Composer 101

Mike Miles | Drupalcon Nashville 2018

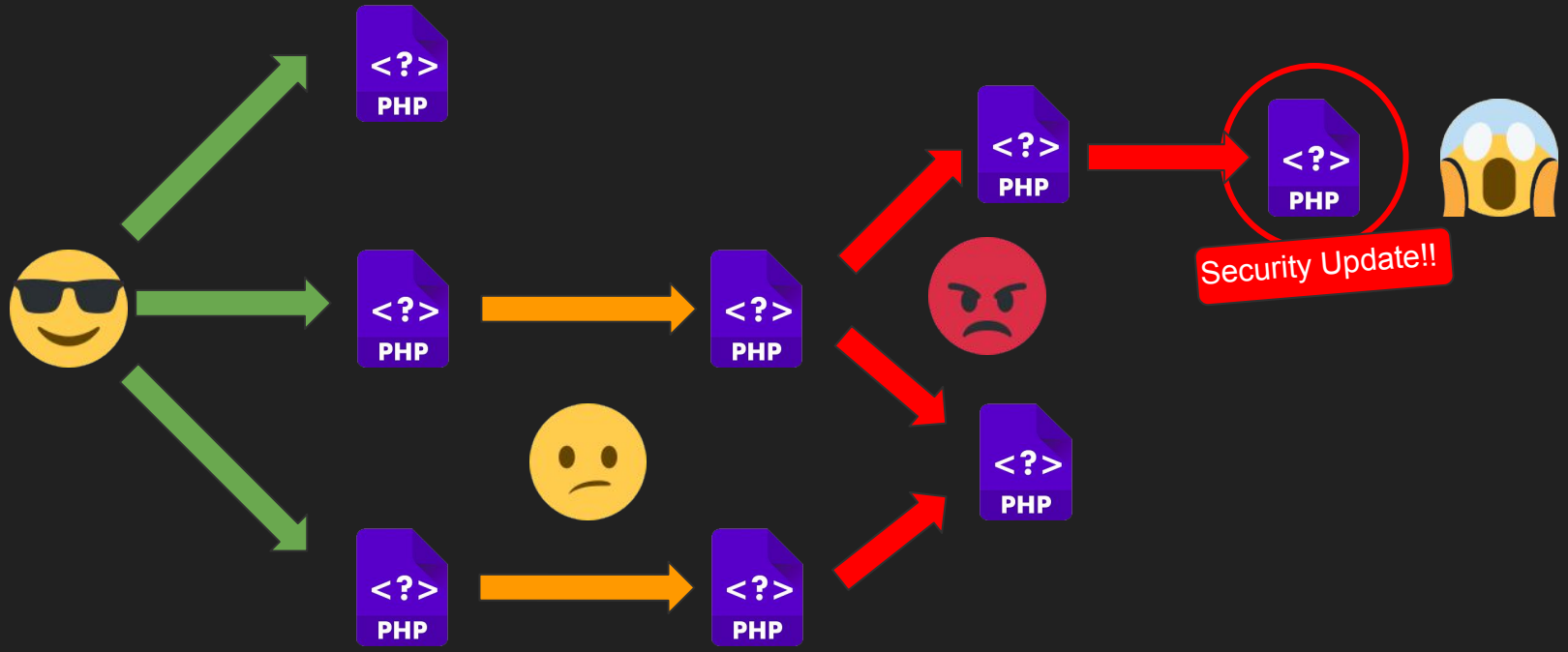
events.drupal.org/node/20624

About Me

Work: Genuine (wearegenuine.com)

Podcast: Developing Up (developingup.com)

Online Handle: mikemiles86 (@mikemiles86)



PHP projects that have a few dependencies may be able simple to maintain. But complex projects with many layers of dependencies, frustrate developers and waste project time on managing those dependencies.

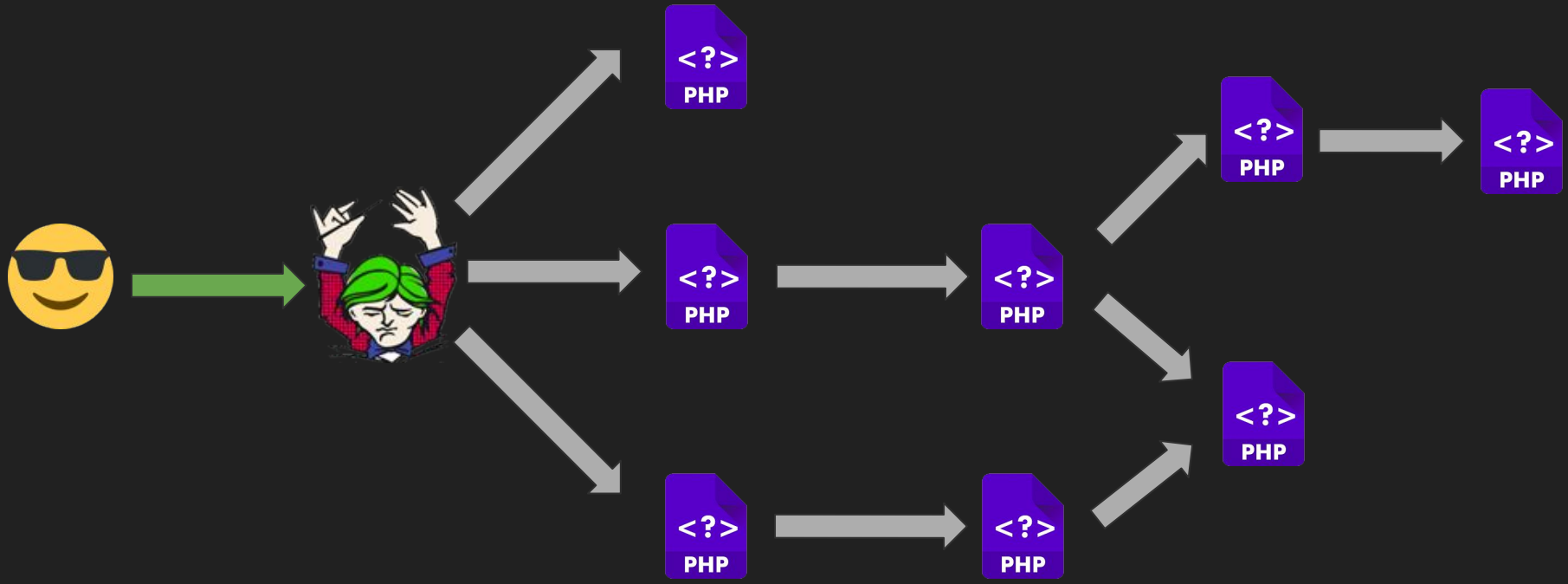
Every project has limited time & budget

The more project time is spent on maintaining 3rd party code, the less time there is available to focus on building what will deliver project value.

