Setup

Initialize Git Repo

```
git init .
```

Initialize Composer Project

```
composer init
```

```
root@c2936624bfa7:/var/www/packages/mpwt/http# composer init
PHP Warning:  Module "oci8" is already loaded in Unknown on line 0


  Welcome to the Composer config generator


This command will guide you through creating your composer.json config.

Package name (<vendor>/<name>) [root/http]: mpwt/http
Description []: MPWT Standard Http Handling
Author [n to skip]: Kay Briyel
Minimum Stability []: dev
Package Type (e.g. library, project, metapackage, composer-plugin) []: library
License []: MIT

Define your dependencies.

Would you like to define your dependencies (require) interactively [yes]? no
Would you like to define your dev dependencies (require-dev) interactively [yes]? no
Add PSR-4 autoload mapping? Maps namespace "MPWT\Http" to the entered relative path. [src/, n to skip]:

{
    "name": "mpwt/http",
    "description": "MPWT Standard Http Handling",
    "type": "library",
    "license": "MIT",
    "autoload": {
        "psr-4": {
            "MPWT\\Http\\": "src/"
        }
    },
    "authors": [
        {
            "name": "Kay Briyel"
        }
    ],
    "minimum-stability": "dev",
    "require": {}
}

Do you confirm generation [yes]?
Generating autoload files
Generated autoload files
PSR-4 autoloading configured. Use "namespace MPWT\Http;" in src/
Include the Composer autoloader with: require 'vendor/autoload.php';
```

Add Codes to /src

Make sure to use correct `namespace` in each file.

```
<?php

namespace MPWT\Http;

class Handler
{

}
```

Loading Library from local development

In `composer.json` of the root laravel project.

```
...
"autoload": {
    "psr-4": {
        "App\\": "app/",
        ...
        "MPWT\\Http\\": "packages/mpwt/http/src/"
    }
},
```

```
composer dump-autoload
```

Versioning

Laravel Versioning

> POE https://poe.com/s/ZjdYV7XSwCjmhWMLY54w

Yes, that's a good approach. If you want to maintain separate versions of your package for Laravel 5 and Laravel 10, having two branches in a single Git repository is a reasonable solution.

Here's how you can set it up:

1. **Create the Git repository**: Create a new Git repository for your package, e.g., `myvendor/mypackage`.
2. **Create the branches**: Within the repository, create two branches:
   - `laravel-5`: This branch will contain the code for the Laravel 5 version of your package.
   - `laravel-10`: This branch will contain the code for the Laravel 10 version of your package.
3. **Develop and maintain the branches**: Develop and maintain the code for each Laravel version in the respective branches. Ensure that the code is compatible with the target Laravel version.
4. **Versioning and releases**: When you're ready to release a new version of your package, create a new tag on the appropriate branch (e.g., `v1.0.0` on the `laravel-5` branch, `v2.0.0` on the `laravel-10` branch).
5. **Composer configuration**: In your project's `composer.json` file, you can specify the required version of your package based on the Laravel version being used:

```
{
    "require": {
        "myvendor/mypackage": "^1.0" // For Laravel 5 projects
    },
    "repositories": [
        {
            "type": "vcs",
            "url": "https://github.com/myvendor/mypackage.git",
            "branch": "laravel-5"
        }
    ]
}
```

```
{
    "require": {
        "myvendor/mypackage": "^2.0" // For Laravel 10 projects
    },
    "repositories": [
        {
            "type": "git",
            "url": "https://github.com/myvendor/mypackage.git",
            "branch": "laravel-10"
        }
    ]
}
```

This approach allows you to maintain separate versions of your package for different Laravel versions, making it easier to manage compatibility and ensure your users can install the correct version for their project.

Remember to update your package's documentation to clearly explain the version requirements for each Laravel version, so your users can easily choose the right package version for their project.

Git versioning

Sure, here's an example of the Git commands to create and push tags for your package versions:

1. **Create a tag for the laravel 5 version**:

```
# Switch to the laravel-5 branch
git checkout laravel-5

# Create a new tag for version 1.2.3
git tag v1.2.3
```

2. **Create a tag for the laravel 10 version**:

```
# Switch to the laravel-10 branch
git checkout laravel-10

# Create a new tag for version 2.0.0
git tag v2.0.0
```

3. **Push the tags to the remote repository**:

```
# Push the tags to the remote repository
git push origin v1.2.3
git push origin v2.0.0
```

Here's a breakdown of the commands:

- `git checkout laravel-5` : Switches to the `laravel-5` branch.
- `git tag v1.2.3` : Creates a new tag named `v1.2.3` on the current branch (in this case, `laravel-5`).
- `git checkout laravel-10` : Switches to the `laravel-10` branch.
- `git tag v2.0.0` : Creates a new tag named `v2.0.0` on the current branch (in this case, `laravel-10`).
- `git push origin v1.2.3` : Pushes the `v1.2.3` tag to the remote repository.
- `git push origin v2.0.0` : Pushes the `v2.0.0` tag to the remote repository.

After running these commands, your tags will be created and pushed to the remote repository. Your users can then specify the appropriate version constraint in their `composer.json` file to install the correct version of your package for their Laravel project.

Remember to update your package's documentation to clearly explain the version requirements and tagging scheme for each Laravel version, so your users can easily choose the right package version for their project.