follow its author's supported diagnostic workflow and consider a separate test instance. Compare the same activity before and after; restore the retained settings if there is no clear improvement.

Do not change map seed, remove save files, purge entities or reset databases as a performance experiment. Those actions destroy evidence and can remove customer progress without addressing the original bottleneck.

Use profiling only when justified

Facepunch provides a server profiler enabled through the `-enableProfiler` startup argument. It requires a restart and can add overhead; recording may substantially degrade performance. Arrange this with support rather than enabling it permanently during an outage.

If a profile is requested, follow the current Facepunch instructions for a short capture and wait for it to finish saving before changing settings or starting another recording. Keep adequate storage space. Remove the profiling startup argument after the investigation and restart normally in the agreed window.

Escalate with useful evidence

Provide the service ID, UTC timestamps, player-count range, affected actions, redacted errors and relevant resource graphs. State which single test was performed and its result. Profiling archives may contain operational details; share them privately with support. Keep the working configuration and backups until a comparable busy period confirms the improvement.

Related guides

- Test gathering changes in isolation
- Protect the active world and identity

Further reading

- Facepunch: server profiler
- uMod: Oxide plugin configuration

Looking for another game? Browse game server setup and troubleshooting guides.

Find Rust server.cfg and the correct identity save folder

Rust keeps each server identity in its own directory. Editing a similarly named folder can appear to do nothing; changing the identity can make an existing world look missing. Locate the running instance before moving or editing any files.

Identify the active instance

1. Record the startup configuration shown for your service. Find `server.identity`, including its exact spelling and case. If the panel does not expose it, ask support to identify it rather than trying different names.
2. In the server files, locate `server/IDENTITY/` relative to the Rust installation. Replace IDENTITY with the recorded value. A hosted container may show a different installation root; the relative identity directory is what matters.
3. Check recent console output and file timestamps against the last successful session. Old identities can remain from previous configurations. Do not delete them just because another directory looks newer.

Save a recoverable copy

Schedule a short maintenance window, announce it, and stop the server normally. Download or back up the complete active identity directory, the startup settings and any Oxide configuration and data. Keep related database files together, including matching journal or WAL files. A lone map save is not necessarily a complete copy of player and plugin state.

Edit server.cfg

The normal location is `server/IDENTITY/cfg/server.cfg`. Preserve the existing file before making one small change. Configuration lines use console-variable syntax without the command-line plus sign:

```
server.hostname "My Rust community"
```

Keep quotes around text containing spaces. Do not paste a complete example startup line into this file. Network endpoints, world parameters and identity values should remain unchanged while testing a hostname edit. Some hosting startup settings are managed separately; record those values too so competing settings can be traced.

Confirm the intended file was loaded

Start the server and wait for normal startup completion. Check the hostname from an external client and inspect the console for configuration errors. Join and verify a familiar building or player location. If the world seems new, stop immediately and compare the identity and startup settings with your notes; do not generate additional saves or attempt a wipe as a repair.

A successful name change proves the selected configuration is being read. It does not prove every setting belongs there. Make further changes individually, checking the relevant variable documentation and the hosting settings each time. Preserve the working copy until a later restart has also succeeded.

If edits disappear

Compare the saved file after a normal restart. A startup template or management integration may regenerate a setting. Report the file path, setting name and time of the restart to support. Remove passwords, access tokens and player database contents from screenshots. Restoring the original configuration and startup values is the first rollback; replacing world files is unnecessary for a configuration-only change.

Related guides

- Grant Rust administrator access safely
- Validate plugin JSON configuration

Further reading

- Facepunch: creating a Rust server

- Facepunch: getting started with server files

Looking for another game? Browse game server setup and troubleshooting guides.

Grant Rust owner or moderator access using Steam IDs

Rust administrator access is tied to a Steam account, not a display name or game-panel login. Grant the lowest role the person needs, then verify with that account. An owner has substantially more authority than an ordinary player.

