Composer
Best Practices

Nils Adermann
@naderman
Private Packagist
https://packagist.com

Dependency Management

- **Dependency Management** vs **Package Management**
- **System state** vs **installation/update instructions**

- Configuration Management
  - Tool to manage system state (puppet, salt, ansible, chef, …)
  - Description of state (master.pp, top.sls, …)

Working on “Libraries”

Publishing Packages

- README.md
  - What is it?
  - How do I use it?
  - How do I contribute? (Code of Conduct)
- Pick a License
  - SPDX
  - MIT, BSD, GPL
  - “proprietary”

Publishing Packages

- CHANGELOG.md
  - BC breaks
    If necessary create UPGRADE.md
  - changes
  - bugfixes
  - new features

Versioning Libraries

Semantic Versioning

x.y.z
(BC-break).(new functionality).(bug fix)

http://semver.org/

Semantic Versioning

Promise of Compatibility

X.Y.Z

- Must be used consistently

  Dare to increment **X**!
- Only valuable if BC/Compatibility promise formalized
  - See http://symfony.com/doc/current/contributing/code/bc.html
  - Document in Changelog

Continuous Integration for Libraries

- Multiple runs
  - **`composer install`** from lock file
  - **`composer update`** for latest deps
  - **`composer update --prefer-lowest --prefer-stable`** for oldest (stable) deps

  - Potentially multiple composer.json files with different platform configurations
    - `COMPOSER=composer-customer1.json php composer.phar update`
    - `COMPOSER=composer-customer1.json php composer.phar install`
    - Don’t use this except for testing

Working on “Applications”

Simple Versioning

- There are no other packages depending on yours
  BC - for Composer consumption - doesn’t matter
- Options:
  - Don’t use versions at all, rely on your VCS
  - Increment a single integer
  - Use semver if you ship the application

How to update?

- “composer update”
    - no isolation of problems unless run very frequently

- “composer update <package...>”
    - explicit conscious updates

- “composer update --dry-run [<package...>]”
    - Understanding and preparing effects of updates
    - Read CHANGELOGs
    - `composer outdated`

Versions Constraints

Exact Match:	1.0.0	1.2.3-beta2	dev-master
Wildcard Range:	1.0.*	2.*	
Hyphen Range:	1.0-2.0 >=1.0.0 <2.1	1.0.0 - 2.1.0 >=1.0.0 <=2.1.0	
(Unbounded Range: Bad!	*>= 1.0)*		
Next Significant Release	~1.2 >=1.2.0 <2.0.0	~1.2.3 >=1.2.3 <1.3.0	
Caret/Semver Operator	^1.2 >=1.2.0 <2.0.0	^1.2.3 >=1.2.3 <2.0.0	**Best Choice for Libraries**

Operatoren: “ “ AND, “||” OR

Stabilities

- **Order** dev -> alpha -> beta -> RC -> stable
- **Automatically from tags**

1.2.3	-> stable
1.3.0-beta3	-> beta

- **Automatically from branches**

Branch	-> Version (Stability)
2.0	-> 2.0.x-dev (dev)
master	-> dev-master (dev)
myfeature	-> dev-myfeature (dev)

- **Choosing**

```
“foo/bar”: “1.3.*@beta”
“foo/bar”: “2.0.x-dev”

“minimum-stability”: “alpha”
```

PRIVATE PACKAGIST

Nils Adermann
@naderman

In case of Errors

```
$ php composer.phar validate

./composer.json is valid for simple usage with composer but has
strict errors that make it unable to be published as a package:
See https://getcomposer.org/doc/04-schema.md for details on the schema
name : The property name is required
description : The property description is required
require.composer/composer : unbound version constraints (dev-master) should be avoided
```

Common: Version entry in composer.json conflicts with tag

```
$ php composer.phar self-update

$ php composer.phar update -vvv
```

Resolution Conflicts: Overly Strict Requirements

```
// composer.json

    "require": {
        "cool/alice": "~1.3",
        "lazy/bob": "~1.2"
    }

// dependencies

    "name": "cool/alice",
    "require": {
        "monolog/monolog": "~1.6"
    }

    "name": "lazy/bob",
    "require": {
        "monolog/monolog": "1.3.*"
    }
```

