Contributing to Paper

==========================

PaperMC is happy you're willing to contribute to our projects. We are usually

very lentient with all submitted PRs, but there are still some guidelines you

can follow to make the approval process go more smoothly.

\*\*Table of contents:\*\*

<!-- Please don't remove the following comments.

To generate and update the TOC: https://github.com/mzlogin/vim-markdown-toc -->

<!-- vim-markdown-toc GFM -->

\* [Use a Personal Fork and not Organization](#use-a-personal-fork-and-not-organization)

\* [Requirements](#requirements)

\* [Understanding Patches](#understanding-patches)

\* [Adding Patches](#adding-patches)

\* [Modifying Patches](#modifying-patches)

\* [Method 1](#method-1)

\* [Using the Paper tool](#using-the-paper-tool)

\* [Manual method: Stashing](#manual-method-stashing)

\* [Method 2 - Fixup commits](#method-2---fixup-commits)

\* [Manual method](#manual-method)

\* [Automatic method](#automatic-method)

\* [PR Policy](#pr-policy)

\* [Formatting](#formatting)

\* [Patch Notes](#patch-notes)

\* [Obfuscation Helpers](#obfuscation-helpers)

\* [Configuration files](#configuration-files)

\* [PaperConfig example](#paperconfig-example)

\* [PaperWorldConfig example](#paperworldconfig-example)

\* [Frequently Asked Questions](#frequently-asked-questions)

\* [I can't find the NMS file I need!](#i-cant-find-the-nms-file-i-need)

\* [Where can I learn how to name method/field?](#where-can-i-learn-how-to-name-methodfield)

\* [My commit doesn't need a build, what do I do?](#my-commit-doesnt-need-a-build-what-do-i-do)

\* [Patching and building is \*really\* slow, what can I do?](#patching-and-building-is-really-slow-what-can-i-do)

\* [I wrote some API, how do I use it in Paper-Server?](#i-wrote-some-api-how-do-i-use-it-in-paper-server)

<!-- vim-markdown-toc -->

## Use a Personal Fork and not Organization

Paper will routinely modify your PR, whether it's a quick rebase or to take care

of any minor nitpicks we might have. Often, it's better for us to solve these

problems for you than make you go back and forth trying to fix it yourself.

Unfortunately, if you use an organization for your PR, it prevents Paper from

modifying it. This requires us to manually merge your PR, resulting in us

closing the PR instead of marking it as merged.

We much prefer to have PRs show as merged, so please do not use repositories

on organizations for PRs.

See <https://github.com/isaacs/github/issues/1681> for more information on the

issue.

## Requirements

To get started with PRing changes, you'll need the following software, most of

which can be obtained in (most) package managers such as `apt` (Debian / Ubuntu;

you will most likely use this for WSL), `homebrew` (macOS / Linux), and more:

- `git` (package `git` everywhere);

- `patch` (often package `patch`);

- A Java 8 or later JDK (packages vary, use Google/DuckDuckGo/etc.).

If you need one, you can find them on [AdoptOpenJDK](https://adoptopenjdk.net/).

- `maven` (often package `maven`; can be found on

[Apache's site](https://maven.apache.org/download.cgi) too).

If you're on Windows, check

[the section on WSL](#patching-and-building-is-really-slow-what-can-i-do).

If you're compiling with Docker, you can use the

[`adoptopenjdk`](https://hub.docker.com/\_/adoptopenjdk/) images like so:

```console

# docker run -it -v "$(pwd)":/data --rm adoptopenjdk:8-jdk-hotspot bash

Pulling image...

root@abcdefg1234:/# javac -version

javac 1.8.0\_252

```

## Understanding Patches

Paper is mostly patches and extensions to Spigot. These patches/extensions are

split into different directories which target certain parts of the code. These

directories are:

- `Paper-API` - Modifications to `Spigot-API`/`Bukkit`;

- `Paper-MojangAPI` - An API for

[Mojang's Brigadier](https://github.com/Mojang/brigadier);

- `Paper-Server` - Modifications to `Spigot`/`CraftBukkit`.

