

Behavioral Driven Development with Behat

by your friend:

Ryan Weaver
@weaverryan



Who is this Hipster?

- The Symfony “Docs” guy
- KnpLabs US - Symfony consulting, training, Kumbaya
- Writer for KnpUniversity.com screencasts



@weaverryan

knplabs.com
github.com/weaverryan



Who is this Hipster?

- The Symfony “Docs” guy
- KnpLabs US - Symfony consulting, training, Kumbaya
- Writer for KnpUniversity.com screencasts
- Husband of the much more talented @leannapelham



@weaverryan

knplabs.com
github.com/weaverryan

 **Knp** Labs

Intro

**Plan, Work, Miscommunicate,
Panic, Put out Fires, Repeat!**
(aka Project Work)

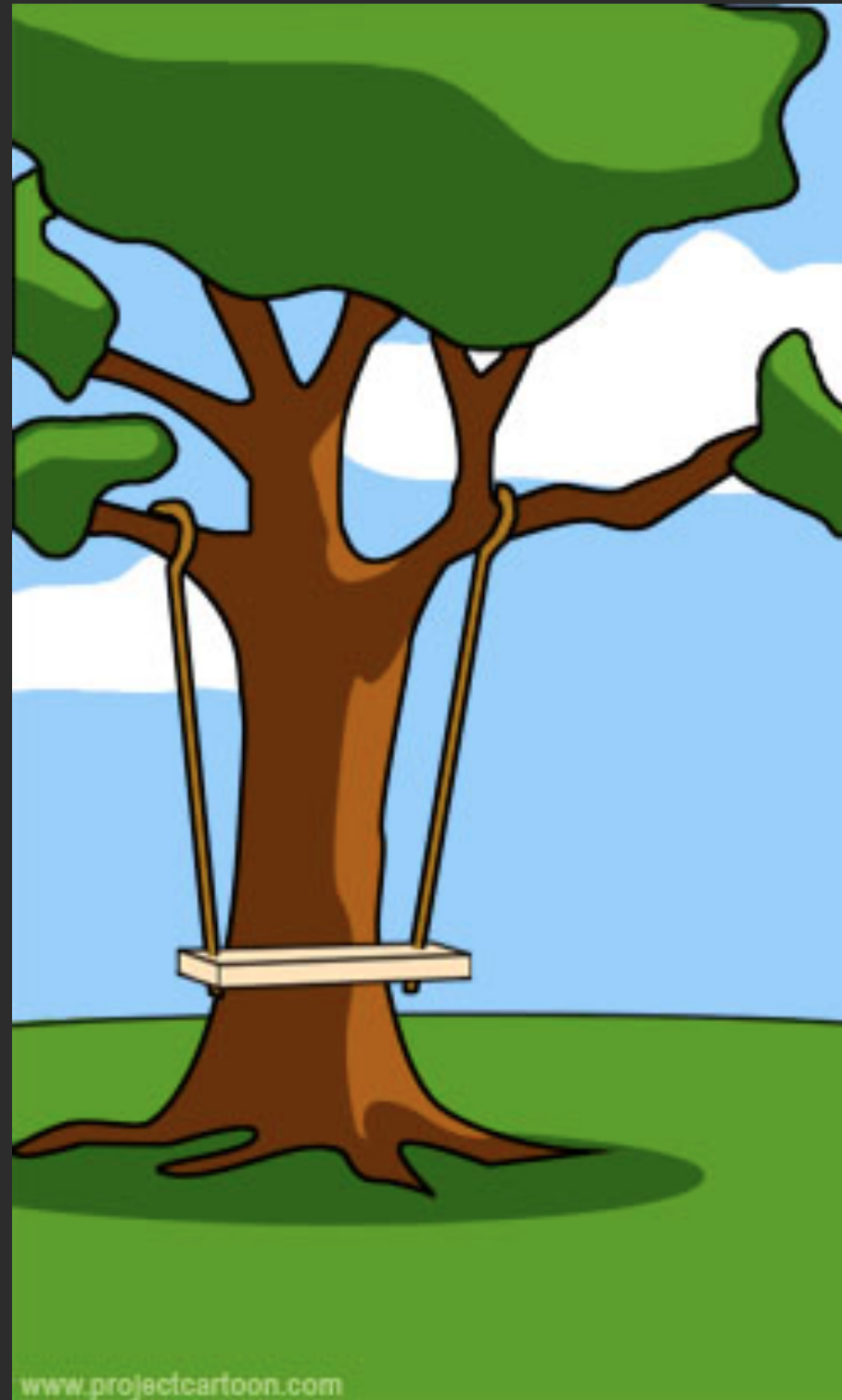
<http://www.flickr.com/photos/lifeontheedge/416514144/>