Prepare and identify the player

Use a trusted game-panel server console or an already secured RCON connection. Never enter an owner command in public chat. Record the current administrator list privately and back up the active identity configuration before changing access. If you will edit files directly, stop the server first and retain the complete original file.

Ask the intended administrator to join. Run `users` in the server console and match the account carefully. Use its full Steam64 identifier. Similar display names and renamed accounts are common; a remembered name is not enough evidence. Have the person confirm their own profile before assigning a role.

Assign one role

The command forms below use a placeholder. Replace STEAM64 with the verified numeric account ID and use a short descriptive name:

```
moderatorid STEAM64 "Moderator name"
ownerid STEAM64 "Owner name"
```

Run only the command for the intended role. Rust distinguishes moderator authentication level 1 from owner authentication level 2. Giving everyone owner access to bypass a plugin error expands their control over the server and does not establish which permission was missing.

Reconnect and verify

1. Have the player disconnect completely and reconnect after the assignment.
2. Test a harmless administrative action appropriate to the role. Avoid testing item creation, player bans or destructive world commands on a live community.
3. Check the console for the result and confirm an unrelated ordinary player has not gained administrator behaviour.
4. Record who approved the access and when it should be reviewed.

Current Facepunch instructions state that these administrator changes are saved automatically. Do not repeatedly overwrite configuration files to force persistence. If access disappears after restart, investigate the active identity and any management process regenerating its configuration.

Separate Rust access from plugin permissions

Oxide plugins may require their own named permission even when the user is a Rust administrator. Read that plugin's documentation and inspect the user's Oxide permissions. Grant only the relevant permission. Conversely, adding a cosmetic chat title should not require making someone a Rust owner.

Recover from a wrong assignment

Keep the console available while testing so you do not depend on the newly authorised player to recover access. If the wrong account was authorised, stop further testing and remove that account using the current Facepunch administrator guidance or ask support to verify the saved administrator configuration. Restore a retained configuration only during a controlled stop, accounting for any legitimate access changes since the backup.

Support needs the service ID, timestamp, intended role and redacted console result. Do not publish the full administrator list, RCON password or another customer's account information.

Related guides

- Connect securely through RCON

- Manage Oxide groups and permissions

Further reading

- Facepunch: creating a Rust server
- uMod: plugin permissions

Looking for another game? Browse game server setup and troubleshooting guides.

Manage Oxide plugin permissions without granting everyone admin

Oxide permissions decide which plugin features a player may use. Rust owner status, a chat prefix and a plugin permission are different things. Diagnose the named feature before changing access for the entire server.

Prepare a permission record

Confirm the plugin loaded successfully and find the exact permission names in its current author documentation. A permission cannot be granted usefully if the plugin never registered it. Record the plugin version, the user's Steam64 ID and current groups. Back up Oxide configuration, permission data and any affected plugin data before restructuring roles.

Use the trusted server console for these commands. Replace uppercase placeholders with real values; the examples are command forms, not accounts that already exist on your server. Prefer a full Steam64 ID over an ambiguous display name.

Inspect before granting

```
oxide.show user STEAM64
oxide.show perm PLUGIN.PERMISSION
```

Look for both direct grants and inherited access. Membership of multiple groups can explain why revoking one direct permission makes no visible difference. Do not remove existing groups until you understand what other features they provide.

Apply a narrow change

For a single-person trial, grant the specific permission directly:

```
oxide.grant user STEAM64 PLUGIN.PERMISSION
```

If the same role is needed for several people, create a deliberately named group with the supported Oxide group command, then grant the documented permission and add the test member:

```
oxide.group add helpers
oxide.grant group helpers PLUGIN.PERMISSION
oxide.usergroup add STEAM64 helpers
```

Do not substitute a wildcard permission. The default group includes ordinary players, so granting it a permission changes the experience for the whole community. Administrator groups should not become a convenient container for cosmetic or paid features.