Resolution Conflicts: Overly Strict Requirements

```
Your requirements could not be resolved to an installable set of packages.

  Problem 1
    - Installation request for lazy/bob ~1.2 -> satisfiable by lazy/bob[1.4.0].
    - Installation request for cool/alice ~1.3 -> satisfiable by cool/alice[1.3.0].
    - lazy/bob 1.4.0 requires monolog/monolog 1.3.* -> satisfiable by monolog/monolog[1.3.0, 1.3.1]
    - cool/alice 1.3.0 requires monolog/monolog ~1.6 -> satisfiable by monolog/monolog[1.6.0, 1.7.0
    - Can only install one of: monolog/monolog[1.6.0, 1.3.0].
    - Can only install one of: monolog/monolog[1.6.0, 1.3.1].
    - Conclusion: don't install monolog/monolog 1.3.1
    - Conclusion: don't install monolog/monolog 1.7.0
    - Conclusion: don't install monolog/monolog 1.3.0
    - Conclusion: don't install monolog/monolog 1.6.0
```

Resolution Conflicts: Overly Strict Requirements

```
// composer.json

    "require": {
        "cool/alice": "~1.3",
        "lazy/bob": "~1.2"
    }

// dependencies

    "name": "cool/alice",
    "require": {
        "monolog/monolog": "~1.6"
    }

    "name": "lazy/bob",
    "require": {
        "monolog/monolog": "1.3.*"
    }
```

Resolution Conflicts: Stabilities

```
// composer.json

    "minimum-stability": "beta",
    "require": {
        "monolog/monolog": "1.*",
        "symfony/symfony": "~2.4",
        "bad/package": "dev-master"
    }

// dependencies

    "name": "bad/package",
    "require": {
        "monolog/monolog": "dev-master",
    }
```

Resolution Conflicts: Stabilities

```
Your requirements could not be resolved to an installable set of packages.

  Problem 1
    - Installation request for bad/package dev-master -> satisfiable by bad/package[dev-master].
    - bad/package dev-master requires monolog/monolog dev-master -> no matching package found.
```

Resolution Conflicts: Stabilities

```
// composer.json

    "minimum-stability": "beta",
    "require": {
        "monolog/monolog": "1.*",
        "symfony/symfony": "~2.4",
        "bad/package": "dev-master"
    }

// dependencies

    "name": "bad/package",
    "require": {
        "monolog/monolog": "dev-master",
    }
```

Resolution Conflicts: Stabilities

```
// composer.json

    "minimum-stability": "beta",
    "require": {
        "monolog/monolog": "1.*@dev",
        "symfony/symfony": "~2.4",
        "bad/package": "dev-master"
    }

// dependencies

    "name": "bad/package",
    "require": {
        "monolog/monolog": "dev-master",
    }
```

Resolution Conflicts: Stabilities

```
// monolog

    "name": "monolog/monolog",
    "extra": {
        "branch-alias": {
            "dev-master": "1.12.x-dev"
        }
    }
```

```
- Installing monolog/monolog (dev-master 5ad421d)
  Cloning 5ad421d6a1d5d7066a45b617e5164d309c4e2852
```

Resolution Conflicts: Stabilities

```
// monolog

    "name": "monolog/monolog",
    "extra": {
        "branch-alias": {
            "dev-master": "2.0.x-dev"
        }
    }
```

Resolution Conflicts: Stabilities

```
Your requirements could not be resolved to an installable set of packages.

  Problem 1
    - Installation request for monolog/monolog 1.*@dev -> satisfiable by
monolog/monolog[1.12.0].
    - Installation request for bad/package dev-master -> satisfiable by bad/package[dev-master].
    - bad/package dev-master requires monolog/monolog dev-master -> satisfiable by
monolog/monolog[dev-master].
    - Can only install one of: monolog/monolog[1.12.0, dev-master].
```

We require “2.*@dev” instead

- Resolution works
- Project is probably broken:
  bad/package may not be compatible with 2.*

No error but unexpected result?

- **`composer why [--tree] foo/bar`**
  ```
  mydep/here 1.2.3 requires foo/bar (^1.0.3)
  ```
- **`composer why-not [--tree] foo/bar ^1.2`**
  ```
  foo/bar 1.2.3 requires php (>=7.1.0 but 5.6.3 is installed)
  ```

Monorepo

- **repo/projectA/composer.json**

```
"repositories": [
    {"type": "path", "url": "../core"}
],
"require": {
    "vendor/projectB": "dev-master"
}
```

- **repo/projectB/composer.json**

```
"name": "vendor/projectB",
"version": "dev-master"
```

How do partial updates work?

```
{   “name”: “zebra/zebra”,
    “require”: {
        “horse/horse”: “^1.0” }}

{   “name”: “giraffe/giraffe”,
    “require”: {
        “duck/duck”: “^1.0” }}
```

How do partial updates work?

