Composer + you

The good, the bad, and the horribly ugly.

@MrDanack

Danack@basereality.com

Part 1 - the good

What problem does Composer solve

- You have a project that depends on a number of libraries.
- Some of those libraries depend on other libraries.
- Loading the library files in PHP is annoying.

Composer finds out which versions of which libraries need to be installed, and installs them

How does it do this

composer.json in the root of your project

```
"name": "basereality/intahwebz",

"require":{
    "aws/aws-sdk-php": "2.4.3",
    "monolog/monolog": "1.4.0",
    "php": ">=5.4.0",
    "zendframework/zend-validator": ">=2.1.5",
    "zendframework/zend-permissions-acl": ">=2.1.0"
},
```

Name - defines current project
Require - the things the current project depends on

Composer update

Finds all the requirements in the root project.

Looks up the required libraries from the listed repositories (Packagist by default). Finds their requirements.

Downloads the packages from the internet, sticks them in 'vendor' folder.

Generates a huge autoloader file.

Generates a 'lock' file that lists the exact versions used.

A repository is just a list of projects by name - we’ll cover them more later.

Composer install

- Reads composer.lock file
- Installs exact same versions per lock file

Lock file needs to be added to VCS

Composer builds PSR0 Autoloader

Each library tells composer where source code exists for each namespace

```
"autoload": {
    "psr-0": {
        "BaseReality": "src/",
        "AnotherLibrary": "lib/"
    }
}
```

Composer compiles all the relevant paths from each of the projects, to build a large autoload_namespaces.php

```
require_once 'vendor/autoload.php';
```

Dependencies can be

PHP version	"php": ">=5.4.0"
PHP extension	"ext-apcu": ">=3.1.9"
C library	"lib-curl": "1.2.3"
Other project	"zendframework/zend-validator": ">=2.1.5"

Version requirements - semver

Exact version: 1.0.2

Range: >=, <, <=, !=   e.g. >=1.0, <2.0, !=1.6

Next Significant Release:

    ~1.2 equivalent to >=1.2,<2.0
    ~1.2.3 is equivalent to >=1.2.3,<1.3

"minimum-stability": "dev"

    dev, alpha, beta, RC, and stable

Version comes from tag

Version requirements - semver

Exclude a single version

```
"require": {
    "twig/twig": "~1.7 , !=1.7.3"
}
```

Exclude a range

```
"require": {
    "monolog/monolog": “<1.0 | >= 1.1”
}
```

Who can tell me what's missing from this?

Requiring branches

Add the branch after a hyphen.

```
{
    "repositories": [
        {
            "type": "vcs",
            "url": "https://github.com/danack/monolog"
        }
    ],
    "require": {
        "monolog/monolog": "dev-bugfix"
    }
}
```

Require specific commit

```
{

"require": {
        "composer/satis": "dev-
master#c7348ef2c84152eae4261236f371d0bc1a28ef7c"
   },
}
```

Probably only useful for debugging.

Requiring branches

Aliasing things to meet requirements

```
{
    "repositories": [
        {
            "type": "vcs",
            "url": "https://github.com/danack/monolog"
        }
    ],
    "require": {
        "monolog/monolog": "dev-bugfix as 2.0.2"
    }
}
```

Require dev

No need to deploy testing libraries to production

```
"require-dev": {
        "athletic/athletic": "~0.1.7",
        "behat/mink": "v1.5.0",
        "behat/mink-goutte-driver": "*",
        "behat/mink-selenium2-driver": "*",
        "phpunit/phpunit": "3.7.*",
        "fzaninotto/faker": "*"
},
```

```
composer install --no-dev
```

Tagging releases

Tag with the semver version.

Version numbers don’t need a V so 1.2.3 not v1.2.3

Tag early - tag often. All numbers can be two digits 1.2.57 is preferable to moving to 1.3.0 before you have to.

You should only be installing from a non-tagged version if you really know what you’re doing and don’t mind inconsistencies between developers.

Top tips - developing in parallel

<u>Tell composer to git clone</u>

--prefer-dist - downloads a zipball over a git-clone
--prefer-source - prefers a git-clone over rather than download zip.