Test the ordinary-player experience

1. Have the test account reconnect if the plugin requires it, then try only the intended feature.
2. Check for the expected success message and any corresponding plugin error.
3. Verify that an unrelated player still cannot use the feature.
4. Check that the new role has not unexpectedly unlocked inventory access, teleport administration or other privileged actions.

Testing solely with a server owner can hide a missing permission or bypass a normal restriction. Where possible, use a separate ordinary account for the acceptance check.

Revoke precisely and preserve evidence

```
oxide.revoke user STEAM64 PLUGIN.PERMISSION
oxide.revoke group helpers PLUGIN.PERMISSION
oxide.usergroup remove STEAM64 helpers
```

Run only the rollback commands matching the change you actually made. Removing a group membership can affect multiple permissions at once. Record the before-and-after output privately so you can explain the result without resetting the complete permission database.

If the permission is reported as unknown, inspect plugin loading and spelling first. If it exists but behaviour is unchanged, check inherited groups, plugin-specific restrictions and the author documentation. Send redacted results to support rather than granting global access as a workaround.

Related guides

- Understand Rust administrator authentication
- Configure BetterChat roles

Further reading

- uMod: plugin permissions
- uMod: Oxide plugin configuration

Looking for another game? Browse game server setup and troubleshooting guides.

Repair a Rust plugin JSON configuration without losing data

A missing comma can stop a Rust plugin from loading its settings. Deleting the plugin's data directory may make the error disappear while losing player progress. Work on the exact configuration file and preserve both the broken and last working versions.

Find the right file

For an Oxide installation, configurable plugins normally create a JSON file under `oxide/config/` matching the plugin name. For example, a configurable `ExamplePlugin.cs` normally uses `ExamplePlugin.json`. Hosted paths can vary. Keep the existing name and extension, and check `oxide/logs/` for the first error associated with that plugin.

Configuration describes behaviour; files under plugin data directories may contain balances, claims, inventories or usage records. Read the plugin author's storage notes before touching either. Do not assume a file containing JSON is disposable configuration.

Preserve a consistent rollback

1. Stop further edits and copy the faulty file to your own computer.
2. Retain the last known working configuration separately, with a date and plugin version.
3. Back up the relevant plugin data and any external database it uses. If consistent copying requires a normal server stop, schedule that first.
4. Record the setting you intended to change. Fix that one edit before trying unrelated adjustments.

Validate syntax locally

Use a local editor with JSON validation. If Python is installed on your own computer, its standard JSON formatter can check a downloaded copy:

```
python3 -m json.tool ExamplePlugin.json
```

This checks JSON syntax, not whether the plugin accepts a setting. Use straight double quotes around keys and strings, retain braces and brackets, and avoid trailing commas or comments. Do not upload configuration containing passwords, webhook URLs or database credentials to a public validation website.

An error line often marks where parsing became impossible, rather than the original mistake. Inspect the preceding line too. Compare with your retained copy instead of repeatedly replacing large sections. Ensure an editor has not silently saved the file with an extra extension.

Apply and watch the result

Upload only the corrected configuration to its original location. Apply it with the documented plugin reload, for example `oxide.reload ExamplePlugin`, from the trusted server console. Read the new console output immediately. A successful JSON check followed by a plugin error may indicate an unsupported value, outdated schema or missing dependency.

If it still fails

Restore the retained working configuration and reload again. Do not overwrite newer player data just to roll back a setting. If an upgrade changed the schema, compare the installed version with the author's current documentation and test migration in a separate environment where possible.

For support, provide the plugin name and version, the first relevant error and a minimal redacted section around the failing setting. Confirm whether restoring the prior configuration fixed it. Never include tokens, database passwords or whole player-data archives in a public post.

Related guides

- Change Stack Size Controller carefully
- Adjust Gather Manager rates

Further reading

- uMod: Oxide plugin configuration
- Python: local JSON validation

Looking for another game? Browse game server setup and troubleshooting guides.