Because the entire structure is based on patches and git, a basic understanding

of how to use git is required. A basic tutorial can be found here:

<https://git-scm.com/docs/gittutorial>.

Assuming you have already forked the repository:

1. Clone your fork to your local machine;

1. Type `./paper patch` in a terminal to apply the changes from upstream;

1. cd into `Paper-Server` for server changes, and `Paper-API` for API changes.

You can also run `./paper server` or `./paper api` for these same directories

respectively.

1. You can also run `./paper setup`, which allows you to type `paper <command>`

from anywhere in the Paper structure in most cases.

`Paper-Server` and `Paper-API` aren't git repositories in the traditional sense:

- Every single commit in `Paper-Server`/`Paper-API` is a patch;

- `origin/master` points to a directory similar to `Paper-Server`/`Paper-API`

but for Paper;

- Typing `git status` should show that we are 10 or 11 commits ahead of master,

meaning we have 10 or 11 patches Spigot and CraftBukkit don't.

- If it says something like `212 commits ahead, 207 commits behind`,

cd into the root directory of the cloned repository and type `git fetch` to

update your upstream. Setting up a remote for the upstream Paper repository

might be necessary.

## Adding Patches

Adding patches to Paper is very simple:

1. Modify `Paper-Server` and/or `Paper-API` with the appropriate changes;

1. Type `git add .` inside these directories to add your changes;

1. Run `git commit` with the desired patch message;

1. Run `./paper rebuild` in the main directory to convert your commit into a new

patch;

1. PR the generated patch file(s) back to this repository.

Your commit will be converted into a patch that you can then PR into Paper.

> ? Please note that if you have some specific implementation detail you'd like

> to document, you should do so in the patch message \*or\* in comments.

## Modifying Patches

Modifying previous patches is a bit more complex:

### Method 1

This method works by temporarily resetting your `HEAD` to the desired commit to

edit it using `git rebase`.

> ? While in the middle of an edit, you will not be able to compile unless you

> \*also\* reset the opposing module(s) to a related commit. In the API's case,

> you must reset the Server, and reset the API if you're editing the Server.

#### Using the Paper tool

The PaperMC build tool provides a handy command to automatically do this type of

patch modification.

1. Type `./paper edit server` or `./paper edit api` depending on which project

you want to edit;

- It should show something like

[this](https://gist.github.com/zachbr/21e92993cb99f62ffd7905d7b02f3159) in

the text editor you get.

- If your editor does not have a "menu" at the bottom, you're using `vim`.

If you don't know how to use `vim` and don't want to

learn, enter `:q!` and press enter. Before redoing this step, do

`export EDITOR=nano` for an easier editor to use.

1. Replace `pick` with `edit` for the commit/patch you want to modify, and

"save" the changes;

- Only do this for \*\*one\*\* commit at a time.

1. Make the changes you want to make to the patch;

1. Type `./paper edit continue` in the root directory to finish and rebuild

patches;

1. PR your modified patch file(s) back to this repository.

#### Manual method: Stashing

In case you need something more complex or want more control, these step-by-step

instruction do exactly what the above slightly automated system above does.

1. If you have changes you are working on, type `git stash` to store them for

later;

- You can type `git stash pop` to get them back at any point.

1. Type `git rebase -i upstream/upstream`;

- It should show something like

[this](https://gist.github.com/zachbr/21e92993cb99f62ffd7905d7b02f3159) in

the text editor you get.

- If your editor does not have a "menu" at the bottom, you're using `vim`.

If you don't know how to use `vim` and don't want to

learn, enter `:q!` and press enter. Before redoing this step, do

`export EDITOR=nano` for an easier editor to use.

1. Replace `pick` with `edit` for the commit/patch you want to modify, and

"save" the changes;

- Only do this for \*\*one\*\* commit at a time.

1. Make the changes you want to make to the patch;

1. Type `git add .` to add your changes;

1. Type `git commit --amend` to commit;

- \*\*Make sure to add `--amend`\*\* or else a new patch will be created.

- You can also modify the commit message and author here.