1. composer update
2. composer update symfony/yaml --prefer-source
3. Fix bugs
4. git commit / push from the symfony/yaml vendors directory

(git remote -v to tell which project you’re on)

<u>Or just git clone into vendors directory</u>

Composer won't overwrite directory, if it meets requirements

I'VE DONE STUFF I AINT PROUD OF
AND THE STUFF I AM PROUD OF IS DISGUSTING.

```
namespace MBExtra {
    class Functions{
        public static function load(){}
    }
}

namespace {
    function mb_ucwords($string) {
        return mb_convert_case($string, MB_CASE_TITLE);
    }
}

// Can now safely load functions with
// \MBExtra\Functions::load();
```

Top tips

Faster autoloading

```
composer dump-autoload --optimize
composer --no-dev --optimize-autoloader update
```

Even faster autoloading
https://github.com/Danack/LowMemoryClassLoader

Modified files

```
composer status -v
```

Don’t add test classes to generated autoloader

```
$autoloader = require('./vendor/autoload.php');
$autoloader->add('Intahwebz', [realpath('./').'/test/']);
```

Repositories

A repository is how Composer searches for packages.

By default Composer uses Packagist as it’s only repository.

You can list repositories in the root composer.json file for a project.

These repositories do not get ‘merged’ from the composer.json of dependencies.

For professional development you _really_ ought to be running your own repository.

Listing repositories - legacy

```
"repositories": [
	{
		"type": "pear",
		"url": "http://pear2.php.net"
	},
]
```

Really legacy.

Much better just to download the files.

Listing repositories - legacy

```
"repositories": [  {
    "type": "package",
    "package": {
        "name": "smarty/smarty",
        "version": "3.1.7",
        "dist": {
            "url": "http://www.smarty.net/files/Smarty-3.1.7.zip",
            "type": "zip"
        },
        "source": {
            "url": "http://smarty-php.googlecode.com/svn/",
            "type": "svn",
            "reference": "tags/Smarty_3_1_7/distribution/"
        }
    }
} ]
```

Really legacy.
Much better just to download the files.

Listing repositories - zip files

```
"repositories": [
    {
        "type": "artifact",
        "url": "zips/"
     }
]
```

Listing repositories - VCS

```
"repositories": [
        {
            "type": "vcs",
            "url": "https://github.com/Seldaek/monolog"
        },
        {
            "type": "vcs",
            "url": "/documents/projects/github/Auryn"
        }
]
```

Local file ‘urls’ read latest committed Git version

Binaries and scripts

List binaries to get copied to vendor/bin

```
"bin": ["bin/my-script", "bin/my-other-script"]
```

Scripts

```
"scripts": {
    "post-install-cmd": [
        "php artisan clear-compiled",
        "php artisan optimize"
    ]
},
```

Part 2 - the bad and ugly

Composer is a technical tool.

Dependency management isn’t a technical problem - it’s a *process* problem.

One solution isn’t going to fit everyone.

Updating Composer

Remember how I said tag, tag, tag? Guess what doesn’t happen for composer?

"composer self-update"

- Composer is a dependency in your project, need to use the same version on all machines

- Not secure - downloads and runs unsigned code from the internet

Replace keyword

Allows libraries to be split up into components, as well as held in one repository

Confusing as anything

~~Massive massive security hole~~ is fixed?

Massive slow down

Maintained forking kind of doesn't work - due to single composer.json per git repo

Philosophical difference

"First of all, this behavior is not a security issue in Composer. While the behavior is unintuitive, it will not result in malicious code being used if you use Composer correctly.

Most importantly you should only ever run composer update yourself manually on your development machine. You should read its output, verify it installed and updated packages as expected (use --dry-run to check what would happen without installing anything)." - Nils Aderman

http://blog.naderman.de/2014/02/17/replace-conflict-forks-explained/

IMPLIED FACEPALM

When something is so ridiculous that a full and proper facepalm is not even necessary

List o' security issues

1. php -r "readfile('https://getcomposer.org/installer');" | php
2. composer self-update
3. ~~Replace~~ - fixed \o/
4. Happily downloads over HTTP when SSL not available
5. No validation of certificates
6. No restriction on who can register stuff on Packagist
7. Github not as secure as it should be
8. Scripts run as privileged user