Change Rust server description and header without changing the world

Use this procedure when players can join but your Rust listing has the wrong description, website or image. These are presentation changes. They should not require a new identity, seed, map or installation. Have game-panel file and startup access before beginning; if those settings are locked, ask support to update only the named presentation fields.

Record what is running

Save a private copy of the current startup values and active `server.cfg`. Note the server identity, connection address and map name so you can confirm they remain unchanged. Take a screenshot of the current public listing without exposing your RCON password. Check whether the panel provides separate name, description, website and header fields. Use that supported editor rather than adding competing overrides.

Facepunch documents `server.hostname`, `server.description`, `server.url` and `server.headerimage` as the relevant settings. The location that wins can depend on your startup template. Editing an unused configuration file will not change the visible listing.

Prepare the public information

1. Write a short, accurate name and description. Put important rules near the beginning: language, group limit and whether the server is modded. Keep access passwords and private staff contacts out of these public fields.
2. Use a website address you control. Open it in a private browser window and confirm that a player can read it without signing in.
3. Host the header image at a stable public HTTPS address that returns the image itself. A sharing page, expiring upload link or authenticated dashboard is unsuitable.
4. Keep the old image and URL available until you have verified the replacement. Use a new filename when changing artwork, so you can distinguish the versions during checks.

5. Change only the presentation fields in the supported panel editor or active configuration. Preserve quoting around text containing spaces. Review the diff before saving.

Check the result

If the hosting template requires a restart, announce a short maintenance window and stop normally. Do not use a forced kill to refresh an image. Once the server is ready, connect through the original address and confirm the expected world and player state still load. Open its listing from a fresh browser view and compare the description, website destination and header against your notes.

A cached image can remain visible after the text changes. First verify that the new URL opens directly outside your account. Do not change identity or regenerate the map to resolve a presentation problem. If the value reverts at every start, check for a panel startup override with support rather than repeatedly editing generated files.

Undo a bad change

Restore the previous presentation values and old image URL, then repeat only the refresh or normal restart your template needs. Keep the current world files untouched. This guide applies to the documented Rust Dedicated settings; image handling and which fields the panel exposes may vary with the installed template.

Related help

- Find Rust server.cfg and the correct identity save folder
- Schedule game server backups and check retention

Choose Rust server browser tags that match your rules

Rust browser tags help players filter servers before joining. A wrong tag can make a healthy server difficult to find or promise rules that you do not actually enforce. This guide changes listing labels only. It does not schedule wipes, install a rules plugin or convert your server between game modes.

Audit the advertised rules

Write down your actual wipe cadence, region and gameplay rules. Check your announcement channel rather than guessing from a previous server name. Record the existing `server.tags` value and save the active configuration before editing. If the panel manages tags through a startup field, use that field as the source of truth.

The official tag table lists supported values and mutually exclusive groups. The browser displays up to four tags. Values such as `weekly` and `monthly` belong to the same schedule group and should not be combined. An unsupported word is not a reliable substitute for a documented tag.

Build a small, truthful tag set

1. Choose one supported wipe-schedule tag that matches your published cadence. If the schedule is unusual, describe it accurately in your public description instead of selecting contradictory categories.
2. Choose a difficulty category only when it matches the running configuration. Do not advertise vanilla simply because it attracts searches if plugins materially change play.
3. Add an appropriate supported region value, using the spelling and case in the official table.
4. Review game-mode settings that may add tags automatically. Facepunch specifically notes that `server.pve` adds PvE. Avoid manually fighting a label that follows an actual gameplay setting.
5. Save the comma-separated list. For example, `server.tags weekly,vanilla,roleplay` illustrates syntax; use those values only if all three describe your server.

Verify with a player view

Reload settings or restart normally according to the hosting template. Keep the old list nearby. Find the server without filters first, then apply one intended filter at a time. Check the displayed tags and ask another player to repeat the search from their own client. Record the time and filters used so support can reproduce a missing result.