The Typical Project



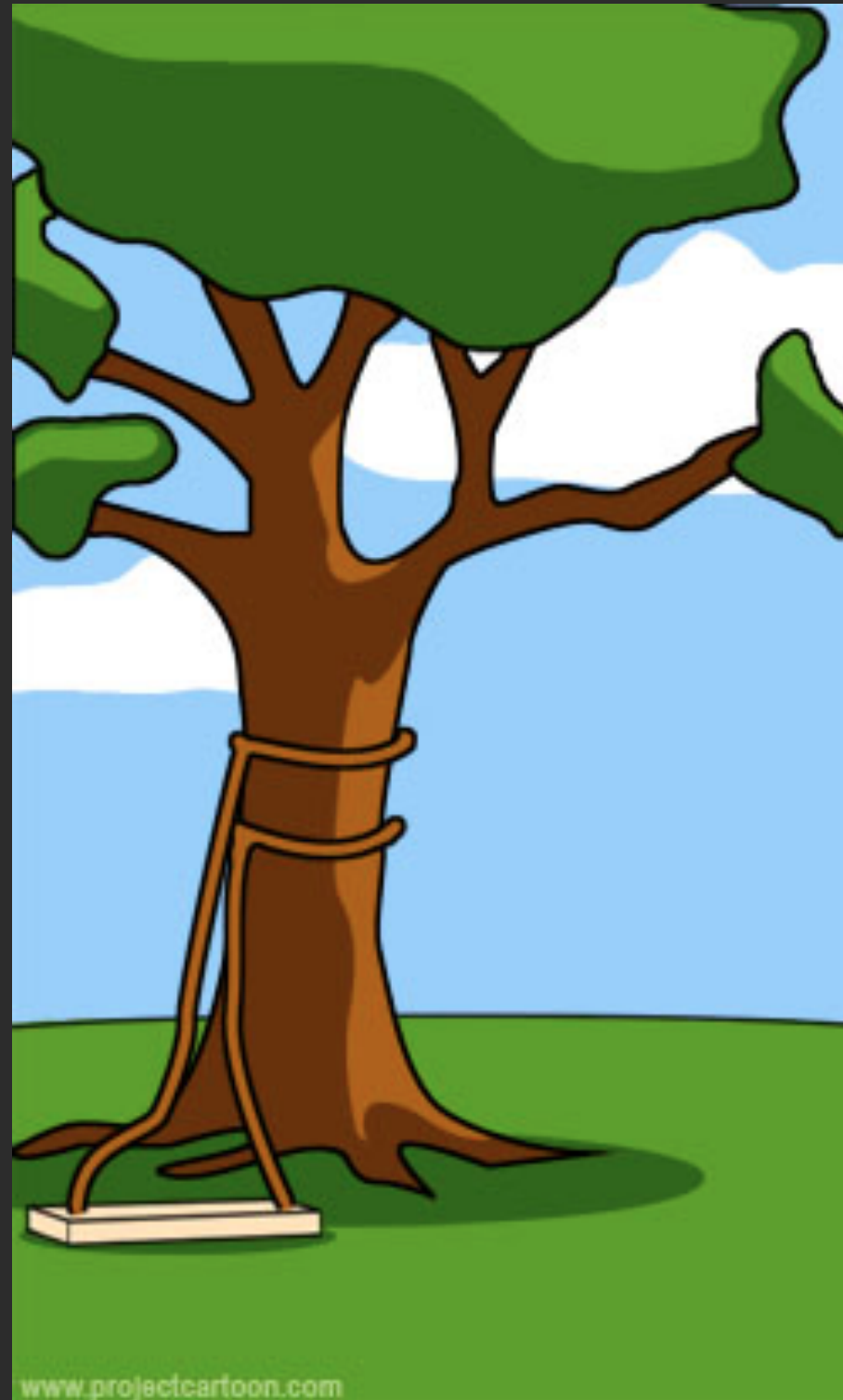
How the customer explained it

<http://www.projectcartoon.com>



How the project leader understood it

<http://www.projectcartoon.com>



How the programmer wrote it

<http://www.projectcartoon.com>



What the customer really needed

<http://www.projectcartoon.com>

... and my personal favorite



What the beta testers received

<http://www.projectcartoon.com>

Computer Science?

Where it breaks down...

1

Different roles,
different languages,
miscommunication

2

Your code and business
values may not align

*I've just dreamt up this
cool new feature that we
should implement!*

Why? Because it's cool!

3

Over-planning, under-planning,
planning...?

