magmodules

Composer Guide for Magento 2

for

Composer is the dependency manager that powers Magento 2. Understanding how it works makes installing, updating, and troubleshooting extensions much easier. This guide covers everything from basics to advanced usage.

Why Composer?

The Old Way (Magento 1)

In Magento 1, extensions were installed by copying files into the codebase. This caused:

- **Version conflicts** - No way to track which version was installed
- **Update nightmares** - Manual file replacement, hoping nothing breaks
- **Dependency hell** - Extensions conflicting with each other
- **No rollback** - Broken install? Good luck reverting
- **Security risks** - No verification of package integrity

The Composer Way

Composer solves all of this:

- **Version management** - Exact versions tracked in `composer.json` and `composer.lock`
- **Dependency resolution** - Automatically handles package dependencies
- **Easy updates** - One command to update, with conflict detection
- **Rollback capability** - `composer.lock` lets you restore exact previous state
- **Package integrity** - Checksums verify packages haven't been tampered with
- **Reproducible builds** - Same `composer.lock` = identical installation everywhere

Real-World Benefits

Scenario: You need to update an extension that requires a newer version of a library that another extension also uses.

- **Without Composer:** Hours of manual checking, potential conflicts, broken store
- **With Composer:** Run `composer update`, Composer figures it out or tells you exactly what conflicts

Composer Basics

Key Files

composer.json Your project's dependency manifest. Lists:

- Required packages and version constraints
- Repositories (where to find packages)
- Autoload configuration
- Scripts and hooks

composer.lock Snapshot of exact versions installed. This file:

- Ensures everyone gets identical versions
- Should be committed to version control
- Is the source of truth for what's actually installed

vendor/ Where Composer installs packages. Never edit files here - they'll be overwritten.

Essential Commands

```
# Install all dependencies from composer.lock
composer install

# Update packages to latest versions within constraints
composer update

# Add a new package
composer require vendor/package

# Remove a package
composer remove vendor/package

# Show installed packages
composer show

# Clear Composer cache
composer clear-cache
```

Checking Available Versions

See All Versions of a Package

```
composer show vendor/package-name --all
```

Example:

```
composer show magmodules/magento2-channable --all
```

Output shows all available versions:

```
versions : * 1.15.0, 1.14.2, 1.14.1, 1.14.0, 1.13.0, ...
```

The `*` indicates your currently installed version.

See Available Updates

```
# All packages with updates available
composer outdated

# Specific vendor
composer outdated magmodules/*

# Show minor and patch updates only (safe updates)
composer outdated --minor-only
```

Check Version Constraints

See what version would be installed:

```
composer show vendor/package-name --available
```

Why Can't I Install a Specific Version?

```
composer why-not vendor/package-name 2.0.0
```

This shows which packages or constraints are blocking that version.

Example:

```
composer why-not magmodules/magento2-channable 2.0.0
```

Output might show:

```
magento/product-community-edition 2.4.6 requires magmodules/magento2-channable (^1.0)
```

Repository Configuration

View Configured Repositories

```
composer config repositories --list
```

Or check `composer.json` directly:

```
grep -A 20 '"repositories"' composer.json
```

Common Repository Types

Packagist (default) Public repository, automatically enabled:

```
{
    "repositories": {
        "packagist.org": {
            "type": "composer",
            "url": "https://packagist.org"
        }
    }
}
```

Magento Marketplace For Adobe Commerce and marketplace extensions:

```
{
    "repositories": {
        "magento": {
            "type": "composer",
            "url": "https://repo.magento.com/"
        }
    }
}
```

Private Repository (Satis/Packagist) For private packages:

```
{
    "repositories": {
        "private": {
            "type": "composer",
            "url": "https://packages.yourcompany.com"
        }
    }
}
```

VCS Repository Direct from Git:

```
{
    "repositories": {
        "custom-module": {
            "type": "vcs",
            "url": "https://github.com/vendor/package.git"
        }
    }
}
```

Path Repository Local development:

```
{
    "repositories": {
        "local-module": {
            "type": "path",
            "url": "../my-local-module"
        }
    }
}
```

Add a Repository via CLI

```
composer config repositories.repo-name composer https://packages.example.com
```

Check Repository Authentication

```
composer config --global --list | grep http-basic
```

Authentication is stored in `auth.json` (global or project level).

Version Constraints Explained

Understanding version constraints prevents unexpected updates.

Exact Version

```
"vendor/package": "1.2.3"
```

Only version 1.2.3, nothing else.

Wildcard

```
"vendor/package": "1.2.*"
```

Any version starting with 1.2 (1.2.0, 1.2.1, 1.2.99).

Range

```
"vendor/package": ">=1.2.0 <2.0.0"
```

Version 1.2.0 or higher, but below 2.0.0.

Tilde (~) - Next Significant Release

```
"vendor/package": "~1.2.3"
```

Equivalent to `>=1.2.3 <1.3.0`. Allows patch updates only.

```
"vendor/package": "~1.2"
```

Equivalent to `>=1.2.0 <2.0.0`. Allows minor and patch updates.

Caret (^) - Semver Compatible

```
"vendor/package": "^1.2.3"
```

Equivalent to `>=1.2.3 <2.0.0`. Allows any non-breaking updates (assuming semver).

This is the most common and recommended constraint.