Composer

getcomposer.org

Composer is a PHP project dependency manager, that handles 3rd party project code, so that the developers do not have to.



Adding a few files and utilizing a few commands, composer can be added to any PHP project. Composer takes care of 3rd party code dependencies, installation and maintenance.

Composer project structure

```
root/
    [composer.phar]
    composer.json
    composer.lock
    vendor/
    // everything else...
```

Every Composer based project has a composer.json file, composer.lock file, and vendor director. Optionally it can contain the composer executable.

Secure Project Structure

```
root/
    [composer.phar]
    composer.json
    composer.lock
    vendor/
    webroot/
     // everything else...
```

For security purposes, keep all composer related files and directories above the webroot of the project. Access vendor code using the composer autoload.php.

Install

// Installing composer

```
root/
    [composer.phar]
    composer.json
    composer.lock
    vendor/
    // everything else...
```

Installation on Windows

https://getcomposer.org/Composer-Setup.exe

For Windows based systems Composer provides an installation program, which will install Composer globally on the system.

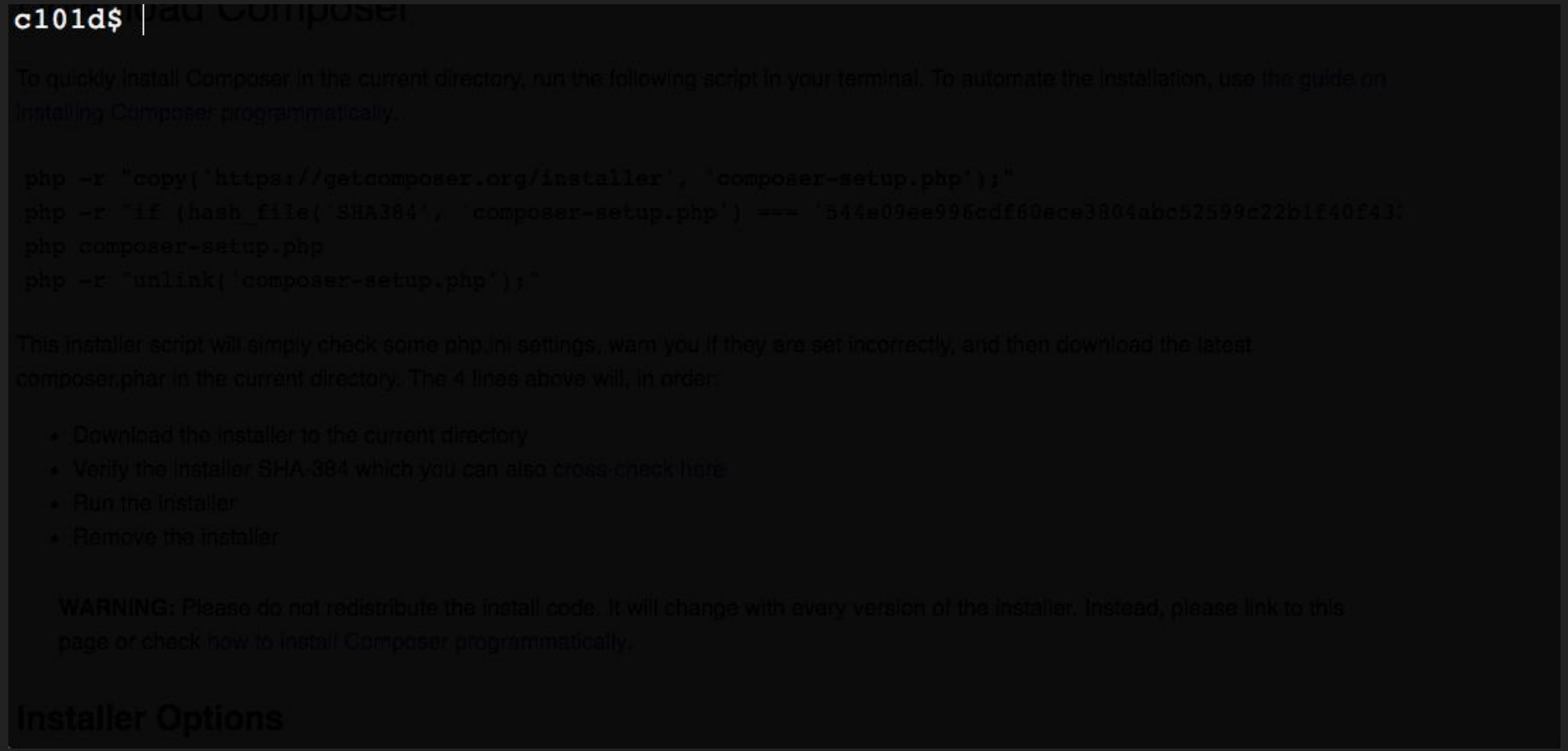
Installation on Linux/Unix/OSX

https://getcomposer.org/download

For Linux/Unix based systems Composer provides instructions for directly downloading, verifying and setting up composer.

Global vs. Per-Project

- Only have to install composer once on your system.
- Can add to PATH to allow using simple command: `composer`
- Every team member is responsible for their own composer install.
- Need to install composer for every project.
- Have to run composer using php command: `php composer`
- Every team member uses same version of composer and is not responsible for install.



Installing composer from command line for a local setup(within a project).
Passing --filename flag to rename file to just `composer`

What just happened?

// command-line install *(local)*

Download and verify composer installer

```
php -r "copy('https://getcomposer.org`,i...
php -r "if(hash_file('SHA348', 'composer...
Installer verified
```

Installs composer.phar into current directory. Flag sets filename

```
php composer-setup.php  --filename=compo...
Composer installed
```

Removes installer

```
php -r "unlink('composer-setup.php');"
```

c101d$ |

To install Composer in the current directory, run the following script in your terminal. To automate the installation, use the guide on installing Composer programmatically.