Allow for browser refresh behaviour before making another edit. If direct joining works but filtering fails, compare the effective tag string with the supported table and remove conflicts first. Changing game or query ports is a separate investigation and should not be your first response to a category mismatch.

Recover from an incorrect listing

Restore the previous tag string if the new set advertises the wrong rules. Recheck that the server remains reachable and the public description is consistent with the restored values. Do not run a wipe to make a listing label true after the fact; agree and announce the schedule independently.

The tag catalogue can change with Rust releases. Revisit the upstream table when adding a new category rather than copying a long string from another community. A successful check means players can find the server under its honest categories, not that every possible search must include it.

Related help

- Find Rust server.cfg and the correct identity save folder
- Diagnose Rust game, query and RCON connection ports

Restore Rust+ pairing after a server move without resetting players

If Rust itself works after a move but Rust+ reports the server offline, investigate the companion connection separately. Do not erase player records or regenerate the world to repair the app. You need the game console, access to the active identity folder and the allocation details supplied for your service.

Preserve the existing pairing state

Before changing anything, take a backup and record the identity in use. Facepunch identifies `companion.id` as the server's Rust+ identity and `player.tokens` as player pairing information. Removing tokens forces players to pair again. Keep these files private, and retain them with the server's consistent save set during an authorised migration.

If the move has already happened, compare filenames and backup dates with the old identity. Do not copy files from a different customer's server or merge two identities. When the expected files are missing, ask support to inspect the retained migration copy. Avoid restoring the entire world merely to replace a pairing identifier.

Check the companion listener

1. Run `app.info` in the server console. Record the companion address and port, excluding any tokens from screenshots or tickets.
2. Compare the reported port with your game-panel allocations. The Rust+ documentation explains that the companion listener needs TCP reachability and that `app.port` can select its port.
3. Ask support to provide or confirm a dedicated companion allocation when the reported port is not assigned to you. A client-side panel setting cannot grant use of an arbitrary host port.

4. If support supplies a replacement, set `app.port` to that allocation and schedule a normal restart. The documented port must be at least 10000; do not choose a number merely because it meets that minimum.
5. If registration or push notifications still fail, include the relevant error and timestamp in a ticket. Support should check outbound access to the documented Facepunch companion service and the specific inbound allocation.

Prove the repair

After startup, run `app.info` again. Confirm the expected listener and open Rust+ using an existing pairing. Test a simple read-only action such as viewing server information. Ask one previously paired player to check too. Record whether the app reaches the server and whether notifications work; these are different checks.

If existing pairing works, there is no reason to reset everyone. If only one phone has trouble while other pairings work, investigate that account or app before changing the server identity. A reachable game port alone does not prove that the companion TCP path is healthy.

Rollback and limits

Restore the previous `app.port` setting only if support confirms that allocation is still valid. Retain the backup and leave player data untouched. Companion details may evolve with Rust updates, so compare your console output with the current official documentation. Do not post pairing tokens, raw identity files or an unrestricted firewall rule in public support discussions.

Related help

- Find Rust server.cfg and the correct identity save folder
- Diagnose Rust game, query and RCON connection ports
- Schedule game server backups and check retention

Take a Rust save checkpoint before a planned panel restart

Use this checklist before planned Rust maintenance when you want a clear recovery point. A game save and a panel backup solve different problems: one records the current world, while the other preserves a copy that can survive later damage. Neither a green panel button nor a restart timer proves that a usable recovery copy exists.

Prepare a quiet maintenance window

Announce the time and timezone, ask players to finish active trades and record the server identity. Check that no backup restore or update is already running. Save the current configuration and note the latest successful backup's timestamp, status and size. Ensure the backup destination has enough allowance before starting another archive.

For modded servers, read each important plugin's persistence instructions. A plugin may hold separate state, use a database or write on a different lifecycle event. Do not assume a base-game save automatically commits every external economy or inventory system. If its documentation is unclear, stop and ask the plugin maintainer before promising a lossless live snapshot.