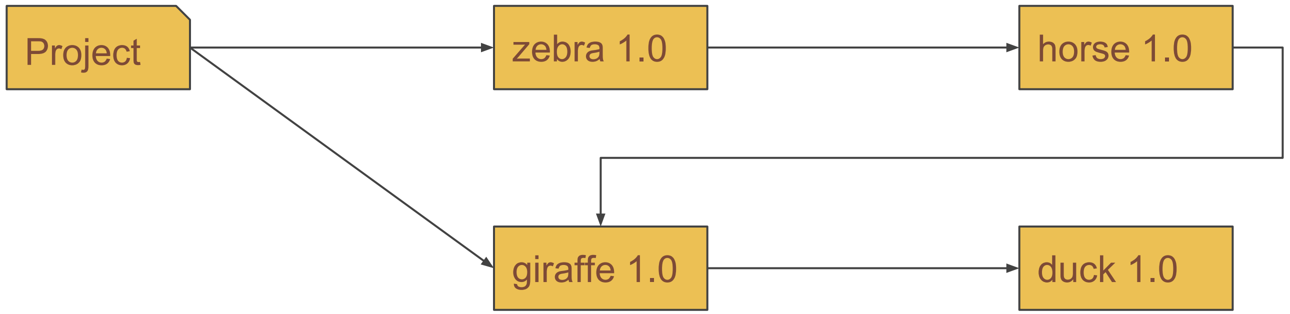
```
{   “name”: “horse/horse”,
    “require”: {
        “giraffe/giraffe”: “^1.0” }}

{   “name”: “duck/duck”,
    “require”: {}}
```

How do partial updates work?

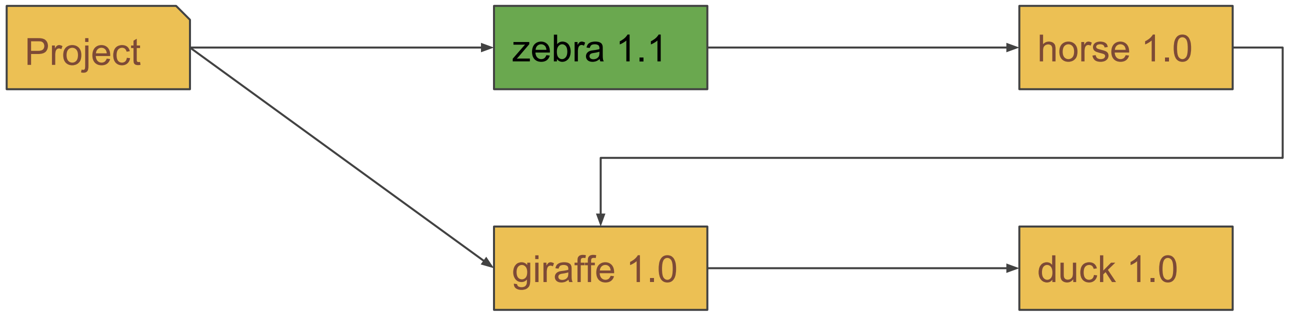
```
{
    “name”: “my-project”,
    “require”: {
        “zebra/zebra”: “^1.0”,
        “giraffe/giraffe”: “^1.0”
    }
}
```

How do partial updates work?