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('SHA384', 'composer-setup.php') === '544e09ee996cdf60ece3804abc52599c22b1f40f43
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

This installer script will simply check some php.ini settings, warn you if they are set incorrectly, and then download the latest composer.phar in the current directory. The 4 lines above will, in order:

- Download the installer to the current directory
- Verify the installer SHA-384 which you can also cross-check here
- Run the installer
- Remove the installer

WARNING: Please do not redistribute the install code. It will change with every version of the installer. Instead, please link to this page or check how to install Composer programmatically.

Installer Options

--install-dir

Installing composer globally(outside of project), follows same steps as local install. Except, composer file is moved to a directory in your PATH.

What just happened?

// command-line install *(global)*

Download and verify composer installer

```
php -r "copy('https://getcomposer.org`,i...
php -r "if(hash_file('SHA348', 'composer...
Installer verified
```

Installs composer.phar into current directory. Flag sets filename

```
php composer-setup.php  --filename=compo...
Composer installed
```

Moves composer.phar into a PATH accessible directory.

```
mv composer.phar /usr/local/bin/composer
```

Removes installer

```
php -r "unlink('composer-setup.php');"
```

Init

```
% composer init
```

```
root/
    [composer.phar]
    composer.json
    composer.lock
    vendor/
    // everything else...
```

composer.json structure

```
{
  "name": "..."
  "description": "...",
  "type": "project",
  "license": "...",
  "authors": [{...}],
  "minimum-stability": "...",
  "require": {...},
  "require-dev": {...},
}
```

Composer.json is a json schema file that defines project metadata, properties and package dependencies.

The `composer init` command executes an interactive guide for generating a basic composer.json file. Prompting for property values and any initial dependencies. Square brackets [] contain default values.

What just happened?

```
% composer init
```

Prompts for project metadata.

```
Package name (<vendor>/<name>)[user/dir]:
Description []: My demo composer project
Author [<name> <email>]:
Minimum Stability []: ""
Package type (e.g, library, ... ) :project
License[]:
```

Prompts for interactive search for project dependencies.

```
... define Dependencies [yes]?
... define dev-dependencies [yes]?
```

Confirm and generate composer.json file.

```
Confirm generation [yes]?
```

```
{
    "name": "mike.miles/c101d",
    "description": "My demo composer project",
    "type": "project",
    "authors": [
        {
            "name": "Mike Miles",
            "email": "MMiles@wearegenuine.com"
        }
    ],
    "require": {}
}
```

After completing the interactive 'composer init' command the following composer.json file is created. It contains basic project metadata.

Repositories

// package repositories

```
root/
    [composer.phar]
    composer.json
    composer.lock
    vendor/
    // everything else...
```

Packagist *The PHP Package Repository*

Browse Submit Create account Sign in

Search packages...

Packagist is the main Composer repository. It aggregates public PHP packages installable with Composer.

Getting Started

Define Your Dependencies

Put a file named *composer.json* at the root of your project, containing your project dependencies:

```
{
    "require": {
        "vendor/package": "1.3.2",
        "vendor/package2": "1.*",
        "vendor/package3": "^2.0.3"
    }
}
```

For more information about packages versions usage, see the composer documentation.

Install Composer In Your Project

Run this in your command line:

```
curl -sS https://getcomposer.org/installer | php
```

Or download composer.phar into your project root.

Publishing Packages

Define Your Package

Put a file named *composer.json* at the root of your package's repository, containing this information:

```
{
    "name": "your-vendor-name/package-name",
    "description": "A short description of what your package does",
    "require": {
        "php": "^5.3.3 || ^7.0",
        "another-vendor/package": "1.*"
    }
}
```

This is the strictly minimal information you have to give.

For more details about package naming and the fields you can use to document your package better, see the about page.

Commit The File

Add the `composer.json` to your git or other VCS repository and commit it.

Publish It

By default composer will look for packages on packagist.org.

Adding repositories (composer)

```
{
 ...
 "repositories":[
  {
    "type": "composer",
    "url": "https://packages.drupal.org/8"
   },
 ],
 "require": {...},
 ...
}
```

Additional composer package repositories can be added to the composer.json file as an object in the 'repositories' array, with type and url attributes specified.

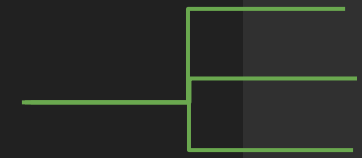
Adding repositories (vcs)

```
{
 ...
 "repositories":[
  {
    "type": "vcs",
    "url": "https://github.com/name/project"
   },
 ],
 "require": {...},
 ...
}
```

Github or other version control repositories can also be added to the 'repositories' array, using type of 'vcs' and providing the url.

require

```
% composer require
```



```
root/
    [composer.phar]
    composer.json
    composer.lock
    vendor/
    // everything else...
```