Create the checkpoint

1. Watch the console during the final maintenance warning. Avoid starting another long-running administrative operation.
2. Use the normal game-panel stop action configured for Rust. If you administer through an authenticated Rust client instead, Facepunch documents `sv quit` as saving and closing the server. Do not type a client-only quit command and assume the remote process stopped.

3. Wait for the process to exit. If the panel stays stopping, preserve the console output and ask support rather than repeatedly forcing it down.
4. With the server stopped, inspect the expected identity folder and record the save filenames and modification times. Do not rename saves to match a different world or protocol.
5. Create a panel backup of the stopped service. Wait for its completed status and record the backup identifier. Download an authorised copy if your plan and policy require one.

Verify before continuing

Check that the checkpoint contains the active identity and required plugin state, not just a similarly named folder. Facepunch's file reference distinguishes map saves, map terrain files and other player data. Keep the set together. Inspect the archive listing where available without unpacking it over the running service.

Now perform the planned maintenance and start normally. Join with a known player account and check an agreed landmark or stored item. Compare the result with your recorded checkpoint time. These checks are verification steps for you to perform, not a claim that CloudExa has tested this particular server.

If startup or verification fails

Keep the failed-start logs and stop further writes before considering recovery. Do not immediately restore an old backup over new player progress. Compare the installed Rust build, plugin versions and save compatibility first, then agree the recovery point with affected staff.

A forced Rust wipe can change save compatibility; this checklist does not bypass it. Keep previous archives until the maintenance is accepted and retention policy permits removal. If a graceful stop consistently exceeds its allowance, have support investigate that timeout separately instead of making forced kills part of the routine.

Related help

- Find Rust server.cfg and the correct identity save folder
- Schedule game server backups and check retention

Preflight a Rust custom map download before the next wipe

A custom map can work for the owner and still fail for every joining player if its URL requires a login or returns a download page. Check the delivery path before your announced wipe. This procedure prepares a future map; it does not authorise replacing the current world during an active wipe.

Keep the current release recoverable

Save the existing startup settings, map URL, map file and current backup identifier. Choose a maintenance date with staff and decide which player data your announced wipe policy preserves. Keep the old download available while the new release is being checked. Use an isolated test service with its own identity and allocations; ask support if you do not have one.

Obtain the map from its authorised author and record the exact release filename. Check the author's Rust-version requirements, distribution permission and any required server extensions. Do not upload a paid map to a publicly indexed repository unless its licence explicitly allows that delivery method.

Check the URL as a player would

1. Publish the map at a stable public HTTPS address under your control or an authorised download service. Use a versioned filename rather than replacing unrelated content behind a familiar name.
2. Open the address in a private browser session with no provider login. Confirm it returns the map download itself, not an HTML page, advertising interstitial or sign-in screen.
3. Download a second copy from another network if practical. Compare its size and a locally calculated checksum with the author's file. Matching checksums verify identical bytes, not

that the map is safe or compatible.
4. Remove short-lived access tokens from the delivery design. Players may need the file long after the administrator's initial test.
5. Check any bandwidth or download limits with the file host. A successful single download does not demonstrate capacity for an entire wipe launch.

Test without touching production

Facepunch's custom-map instructions use `server.levelurl` for the direct download and advise removing procedural seed and world-size arguments for this configuration. Apply those changes only to the test service first, leaving its unique identity and assigned ports in place. A local `file:///` reference is not a substitute for the URL that remote players need.

Start the test instance and read its map-loading output. Join from a normal Rust client using the matching supported game version. Check terrain, spawn and access to a few key monuments. Record any author-recommended tests and the result rather than simply writing “map works.” Stop the test if it tries to load a production identity.

Release or roll back

Only after the test and announced wipe window should you apply the approved URL to production through the supported startup editor. Take the agreed checkpoint first and watch the first start. If the map cannot be downloaded, stop and restore the previous startup settings before further changes; do not delete the current saves to make startup continue.