1. Type `git rebase --continue` to finish rebasing;

1. Type `./paper rebuild` in the root directory;

- This will modify the appropriate patches based on your commits.

1. PR your modified patch file(s) back to this repository.

### Method 2 - Fixup commits

If you are simply editing a more recent commit or your change is small, simply

making the change at HEAD and then moving the commit after you have tested it

may be easier.

This method has the benefit of being able to compile to test your change without

messing with your HEADs.

#### Manual method

1. Make your change while at HEAD;

1. Make a temporary commit. You don't need to make a message for this;

1. Type `git rebase -i upstream/upstream`, move (cut) your temporary commit and

move it under the line of the patch you wish to modify;

1. Change the `pick` to the appropriate action:

1. `f`/`fixup`: Merge your changes into the patch without touching the

message.

1. `s`/`squash`: Merge your changes into the patch and use your commit message

and subject.

1. Type `./paper rebuild` in the root directory;

- This will modify the appropriate patches based on your commits.

1. PR your modified patch file(s) back to this repository.

#### Automatic method

1. Make your change while at HEAD;

1. Make a fixup commit. `git commit -a --fixup <hashOfPatchToFix>`;

- You can also use `--squash` instead of `--fixup` if you want the commit

message to also be changed.

- You can get the hash by looking at `git log` or `git blame`; your IDE can

assist you too.

- Alternatively, if you only know the name of the patch, you can do

`git commit -a --fixup "Subject of Patch name"`.

1. Rebase with autosquash: `git rebase --autosquash -i upstream/upstream`.

This will automatically move your fixup commit to the right place, and you just

need to "save" the changes.

1. Type `./paper rebuild` in the root directory;

- This will modify the appropriate patches based on your commits.

1. PR your modified patch file(s) back to this repository.

## PR Policy

We'll accept changes that make sense. You should be able to justify their

existence, along with any maintenance costs that come with them. Using

[obfuscation helpers](#obfuscation-helpers) aids in the maintenance costs.

Remember that these changes will affect everyone who runs Paper, not just you

and your server.

While we will fix minor formatting issues, you should stick to the guide below

when making and submitting changes.

## Formatting

All modifications to non-Paper files should be marked.

- Multi-line changes start with `// Paper start` and end with `// Paper end`;

- You can put a comment with an explanation if it isn't obvious, like this:

`// Paper start - reason`.

- The comments should generally be about the reason the change was made, what

it was before, or what the change is.

- Multi-line messages should start with `// Paper start` and use `/\* Multi

line message here \*/` for the message itself.

- One-line changes should have `// Paper` or `// Paper - reason`.

Here's an example of how to mark changes by Paper:

```java

entity.getWorld().dontbeStupid(); // Paper - was beStupid() which is bad

entity.getFriends().forEach(Entity::explode);

entity.a();

entity.b();

// Paper start - use plugin-set spawn

// entity.getWorld().explode(entity.getWorld().getSpawn());

Location spawnLocation = ((CraftWorld)entity.getWorld()).getSpawnLocation();

entity.getWorld().explode(new BlockPosition(spawnLocation.getX(), spawnLocation.getY(), spawnLocation.getZ()));

// Paper end

```

We generally follow usual Java style (aka. Oracle style), or what is programmed

into most IDEs and formatters by default. There are a few notes, however:

- It is fine to go over 80 lines as long as it doesn't hurt readability.

There are exceptions, especially in Spigot-related files

- When in doubt or the code around your change is in a clearly different style,

use the same style as the surrounding code.

## Patch Notes

When submitting patches to Paper, we may ask you to add notes to the patch

header. While we do not require it for all changes, you should add patch notes

when the changes you're making are technical, complex, or require an explanation

of some kind. It is very likely that your patch will remain long after we've all

forgotten about the details of your PR; patch notes will help us maintain it

without having to dig back through GitHub history looking for your PR.

These notes should express the intent of your patch, as well as any pertinent

technical details we should keep in mind long-term. Ultimately, they exist to

make it easier for us to maintain the patch across major version changes.