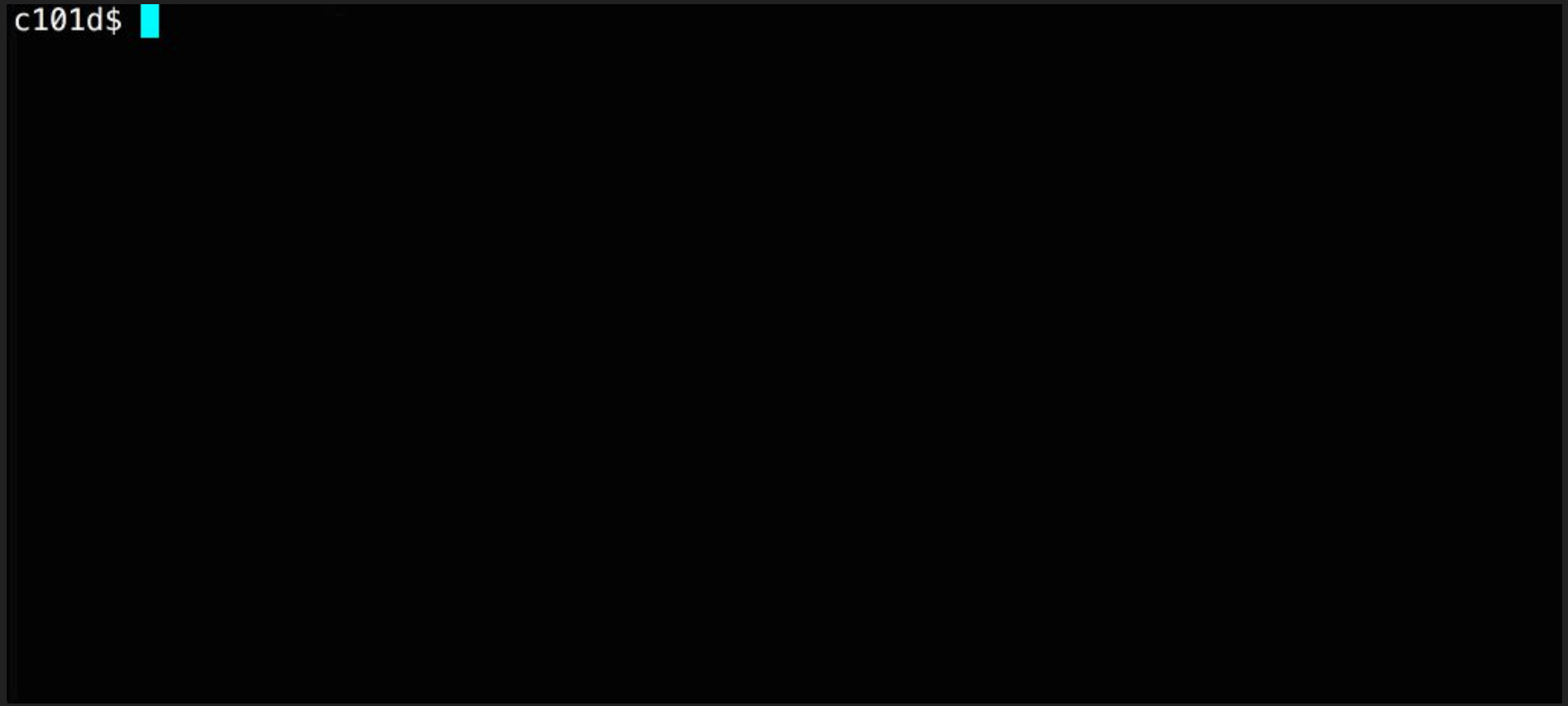
Composer require variations

```
% composer require

% composer require <vendor>/<package>

% composer require <vendor>/<package> <version>
```

The 'require' command has optional values for defining a specific package and version constraints. Without any options, it launches an interactive search.



The composer 'require' command when used without parameters, prompts an interactive search for packages that match a provided term. Displaying a list of matching packages from all known repositories.*

**The --no-suggest flag was passed for clean demo output.*

What just happened?

Prompt for package search keyword.

Searches known repositories and returns list of matching packages.

Prompt to select package and optionally version.

Adds package and version to composer.json

Downloads package and dependencies into vendor and update composer.lock

```
% composer require

Search for package: log

Found 15 packages matching log:
  [0] monolog/monolog
  [1] psr/log
  ...

Enter package # to add... : 0
Enter package version constraint... :

Using version ^1.23 for monolog/monolog
./composer.json has been updated

  - Installing psr/log(1.0.2)
  - Installing monolog/monolog(1.23.0)
Writing lock file
```

composer.lock

```
{
 "packages": [
   {
      "name": "monolog/monolog",
      "version": "1.23.0",
      "source": { ... },
      "require": {
          "php": ">=5.3.0",
          "psr/log": "~1.0"
       },
     ...
```

Composer.lock is a generated JSON schema file that contains a "packages" array with data about all installed project dependencies. Including the exact version installed, repository location and any child dependencies.

DO NOT EDIT THE LOCK FILE

The composer.lock files is generated and maintained by Composer and should never be directly edited.

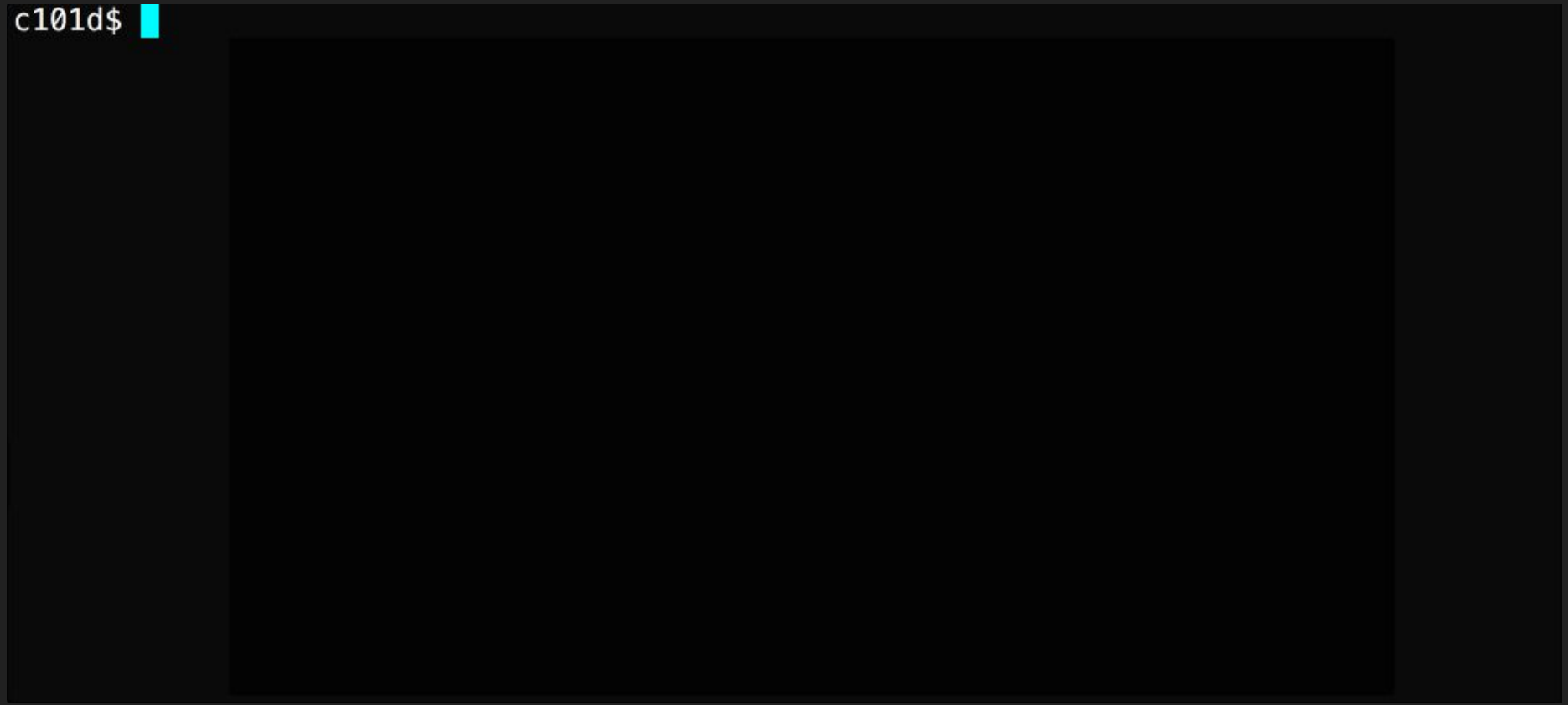
/vendor

```
root/
    ...
    vendor/
        composer/
        monolog/
          monolog/
        psr/
          log/
```