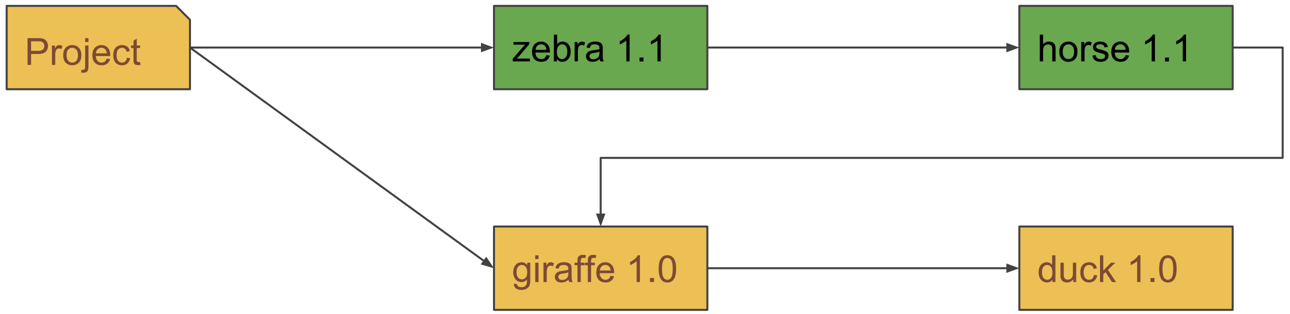
```
Now each package releases 1.1
```

How do partial updates work?



```
$ composer update --dry-run zebra/zebra
    Updating zebra/zebra (1.0 -> 1.1)
```

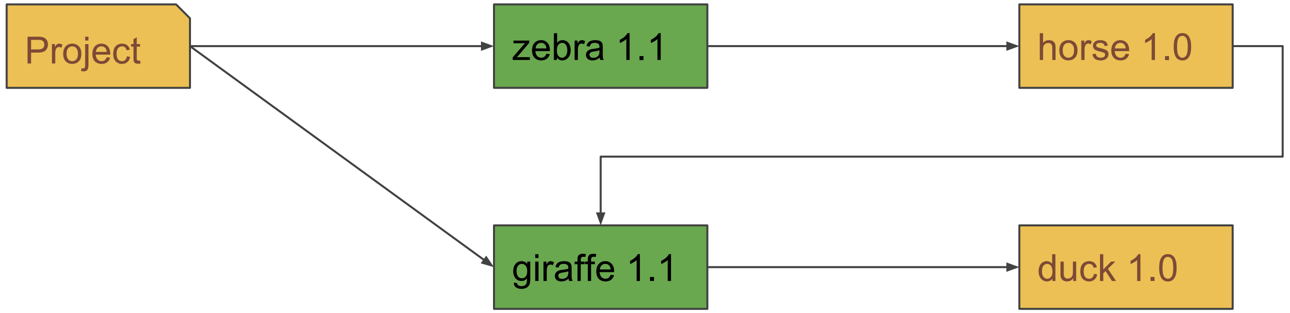
How do partial updates work?



```
$ composer update --dry-run zebra/zebra --with-dependencies
    Updating horse/horse (1.0 -> 1.1)
    Updating zebra/zebra (1.0 -> 1.1)
```

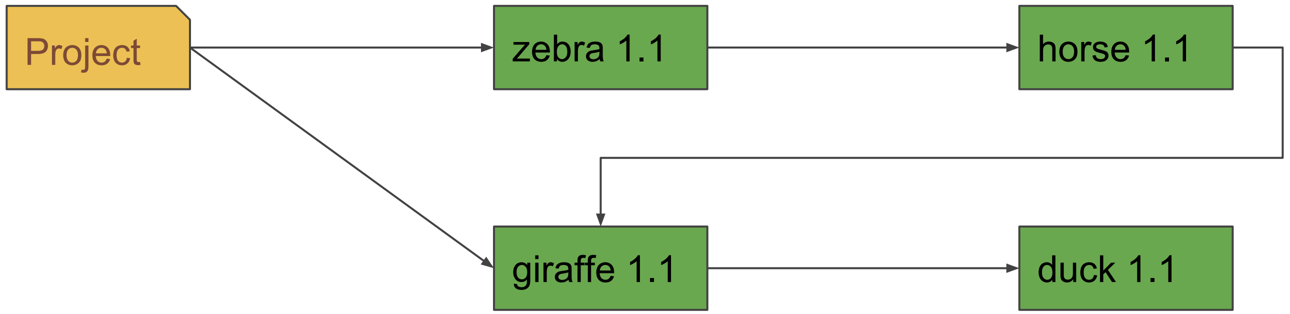
Nils Adermann
@naderman

How do partial updates work?



```
$ composer update --dry-run zebra/zebra giraffe/giraffe
   Updating zebra/zebra (1.0 -> 1.1)
   Updating giraffe/giraffe (1.0 -> 1.1)
```

How do partial updates work?



```
$ composer update zebra/zebra giraffe/giraffe --with-dependencies
    Updating duck/duck (1.0 -> 1.1)
    Updating giraffe/giraffe (1.0 -> 1.1)
    Updating horse/horse (1.0 -> 1.1)
    Updating zebra/zebra (1.0 -> 1.1)
```

The Lock File

- Contents
    - all dependencies including transitive dependencies
    - Exact version for every package
    - download URLs (source, dist, mirrors)
    - Hashes of files
- Purpose
    - Reproducibility across teams, users and servers
    - Isolation of bug reports to code vs. potential dependency breaks
    - Transparency through explicit updating process

Commit The Lock File

- If you don’t
  - composer install without a lock file is a composer update
  - Conflict can randomly occur on install
  - You may not get the same code
  - You no longer manage change
    Change is managing you!