Map compatibility and server extensions change between Rust releases. Repeat the preflight for each map revision. Keep the exact map file, URL and author release notes with your backup record so a later restore is not dependent on remembering which “final” upload was used.

Related help

- Find Rust server.cfg and the correct identity save folder
- Schedule game server backups and check retention

Diagnose Rust custom map checksum mismatches without deleting the world

A “World File Mismatch” or map checksum error usually needs an investigation of which map bytes the server and client received. Deleting the world is not a diagnostic step. Begin with the service stopped if players cannot join, and preserve the current files before experimenting with names or URLs.

Capture the mismatch

Record the exact client error, time, Rust version and affected connection address. Ask whether everyone is affected or just players who downloaded an earlier map. Save the server's startup output showing the map name, and note the effective `server.levelurl`. Keep access tokens out of tickets; a public map URL should not contain administrator credentials.

Take a backup of the identity folder, current map and startup configuration. Keep the original downloaded map separately. Do not rename a saved world to force a different map to accept it, and do not replace the production map with a file sent by an unknown player.

Compare the delivery chain

1. Open the configured URL outside your file-host account. Confirm the response is the actual map. Facepunch notes that an unrelated HTTP page can be returned where Rust expects map content.
2. Download the file locally and compare its byte size and checksum with your authorised release copy. Record the algorithm used so staff compare like with like.
3. Compare that release with the map present on the server. Use an authorised download through the game panel; do not edit it in place.
4. Review recent changes. Was a different map uploaded under the same filename? Was an existing map renamed? Did the URL start redirecting to a login page?

5. Check the author's release requirements against the installed Rust build. A matching filename alone does not establish matching content or compatibility.

Choose the smallest safe repair

If the URL returns a web page, repair the public download at the file host and recheck it before changing the server. If the map release changed, use an explicit new versioned file and an isolated test instance. The hosting instructions explain why players need a direct, continuously available download.

Facepunch warns that custom maps are not downloaded afresh on every start when already present, and that mid-wipe terrain changes can leave existing entities in inappropriate positions. Therefore, do not follow a cache-deletion recipe against a live world without a reviewed recovery plan. Ask support and the map author to agree the exact files involved, expected save compatibility and maintenance window.

Verify and recover

Test the corrected release on a separate identity and have both a previously affected player and a fresh downloader join. Record which map version each received. Confirm terrain and representative buildings before returning production to service.

If checks fail, stop the test and retain the original production map and save set. Restore the original URL and configuration only with the corresponding unchanged files. Keep evidence of the download response and checksums for support. This guide intentionally avoids renaming protocol-version saves or deleting player caches as a blanket fix; those actions can hide the actual distribution error.

Related help

- Find Rust server.cfg and the correct identity save folder
- Schedule game server backups and check retention
- Diagnose Rust game, query and RCON connection ports

Move Rust plugin data to a test server without losing state

Copying a Rust plugin's source file does not necessarily copy its saved state. Use this workflow to prepare a test instance before an update or migration. It is intended for administrators with panel file access and permission to copy their own server data. The production server remains the authoritative copy throughout the test.

Inventory the plugin's state

Record the plugin name, installed release, dependencies and loader version. Read that release's documentation for configuration, data, language files and any external database. uMod documents the default configuration location as `oxide/config`, but hosts can relocate it. Do not assume all persistent data lives beside the configuration.

List the specific files or folders the plugin author names. Check whether there is a documented export or shutdown procedure. For example, Raid Tracker's upgrade notes distinguish several directories and warn that its v1 and v2 formats are incompatible. That is a reason to follow the exact plugin's instructions, not a universal migration recipe for every plugin.

Make a consistent copy

1. Prepare a separate test service, identity and allocated ports. Disable public announcements and production-facing integrations there.
2. Schedule a short production maintenance window if the plugin cannot provide a documented consistent online export. Stop normally and wait for plugin shutdown activity to complete.
3. Create a full panel backup and record its identifier. Then copy the documented plugin files and related state together, preserving the directory layout and filenames.
4. For an external database, use a consistent export prepared by an authorised database administrator. Put it in a separate test database with a different restricted user. Never

point the test plugin at production.
5. Restart production unchanged once its checkpoint is complete. Keep the captured copy read-only as your baseline and work from another copy in the test instance.