The vendor directory holds the files for all installed 3rd party packages. The directory is organized into vendor directories and then package directories. A vendor can contain many package directories.

DO NOT COMMIT VENDOR

When using composer it is best not to add the vendor directory to your project version control repository, so that you do not have to maintain 3rd party code.



The composer 'require' command when passed a vendor/package value will search for the package across the known repositories. If found it will get the latest package version and install it along with any child dependencies. Updating the composer.json and composer.lock files.

What just happened?

```
% composer require <package>
```

Adds latest version of package to composer.json

```
 Using version 5.7 of phpunit/phpunit
./composer.json has been updated
```

Downloads package and all dependencies into the vendor/ directory.

```
  - Installing symfony/yml(v3.4.6)
  - Installing sebastian/version(2.0.1)
  ...
  - Installing phpunit/phpunit(5.7.27)
```

Provides suggestions of additional packages to install.

```
symfony/yaml suggests installing...
phpunit/phpunit suggests installing...
```

Adds package and dependency information to composer.lock

```
Writing lock file
```

Package version constraints

```
{
 ...
 "require": {
    "vendor/package": "1.0.1",
    "vendor/package": ">=1.0 <2.0",
    "vendor/package": "1.0.*",
    "vendor/package": "~1.2",
    "vendor/package": "2.0@dev"
 }
 ...
}
```

Composer uses constraint strings to figure out which version of a package to add to a project. It supports a range of ways to define package constraints.

The composer 'require' command when passed package and version constraint parameters will find the latest version of a package that meets the constraint criteria. It will then install the package and any child dependencies.

**The --no-suggest flag was passed for clean demo output.*

What just happened?

```
% composer require <package> <version>
```

Adds specified version of package to composer.json

```
./composer.json has been updated
```

Downloads package and all dependencies into the vendor/ directory.

```
  - Installing symfony/polyfill-mbstring...
  - Installing symfony/translation(v3.4.6)
  ...
  - Installing behat/behat(3.3.1)
```

Adds package and dependency information to composer.lock

```
Writing lock file
```

```
{
    "name": "mike.miles/c101d",
    "description": "my demo composer project",
    "type": "project",
    "authors": [...],
    "require": {
       "monolog/monolog": "^1.20.0",
       "phpunit/phpunit": "^5.7",
       "behat/behat": "3.3.*"
    }
}
```

After adding packages using the 'composer require' command, project dependencies are added to the composer.json 'require' array.

install

```
% composer install
```

```
root/
    [composer.phar]
    composer.json
    composer.lock
    vendor/
    // everything else...
```