- The lock file exists to be commited!

Nils Adermann
@naderman

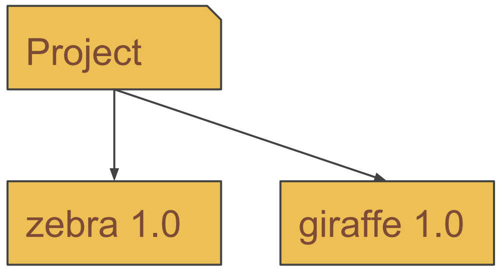
How to resolve lock merge conflicts?

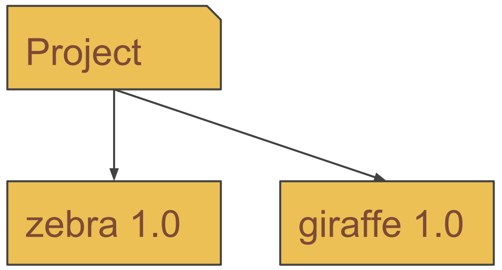
- composer.lock cannot be merged without conflicts
  - contains hash over relevant composer.json values

- `git checkout <refspec> -- composer.lock`
  - `git checkout master -- composer.lock`

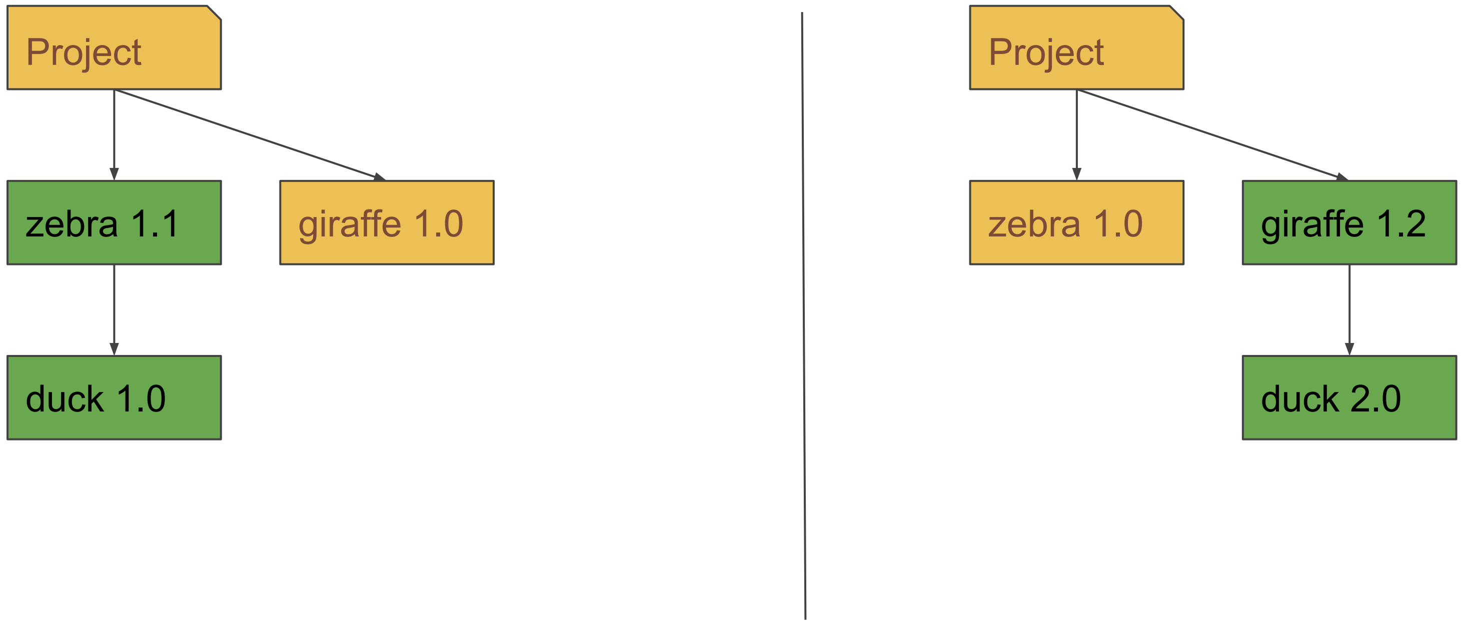
- Repeat: `composer update <list of deps>`
  - Store parameters in commit message
  - Separate commit for the lock file update

How to resolve lock merge conflicts?