Deployment relies on network

Packagist and Github can be down

Packagist seems to be slow sometimes

Programmer time is (probably) the most expensive cost for your company

Running your own Packagist

Why you should to do this

- Not dependent on Packagist and it’s “Gentleman’s agreement”
- Not dependent on other people’s repositories
- Can deploy without needing Github to be up
- Upgrading versions should require some thought
- Less vulnerable to MITM attacks.
- Way faster

Running your own Packagist

Packagist powered by Satis, available on Github or Packagist. All it does is provide a packages.json file to act as a repository

Example implementation http://github.com/Danack/Bastion

Different ways of doing this

- List repos - allow composer to download

- Provide zip artifacts

Composer style download

Define a satis.json

```
"repositories": [
      { "packagist": false  },
      { "type": "vcs", "url": "https://github.com/Danack/Jig"    },
      { "type": "vcs", "url": "https://github.com/Danack/mb_extra"   },
      { "type": "vcs", "url": "https://github.com/Danack/intahwebz-core" },
      { "type": "vcs", "url": "https://github.com/sebastianbergmann/phpunit" },
      {"type": "vcs", "url":"https://github.com/sebastianbergmann/php-file-
iterator" },
       …
       …
]
```

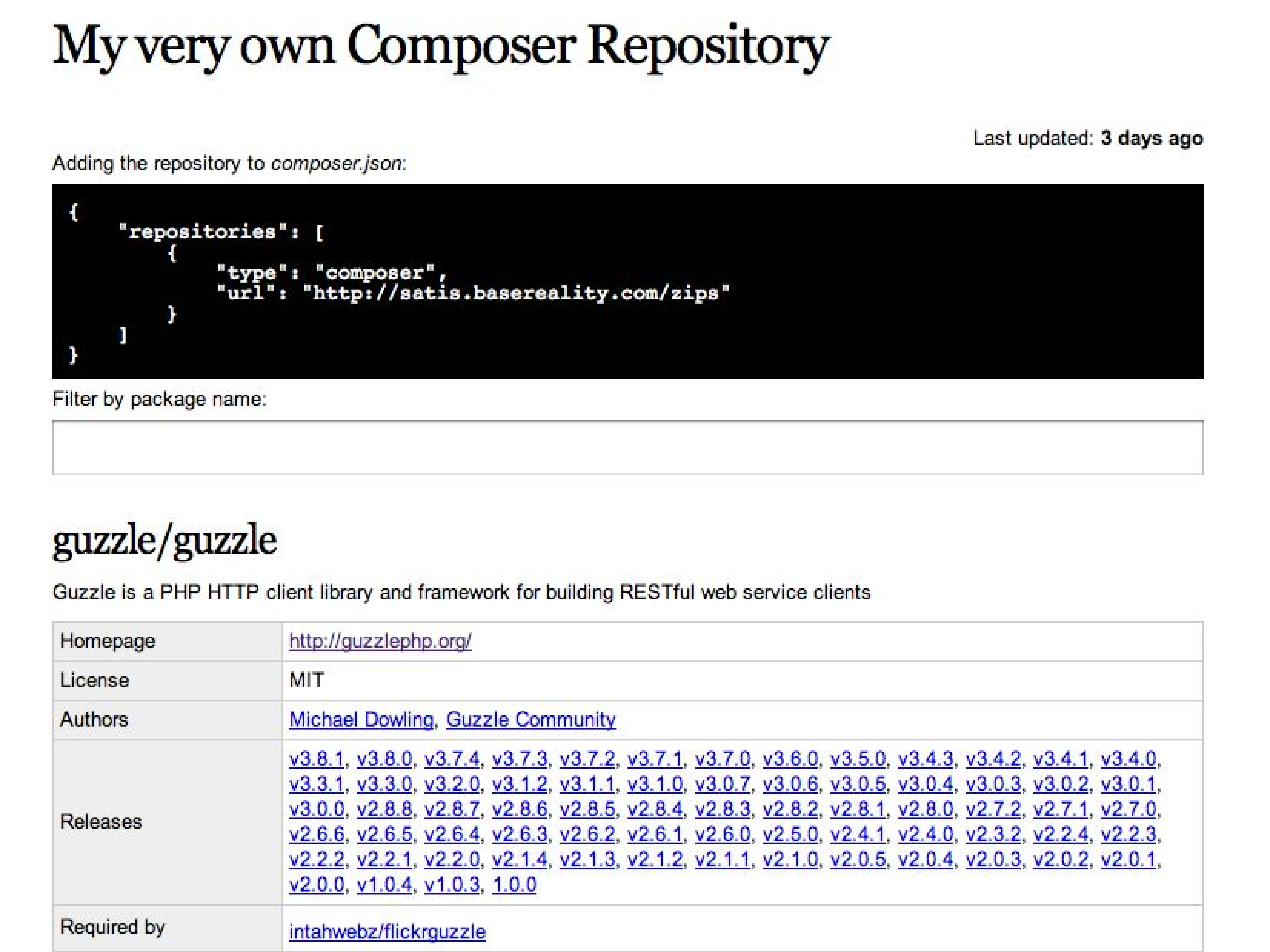
Build it with satis

```
php vendor/bin/satis build satis.json satisOutput -vv
```