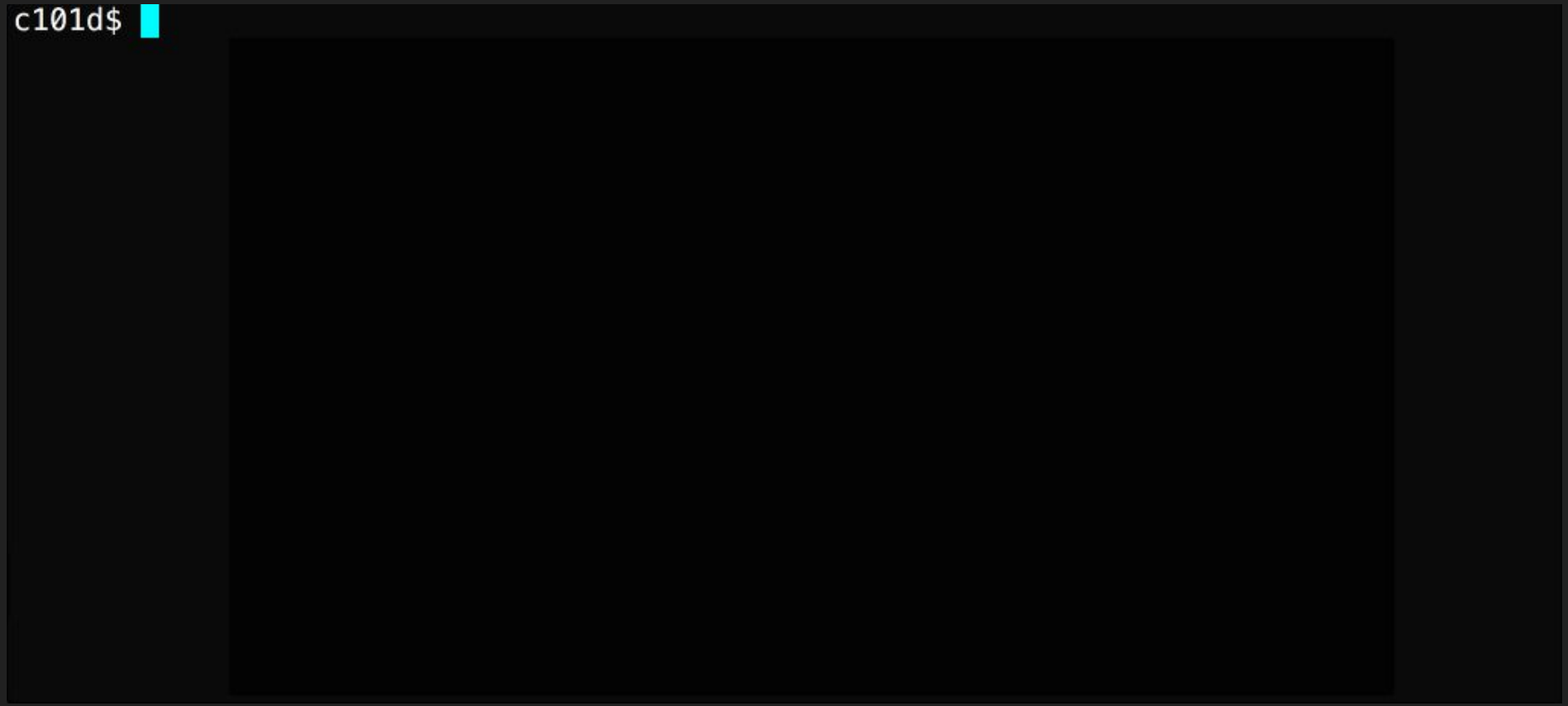
No Lock vs. Lock

- Composer will read requirements from the composer.json file if there is no lock file
- Composer will use version constraints to find matching package versions.
- Can result in different versions per install.

- Composer will read requirements from composer.lock if present.
- Composer will install exact version specified in the lock file.
- Same version of packages will be installed every time.

COMMIT COMPOSER.LOCK TO REPO

It is best practice to add your composer.lock file to your version control repository. Doing so guarantees that every developer (and environment) on the project uses the same version of 3rd party packages.



When the composer `install` command is run, it will install all known project dependencies. It will read dependencies from composer.lock if it exist (installing exact versions), else it will read from composer.json.

**The --no-suggest flag was passed for clean demo output.*

What just happened?

```
% composer install
```

If composer.lock exists, then reads dependency information from there, else from composer.json.

```
Loading composer repositories...
Installing dependencies from lock file
```

Downloads all packages and all dependencies into the vendor directory.

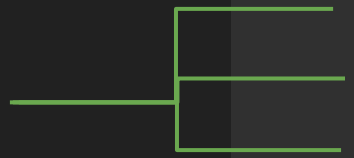
```
  - Installing webmozart/assert(1.3.0)...
  - Installing symfony/polyfill-mbstring...
  - Installing psr/log(1.0.2)...
  ...
  - Installing monolog/monolog(1.23.0)...
  - Installing phpunit/phpunit(5.7.27)...
  - Installing behat/behat(3.3.1)...
```

If composer.lock file is not present then it is created.

```
Writing lock file
```

update

```
% composer update
```



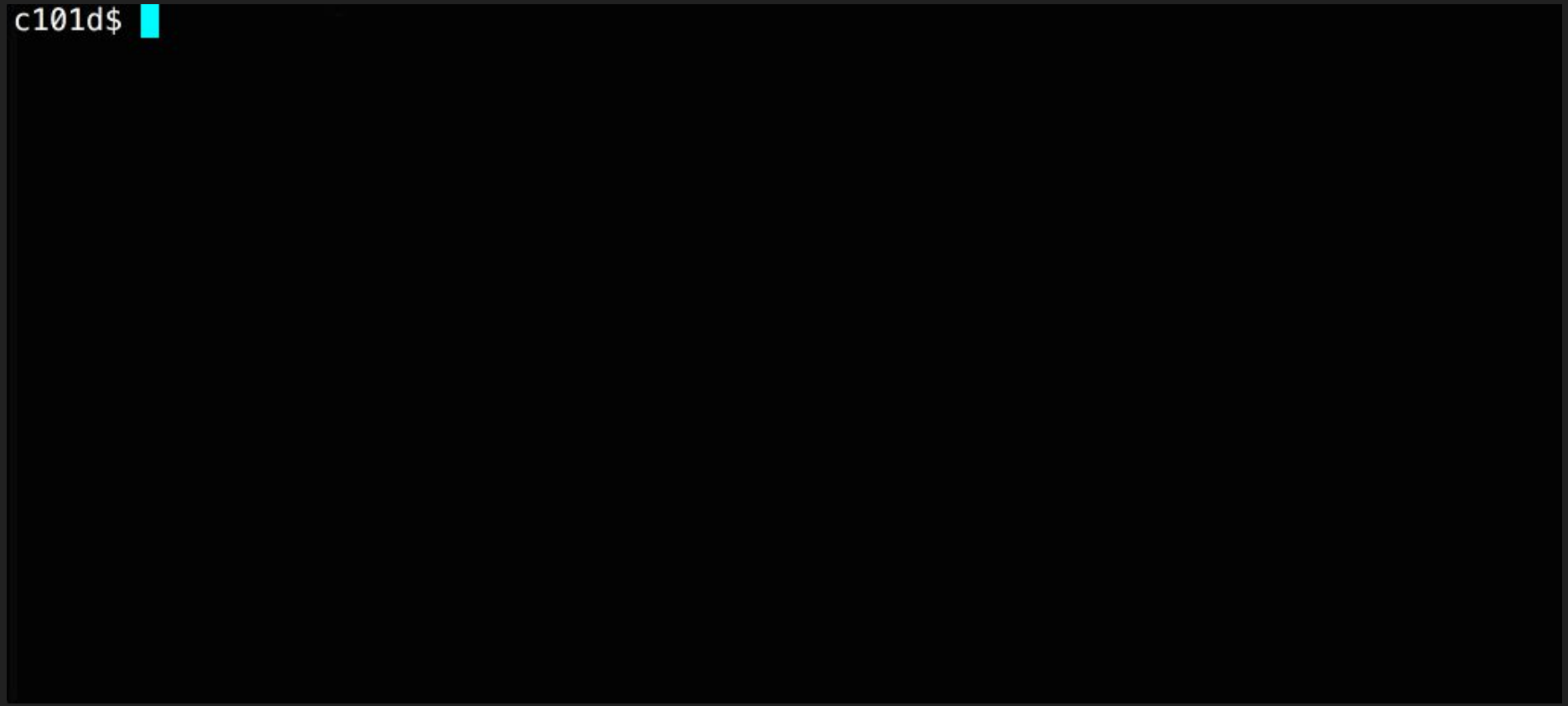
```
root/
    [composer.phar]
    composer.json
    composer.lock
    vendor/
    // everything else...
```

Composer update variations

```
% composer update

% composer update <vendor>/<package>
```

The 'update' command has optional values for defining a specific package to update. If not specified it will update all packages in project.