If you add a message to your commit in the `Paper-Server`/`Paper-API`

directories, the rebuild patches script will handle these patch notes

automatically as part of generating the patch file. If you are not

extremely careful, you should always just `squash` or `amend` a patch (see the

above sections on modifying patches) and rebuild.

Editing messages and patches by hand is possible, but you should patch and

rebuild afterwards to make sure you did it correctly. This is slower than just

modifying the patches properly after a few times, so you will not really gain

anything but headaches from doing it by hand.

Underneath is an example patch header/note:

```patch

From 02abc033533f70ef3165a97bfda3f5c2fa58633a Mon Sep 17 00:00:00 2001

From: Shane Freeder <theboyetronic@gmail.com>

Date: Sun, 15 Oct 2017 00:29:07 +0100

Subject: [PATCH] revert serverside behavior of keepalives

This patch intends to bump up the time that a client has to reply to the

server back to 30 seconds as per pre 1.12.2, which allowed clients

more than enough time to reply potentially allowing them to be less

tempermental due to lag spikes on the network thread, e.g. that caused

by plugins that are interacting with netty.

We also add a system property to allow people to tweak how long the server

will wait for a reply. There is a compromise here between lower and higher

values, lower values will mean that dead connections can be closed sooner,

whereas higher values will make this less sensitive to issues such as spikes

from networking or during connections flood of chunk packets on slower clients,

at the cost of dead connections being kept open for longer.

diff --git a/src/main/java/net/minecraft/server/PlayerConnection.java b/src/main/java/net/minecraft/server/PlayerConnection.java

index a92bf8967..d0ab87d0f 100644

--- a/src/main/java/net/minecraft/server/PlayerConnection.java

+++ b/src/main/java/net/minecraft/server/PlayerConnection.java

```

## Obfuscation Helpers

In an effort to make future updates easier on ourselves, Paper tries to use

obfuscation helpers whenever possible. The purpose of these helpers is to make

the code more readable and maintainable. These helpers should be be made as easy

to inline as possible by the JVM whenever possible.

An obfuscation helper to access an obfuscated item may be as simple as something

like this:

```java

public final int getStuckArrows() { return this.bY(); } // Paper - OBFHELPER

```

Or it may be as complex as forwarding an entire method so that it can be

overridden later:

```java

public boolean be() {

// Paper start - OBFHELPER

return this.pushedByWater();

}

public boolean pushedByWater() {

// Paper end

return true;

}

```

While they may not always be done in exactly the same way, the general goal is

always to improve readability and maintainability. Use your best judgment and do

what fits best in your situation.

## Configuration files

To use a configurable value in your patch, add a new entry in either the

`PaperConfig` or `PaperWorldConfig` classes. Use `PaperConfig` if a value

must remain the same throughout all worlds, or the latter if it can change

between worlds. World-specific configuration options are preferred whenever

possible.

### PaperConfig example

```java

public static boolean saveEmptyScoreboardTeams = false;

private static void saveEmptyScoreboardTeams() {

// This is called automatically!

// The name also doesn't matter.

saveEmptyScoreboardTeams = getBoolean("settings.save-empty-scoreboard-teams", false);

}

```

Notice that the field is always public, but the setter is always private. This

is important to the way the configuration generation system works. To access

this value, reference it as you would any other static value:

```java

if (!PaperConfig.saveEmptyScoreboardTeams) {

```

It is often preferred that you use the fully qualified name for the

configuration class when accessing it, like so:

`com.destroystokyo.paper.PaperConfig.valueHere`.

If this is not done, a developer for Paper might fix that for you before

merging, but it's always nice if you make it a habit where you only need 1-2

lines changed.

### PaperWorldConfig example

```java

public boolean useInhabitedTime = true;

private void useInhabitedTime() {

// This is called automatically!

// The name also doesn't matter.

useInhabitedTime = getBoolean("use-chunk-inhabited-timer", true);

}

```

Again, notice that the field is always public, but the setter is always private.