Downloads all versions of your require list, and all the dependencies.

Builds a static website

```
php -S localhost:8000 -t satisOutput/
```



My very own Composer Repository

Last updated: **3 days ago**

Adding the repository to *composer.json*:

```
{
    "repositories": [
        {
            "type": "composer",
            "url": "http://satis.basereality.com/zips"
        }
    ]
}
```

Filter by package name:

guzzle/guzzle

Guzzle is a PHP HTTP client library and framework for building RESTful web service clients

Homepage	http://guzzlephp.org/
License	MIT
Authors	Michael Dowling, Guzzle Community
Releases	v3.8.1, v3.8.0, v3.7.4, v3.7.3, v3.7.2, v3.7.1, v3.7.0, v3.6.0, v3.5.0, v3.4.3, v3.4.2, v3.4.1, v3.4.0, v3.3.1, v3.3.0, v3.2.0, v3.1.2, v3.1.1, v3.1.0, v3.0.7, v3.0.6, v3.0.5, v3.0.4, v3.0.3, v3.0.2, v3.0.1, v3.0.0, v2.8.8, v2.8.7, v2.8.6, v2.8.5, v2.8.4, v2.8.3, v2.8.2, v2.8.1, v2.8.0, v2.7.2, v2.7.1, v2.7.0, v2.6.6, v2.6.5, v2.6.4, v2.6.3, v2.6.2, v2.6.1, v2.6.0, v2.5.0, v2.4.1, v2.4.0, v2.3.2, v2.2.4, v2.2.3, v2.2.2, v2.2.1, v2.2.0, v2.1.4, v2.1.3, v2.1.2, v2.1.1, v2.1.0, v2.0.5, v2.0.4, v2.0.3, v2.0.2, v2.0.1, v2.0.0, v1.0.4, v1.0.3, 1.0.0
Required by	intahwebz/flickrguzzle

Running your own Packagist

Now tell your project to use the newly built Satis

```
"repositories": [
        {
            "packagist": false
        },
        {
            "type": "composer",
            "url": "http://localhost:8000/"
        }
}
```

Upload it to an S3 static site

- Create bucket name satis.basereality.com in a region near you

- Add CNAME to point to satis.basereality.com.s3-website-eu-west-1.

  amazonaws.com

- Upload the files

- Add appropriate access rules

There is an S3 plugin for Composer, that uses S3 key/secret but suffers a chicken and egg problem.

Running your own Packagist

Then use the version on S3

```
"repositories": [
        {
            "packagist": false
        },
        {
            "type": "composer",
            "url": "https://satis.basereality.com"
        }
}
```

Convenient yet dangerous!

This is not secure.

Composer doesn’t check SSL certificate for validity. Without signing, you cannot trust DNS.

Hard code IP address in HOSTS file?

Which means you can't use S3 directly - have to use your own server. Which may be a good idea anyone to allow better access control.

Or only expose to internal VPN

Alternatively - satis with zip artifacts

- PHP code downloads zip-files directly from github, modifies them to add version
- Satis scans zip files
- Builds static satis site
- Use either locally or host it on a server

Alternatively - satis with zip artifacts

- Allows better control of which versions are available
- Slightly more secure
- Faster
- Allows non-Ascii chars in filenames !