When the composer `update` command is passed a vendor/package name, it will attempt to update the package to the latest version that matches the version constraints specified in composer.json.

What just happened?

```
% composer update <package>
```

Retrieves information about project and dependencies.

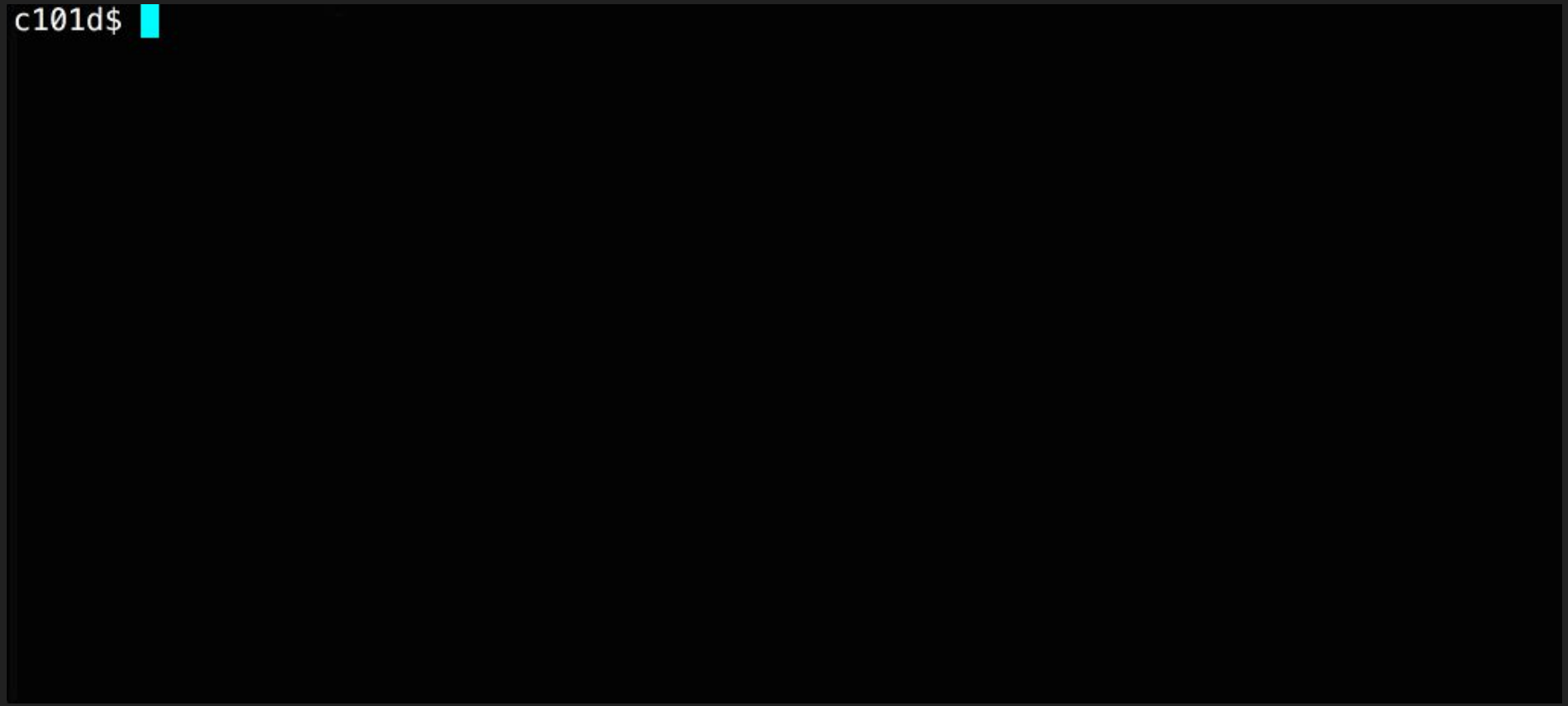
```
Loading composer repositories...
Updating dependencies...
```

Updates package to latest version that meets constraints. As well as, any dependencies.

```
  - Updating behat/behat(3.3.1 => 3.4.3)...
```

Updates composer.lock file is with new package version.

```
Writing lock file
```



When the composer `update` command is run with no package specified, it will attempt to update all project dependencies to the latest versions that match the version constraints specified in composer.json.

What just happened?

```
% composer update
```

Retrieves information about project and dependencies.

```
Loading composer repositories...
Updating dependencies...
```

Updates all packages to latest versions that meets constraints. As well as, any dependencies.

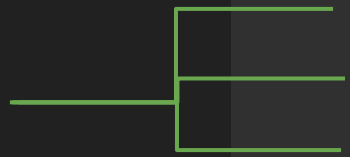
```
  - Updating monolog/monolog(1.20.0 =>...
```

Updates composer.lock file is with new package versions.