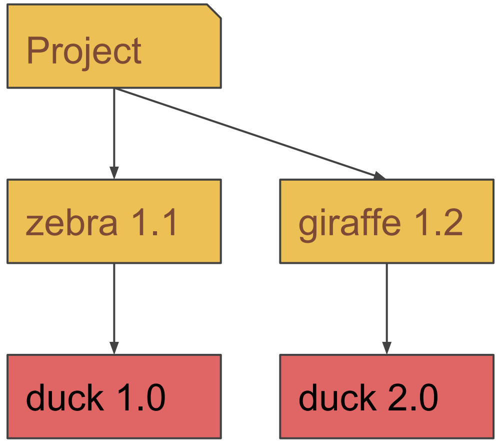




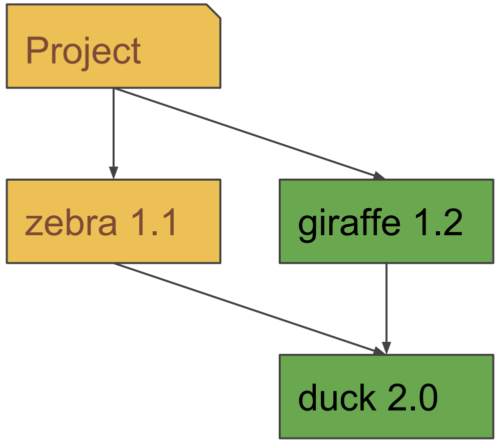
How to resolve lock merge conflicts?



How to resolve lock merge conflicts?



Merge results in invalid dependencies



Rerunning update is safe

PRIVATE PACKAGIST

Autoloader Optimization

- `composer install` **`--optimize-autoloader`**
  - `composer dump-autoload --optimize`
- `composer install --optimize-autoloader` **`--classmap-authoritative`**
  - `composer dump-autoload --optimize --classmap-authoritative`
- `composer install --optimize-autoloader` **`--apcu-autoloader`**
  - `composer dump-autoload --optimize --apcu`

https://getcomposer.org/doc/articles/autoloader-optimization.md

Nils Adermann
@naderman

Platform Configuration

composer update --ignore-platform-reqs

Better:

```
"config": {
    "platform": {
        "php": "5.6.4",
        "ext-mongo": "1.0.0"
    }
}
```

Using Forked Packages

Custom repositories have priority:

```
"repositories": [
    {
        "type": "vcs",
        "url": "https://github.com/naderman/symfony"
    }
],
"require": {
    "symfony/symfony": "dev-master"
}
```

Using Forked Packages

Custom branches are available (composer show -v symfony/symfony)

```
"repositories": [
    {
        "type": "vcs",
        "url": "https://github.com/naderman/symfony"
    }
],
"require": {
    "symfony/symfony": "dev-my-patch"
}
```

Using Forked Packages

Aliases allow other dependencies to resolve against custom branches:

```
"require": {
    "symfony/symfony": "dev-my-patch as 3.1.0"
    "other/package": "1.23"
}

"name": “other/package”
"require": {
    "symfony/symfony": "^3.1"
}
```

Community Tools

- http://packanalyst.com
- http://semver.mwl.be
- http://melody.sensiolabs.org
- http://packagist.graphstory.com
- https://github.com/ziadoz/awesome-php

Summary

- Library CI: composer update --prefer-lowest --prefer-stable
- composer update [--dry-run] <package>
- git checkout <branch> -- composer.lock
- composer dump-autoload --optimize --classmap-authoritative
- composer why/why-not

- Formalize BC promises for users of your libraries
- SemVer: Don't be afraid to increase the major version
- Document changes to dependencies

Commit the composer.lock file!

Thank you!

Questions / Feedback?

E-Mail: n.adermann@packagist.com
Twitter: @naderman