Alternatively - satis with zip artifacts

Use Bastion to download zip files, then a simple satis file of

```
"repositories": [
        {
            "packagist": false
        },
        {
            "type": "artifact",
            "url": "zips/"
        },
}
```

Summary

- Composer - good

- Packagist - bad

- Satis - hour or two to setup, pays for itself very easily with faster dev, deployment times. Also far more secure, so good.

FIN

getcomposer.org
github.com/composer/satis

Cheat sheet http://composer.json.jolicode.com/

github.com/Danack/Bastion

github.com/Danack/LowMemoryClassLoader

@MrDanack

Danack@basereality.com

Notes

Renaming packages

```
{
    "name": "new/name",
    "replace": {
        "old/name": "*"
    }
}
```

Works from root only?

Avoid test in namespaces

```
require_once('../vendor/autoload.php');

$autoloader->add('SomeProject', [realpath('./').'/test/']);
```

Tools shouldn't be intelligent

Tools shouldn't be intelligent

"If the product is used as a tool, its interface should be as unintelligent as possible. Stupid is predictable; predictable is learnable; learnable is usable.

My guess is that if there is any "next thing" in search interfaces, it will come not from smarter UIs, but from dumber ones in which the user does more work - the Graffiti effect. If a small quantity of user effort can produce a substantial improvement in user experience (which is a big if), the user will accept the bargain. Hey, it made Jeff Hawkins rich."

http://unqualified-reservations.blogspot.co.uk/2009/07/wolfram-alpha-and-hubristic-user.html

Notes

Words words words
"minimum-stability": "stable",

Require dev in depdendecy doesn’t work? Had to add this to root composer.json of project.
    "intahwebz/core": "dev-master",

"minimum-stability": "dev" -> clone all the things

https://github.com/easybiblabs/s3-syncer
http://tech.m6web.fr/composer-installation-without-github.html

MITM attacks:
https://github.com/composer/composer/issues/1074

Accidental hacking
https://github.com/Danack/guzzle/pull/2

Why you shouldn't trust Github

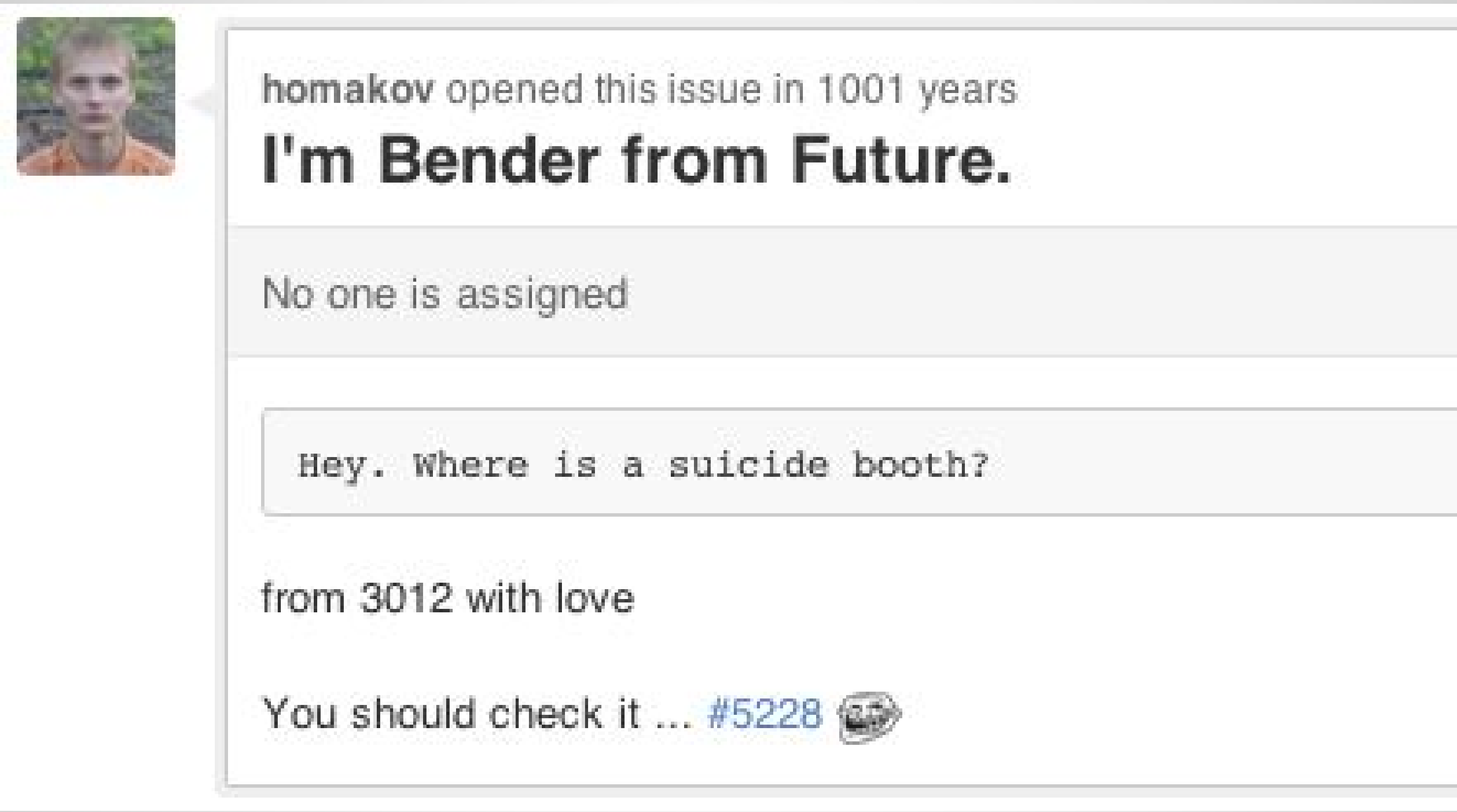
(al [illegible]