To access this value, you'll need an instance of the `net.minecraft.World`

object:

```java

return this.world.paperConfig.useInhabitedTime ? this.w : 0;

```

## Frequently Asked Questions

### I can't find the NMS file I need!

By default, Paper (and upstream) only import files we make changes to. If you

would like to make changes to a file that isn't present in `Paper-Server`'s

source directory, you just need to add it to our import script ran during the

patching process.

1. Save (rebuild) any patches you are in the middle of working on! Their

progress will be lost if you do not;

1. Identify the name(s) of the file(s) you want to import.

- A complete list of all possible file names can be found at

`./work/Minecraft/$MCVER/spigot/net/minecraft/server`. You might find

[MiniMappingViewer] very useful to find the file you need.

1. Open the file at `./scripts/importmcdev.sh` and add the name of your file to

the script. Skip to the 2nd last header and follow the instructions there;

1. Re-patch the server `./paper patch`;

1. Edit away!

> ? This change is temporary! \*\*DO NOT COMMIT CHANGES TO THIS FILE!\*\*

> Once you have made your changes to the new file, and rebuilt patches, you may

> undo your changes to `importmcdev.sh`.

Any file modified in a patch file gets automatically imported, so you only need

this temporarily to import it to create the first patch.

To undo your changes to the file, type `git checkout scripts/importmcdev.sh`.

### Where can I learn how to name method/field?

For most cases, it is preferred if you use [yarn], as their license works with

Paper's license. If you can't do that, [MiniMappingViewer] is always around to

provide you with more of a general idea, \*but\* you cannot use the Mojang names

for more than understanding the code.

[yarn] is in general more thorough than Mojang's own mappings, as they include

method arguments as well, whereas Mojang's do not. If you need local variables

to understand the code, you might be more lucky with ModCoderPack.

For more information on the Mojang name licensing issues, check this out:

<https://cpw.github.io/MinecraftMappingData.html>

### My commit doesn't need a build, what do I do?

Well, quite simple: You add `[CI-SKIP]` to the start of your commit subject.

This case most often applies to changes to files like `README.md`, this very

file (`CONTRIBUTING.md`), the `LICENSE.md` file, and so forth.

### Patching and building is \*really\* slow, what can I do?

This only applies if you're running Windows. If you're running a prior Windows

release, either update to Windows 10 or move to macOS/Linux/BSD.

In order to speed up patching process on Windows, it's recommended you get WSL

2. This is available in Windows 10 v2004, build 19041 or higher. (You can check

your version by running `winver` in the run window (Windows key + R)). If you're

out of date, update your system with the

[Windows Update Assistant](https://www.microsoft.com/en-us/software-download/windows10).

To setup WSL 2, follow the information here:

<https://docs.microsoft.com/en-us/windows/wsl/install-win10>

You will most likely want to use the Ubuntu apps. Once it's set up, install the

required tools with `sudo apt-get update && sudo apt-get install $TOOL\_NAMES

-y`. Replace `$TOOL\_NAMES` with the packages found in the

[requirements](#requirements). You can now clone the repository and do

everything like usual.

> ? Do not use the `/mnt/` directory in WSL! Instead, mount the WSL directories

> in Windows like described here:

> <https://www.howtogeek.com/426749/how-to-access-your-linux-wsl-files-in-windows-10/>

### I wrote some API, how do I use it in Paper-Server?

To install the API to your local maven repository, do the following:

- Enter the API directory by running `./paper api`;

- Run `mvn install`.

- If you are working on a patch without much care for whether the tests

pass, you can instead run `mvn install -DskipTests`. Do not PR changes

without running tests first.

- If a test failed, you have to identify the failing tests by scrolling up a

couple lines (i.e. around 50-200). You should find it fairly quickly.

- If you later need to use the Paper-API, you might want to remove the jar

from your local maven repository.

If you use Windows and don't usually build using WSL, you might not need to

do this.

You can now use the API in your plugin to test it before PRing. You will also

need to do this to build the Server with the implemented API.

[MiniMappingViewer]: https://minidigger.github.io/MiniMappingViewer/

[yarn]: https://github.com/FabricMC/yarn