Stability Flags

```
"vendor/package": "1.0.0@beta"
"vendor/package": "dev-main"
```

Use for pre-release or development versions.

Installing Extensions

Magmodules Repository

When you purchase an extension from Magmodules, you get access to our private Composer repository. This is why Composer matters for our customers:

- **Instant access** - Extensions are available immediately after purchase
- **Easy updates** - Run `composer update` to get the latest version
- **License validation** - Your domain is authorized automatically
- **All purchases in one place** - Single repository for all your Magmodules extensions

Setting up access:

1. After purchase, you receive repository credentials via email
2. Add the Magmodules repository to your `composer.json`:

```
{
    "repositories": {
        "magmodules": {
            "type": "composer",
            "url": "https://packages.magmodules.eu"
        }
    }
}
```

Or via CLI:

```
composer config repositories.magmodules composer https://packages.magmodules.eu
```

3. Add authentication to `auth.json`:

```
{
    "http-basic": {
        "packages.magmodules.eu": {
            "username": "fake-7f3a9c2e8b1d4f6a0e5c8b2d9f4a7c1e",
            "password": "fake-2d8f4b6a1c9e3d7f5b0a8c4e6d2f9a3b"
        }
    }
}
```

4. Install the extension:

```
composer require magmodules/magento2-channable
bin/magento setup:upgrade
bin/magento setup:di:compile
bin/magento setup:static-content:deploy
```

```
bin/magento cache:flush
```

Checking your available packages:

Once authenticated, see all packages you have access to:

```
composer search magmodules
```

Why not just download a ZIP?

We use Composer exclusively because:

- **Version tracking** - Always know what version you're running
- **Dependency management** - Extensions may require specific library versions
- **Safe updates** - Composer checks compatibility before updating
- **Consistent deploys** - Same version on staging and production via `composer.lock`
- **No manual file management** - No risk of incomplete uploads or permission issues

From Packagist (Public)

Some Magmodules extensions are also available on the public Packagist repository:

```
composer require magmodules/magento2-googleshopping
bin/magento setup:upgrade
bin/magento setup:di:compile
bin/magento setup:static-content:deploy
bin/magento cache:flush
```

From Magento Marketplace

First, set up authentication:

1. Get keys from marketplace.magento.com → My Profile → Access Keys
2. Create or edit `auth.json` in Magento root:

```
{
    "http-basic": {
        "repo.magento.com": {
            "username": "fake-7f3a9c2e8b1d4f6a0e5c8b2d9f4a7c1e",
            "password": "fake-2d8f4b6a1c9e3d7f5b0a8c4e6d2f9a3b"
        }
    }
}
```

Or when prompted during install, enter public key as username, private key as password.

From Private Repository

```
# Add repository first
composer config repositories.private composer https://packages.example.com

# Then require the package
composer require vendor/private-package
```

Specific Version

```
composer require vendor/package:1.2.3
```

Development Version

```
composer require vendor/package:dev-main
```

Troubleshooting

Memory Errors

```
php -d memory_limit=-1 /usr/local/bin/composer update
```

Or set in `php.ini`:

```
memory_limit = 4G
```

Slow Downloads

Use parallel downloads:

```
composer config --global repo.packagist.org composer https://packagist.org
composer global require hirak/prestissimo
```

Or with Composer 2 (built-in parallel downloads):

```
composer self-update --2
```

Lock File Out of Sync

```
Warning: The lock file is not up to date with the latest changes in composer.json
```

Fix:

```
composer update --lock
```

Class Not Found After Install

Regenerate autoloader:

```
composer dump-autoload
bin/magento setup:di:compile
```

Package Not Found

Check if repository is configured:

```
composer config repositories --list
```

Check package name spelling:

```
composer search package-name
```

Dependency Conflicts

See what's causing the conflict:

```
composer why vendor/conflicting-package
composer why-not vendor/desired-package 2.0.0
```

Clear Everything and Start Fresh

```
rm -rf vendor/ generated/
composer clear-cache
composer install
```

Best Practices

Always Commit composer.lock

This ensures all environments have identical packages:

```
git add composer.json composer.lock
git commit -m "Add new extension"
```

Use composer install in Production

```
# Development: allows updates
composer update

# Production: uses exact versions from lock file
composer install --no-dev --optimize-autoloader
```

Review Before Updating

```
# See what would change
composer update --dry-run

# Then actually update
composer update
```

Keep Composer Updated

```
composer self-update
```

Don't Edit vendor/ Files

Changes will be lost on next `composer install` or `update`. Instead:

- Create patches (using `cweagans/composer-patches`)
- Override classes via Magento's preference/plugin system
- Fork the package if extensive changes needed

Use Exact Versions for Critical Packages

For packages where unexpected updates could break things:

```
"vendor/critical-package": "1.2.3"
```

Document Private Repositories

In your project's README:

```
## Required Repositories

This project requires access to:
- repo.magento.com (Magento Marketplace credentials)
- packages.ourcompany.com (internal packages)

Contact DevOps for authentication credentials.
```

Useful Aliases

Add to your `.bashrc` or `.zshrc`:

```
alias ci="composer install"
alias cu="composer update"
alias cr="composer require"
alias cs="composer show"
alias co="composer outdated"
alias ccc="composer clear-cache"
```

Need More Help?

Documentation:

- All Help Articles - Complete documentation overview

Support:

- Contact Support - Get help from our team