How can he commit:

https://github.com/rails/rails/commit/b83965785db1eec019edf1fc272b1aa393e6dc57

Bender from the future:

https://github.com/rails/rails/issues/5239



homakov opened this issue in 1001 years

I'm Bender from Future.

No one is assigned

```
Hey. Where is a suicide booth?
```

from 3012 with love

You should check it ... #5228

JSON is a stupid choice

```
"require": {
    "some/lib": "~1.7, !=1.7.3, !=1.7.5"
}
```

"I removed comments from JSON because I saw people were using them to hold parsing directives, a practice which would have destroyed interoperability. I know that the lack of comments makes some people sad, but it shouldn't.

Suppose you are using JSON to keep configuration files, which you would like to annotate. Go ahead and insert all the comments you like. Then pipe it through JSMin before handing it to your JSON parser."

https://plus.google.com/+DouglasCrockfordEsq/posts/RK8qyGVaGSr

"Commit your composer.lock and use composer install. You should rarely ever use composer update, as the whole point is "get me the latest version of code that is fitting these requirements." That could be a whole range of things and some of it could be dodgy, so my advice goes a step further:

NEVER run "$ composer update" at all. EVER.

Run '$ composer update specific-package' then check the code. Run your tests, see if it works, see if its legit and commit your lock file."

- Phil Sturgeon

"Anyway I deleted the offending fork, @mlebkowski @nediam please take notice and use https://getcomposer.org/doc/05-repositories.md#vcs instead of abusing packagist to fork packages.

If it’s a real fork you intend to maintain you can submit it to packagist but then you should not use replace and you should definitely add a note explaining what your version does differently."

https://github.com/schmittjoh/JMSTranslationBundle/issues/177

Ugly stuff

Renaming packages not supported

Package name always comes from master branch. This sucks as it makes forking projects temporarily be not feasible (but would be difficult to fix).

Packagist not secure

~~‘Replaces’ is a global replace within a Satis repository. Absolutely no security on who can set that.~~

Update your composer.

Not entirely stable

Still in development. If you’ve got your own build system that fits your own need, then maybe only use it for publishing projects, rather than consuming them.

Stupid format

If you're ever writing something by hand, (and so may need to comment it) then choosing a format that doesn't support comments is odd.