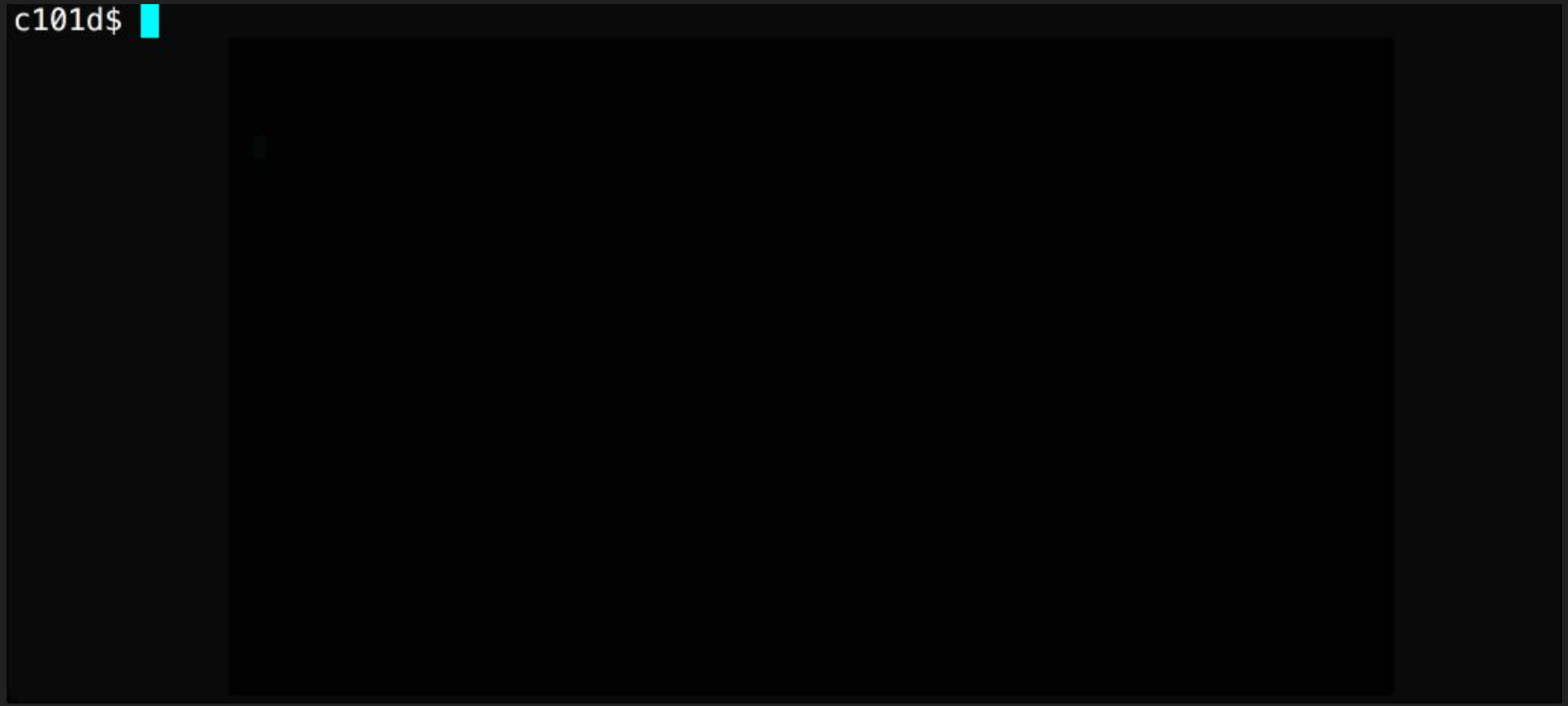
```
Writing lock file
```

remove

```
% composer remove
```



```
root/
    [composer.phar]
    composer.json
    composer.lock
    vendor/
    // everything else...
```



When the composer 'remove' command is run it will delete the specified package and any dependencies not used by other packages from the `vendor` directory. All information for the removed packages will be removed from composer.json and composer.lock files.

What just happened?

```
% composer remove <package>
```

Retrieves information about package and dependencies.

```
Loading composer repositories...
```

Removes package and dependencies from composer.json and composer.lock

```
Updating dependencies...
```

Removes package and all dependencies from vendor directory

```
  - Removing symfony/yml(v3.4.6)
  - Removing sebastian/version(2.0.1)
  ...
  - Removing phpunit/phpunit(5.7.27)
```

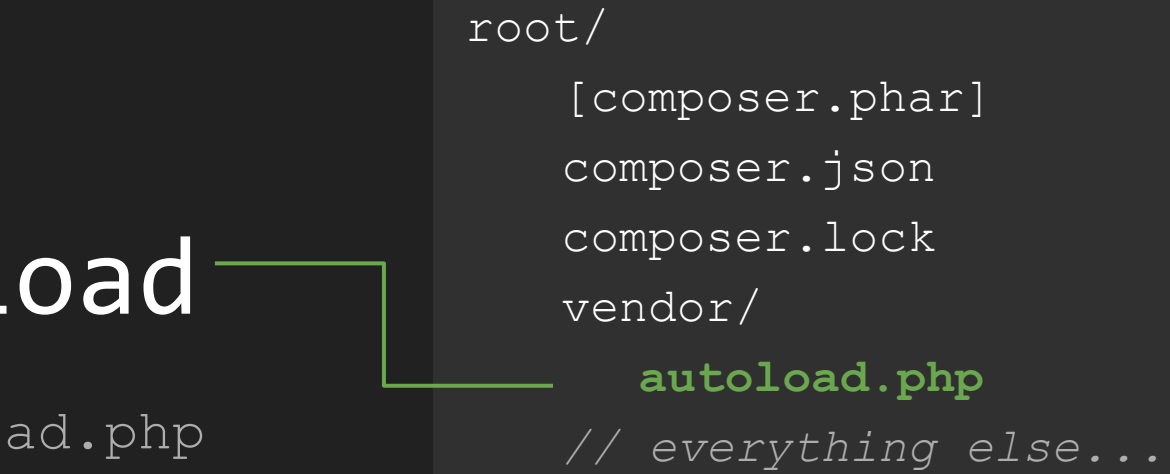
Now what?

```
root/
    [composer.phar]
    composer.json
    composer.lock
    vendor/
    // everything else...
```

Since composer is handling all of the 3rd party code for your PHP project, you can now focus on everything else.

autoload

```
// autoload.php
```



Composer autoload.php

```
<?php

  require __DIR__ . '../vendor/autoload.php';

  $log = new Monolog\Logger('name');
  $log->pushHandler(new Monolog\Handler\StreamHandler('app.log',
Monolog\Logger::WARNING));
  $log->addWarning('Foo');
```

Composer generates an autoload.php file in the `vendor/` directory. This file can be used for PSR-4 autoloading of any installed packages. Use it in your application code to access and use project dependencies.

Links & Resources

> bit.ly/Dcon18Composer

> getcomposer.org

> packagist.org

Questions / Feedback?

@mikemiles86

</presentation>