Prove the baseline before upgrading

Install the same plugin release on the test server first. Check startup logs for missing dependencies and data-format errors. Choose a small set of known records relevant to that plugin, such as one permission group or one stored record, and compare them with your checkpoint notes. Avoid testing paid rewards or outgoing webhooks against live accounts.

Only after the unchanged copy behaves correctly should you apply the proposed plugin update to a second test checkpoint. Compare the author's schema-migration instructions carefully. A plugin that starts without a compile error may still have ignored or regenerated its data. Inspect file changes and functional behaviour, not just the green running indicator.

Keep rollback coherent

If the new release fails, stop the test and restore its matching old plugin, configuration and data set together. Do not run an older binary against data already converted by a newer release unless the author explicitly supports it. Preserve the failed test logs for the maintainer.

Do not copy test progress back over production after players have continued playing. A later production rollout needs a fresh checkpoint and the same verified migration steps. Keep both release notes and backup references with the change record so staff can distinguish a tested procedure from a one-off file transfer.

Related help

- Repair a Rust plugin JSON configuration without losing data
- Manage Oxide plugin permissions without granting everyone admin
- Schedule game server backups and check retention

Test a Rust staging branch on a separate server identity

Rust's staging branch lets administrators examine changes before using them on a normal server. Treat it as a separate environment. A staging test should never silently change the software or save files of your active community server. Ask support to supply a separate test service if your game panel does not expose branch selection.

Define the test boundary

Write down the production branch, Rust build, plugin loader and identity. Decide what you are trying to learn: whether a particular plugin loads, whether a custom map starts or whether a reported bug reproduces. A narrow question makes it easier to distinguish a useful test from an unrelated software upgrade.

Give the test service its own installation, identity, connection allocations and credentials. Avoid sharing a writable world directory with production. If you need representative world data, begin with an authorised stopped checkpoint and copy it into the test environment. Keep an untouched copy outside the test installation.

Select the branch through the supported control

1. Confirm the test service can start on its current branch before changing anything. Record that baseline and its configuration.
2. Use the panel's branch setting or have support apply the change. Facepunch's server guide gives `app_update 258550 -beta staging` as a SteamCMD branch example. This is an installer instruction, not a command for the running Rust console.
3. Keep production's automatic updater pointed at its original branch. Do not edit a shared startup template to change one test service.
4. Check that your test client uses the corresponding Rust branch. A stable client failing to join staging is not proof of a broken network.

5. Verify that any loader or plugin build supports the tested branch. An ordinary public-release plugin package is not automatically compatible with unreleased server changes.

Run a controlled check

Start with only the components needed for your question. Capture the server build and startup log, then connect with the matching client. Follow a short written scenario and record expected versus observed behaviour. For a plugin, check its actual function and persistence after a normal restart, not merely whether the source compiled.

If you use copied player data, restrict access and remove or disable production webhooks, reward integrations and external database connections. Do not let staging send real purchase rewards or overwrite shared records. Avoid publishing performance comparisons unless the environments and measurement method genuinely support them.

Return to the public branch safely

Facepunch notes that returning from a beta can require explicitly selecting the `public` branch. Ask support to confirm the effective branch and build after the change. Keep the original pre-test world checkpoint; a branch rollback does not establish that a save modified by a newer build remains compatible.

If the staging save will not load on public, retain the evidence and restore only the test service's original compatible checkpoint. Do not rename save protocol suffixes to bypass Rust's compatibility checks. Record the test conclusion, retire the temporary service through your usual approved process, and leave production unchanged until a separate rollout is planned.

Related help

- Find Rust server.cfg and the correct identity save folder
- Schedule game server backups and check retention
- Fix Rust update and startup failures without wiping