Act 1

Getting down with BDD

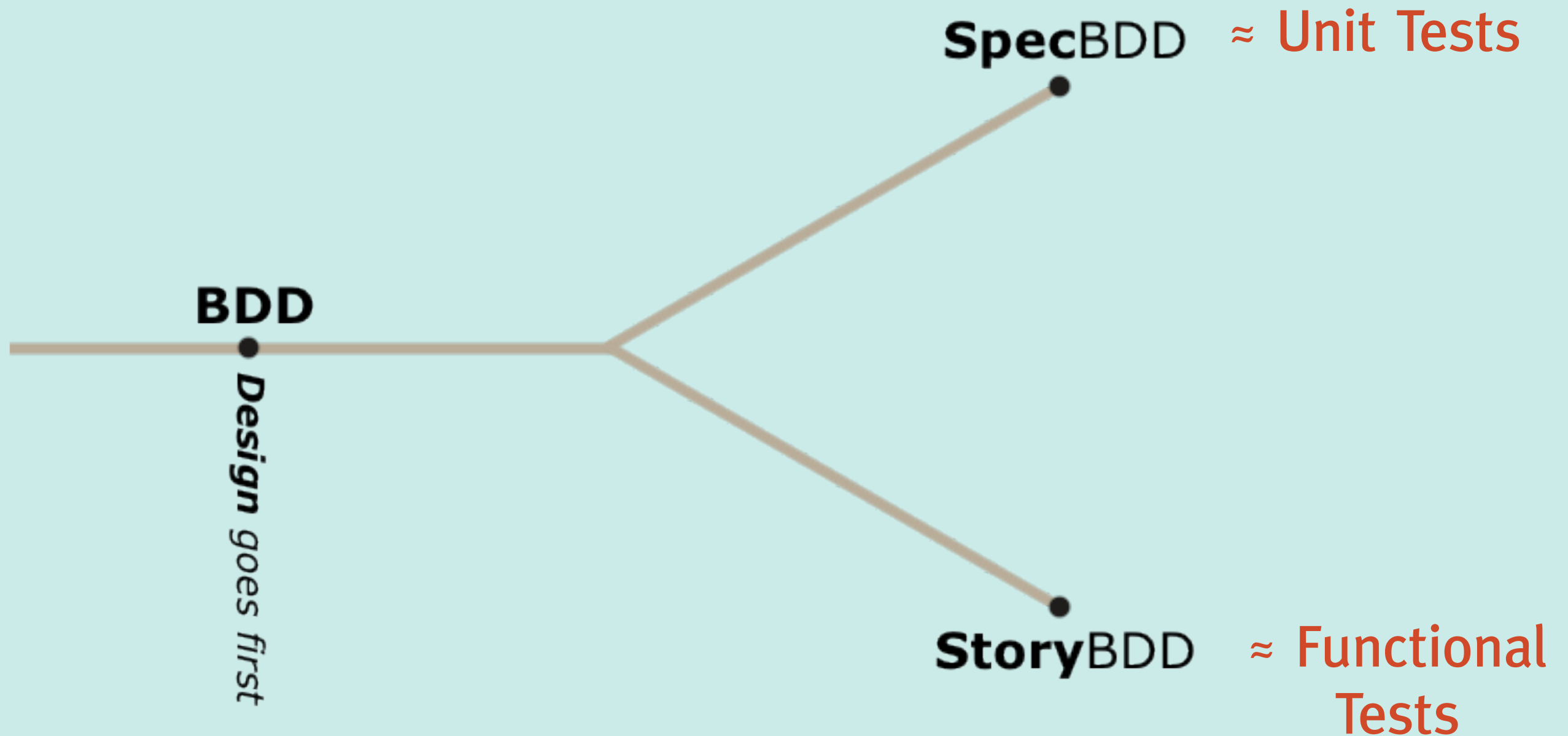
Evolution of Test-Driven Development



**“Behaviour” is a more
useful word, than “test”**

© Dan North, 2003

Evolution of Test-Driven Development



Specification BDD



Scenario-oriented BDD

(Story BDD)

business
analysts

project
managers

product
owners

Let's create a single
vocabulary and process

developers

QA

business
analysts

project
managers

product
owners

... for planning, implementing,
and testing a feature

developers

QA

business
analysts

project
managers

product
owners

... with a focus on the
behavior of the feature

developers

QA

Solution

1. **Define** business value for the features
2. **Prioritize** features by their business value
3. **Describe** them with readable scenarios
4. And only then - **implement** them

Act 2

Gherkin

A project consists of **many**
features

These need to be planned,
written and shared

Gherkin

==

a structured language to
describe a feature


Feature: {custom_title}

In order to {A}

As a {B}

I need to {C}

- {A} - the benefit or **value** of the feature
- {B} - the **role** (or person) who will benefit
- {C} - short feature **description**



| The person “writing” this feature - the “I”

Solution

1. **Define** business value for the features
2. **Prioritize** features by their business value
3. **Describe** them with readable scenarios
4. And only then - **implement** them

read news in French

Feature: I18n

In order to read news in french

As a french user

I need to be able to switch locale

read news in French

The **business value**



Feature: I18n

In order to **read news in french**

As a french user

I need to be able to switch locale

read news in French

The person who benefits

+

The “author” of this feature

Feature: I18n

In order to read news in french

As a french user

I need to be able to switch locale

read news in French

Description of the feature, the action the person will take

Feature: I18n

In order to read news in french

As a french user

I need to be able to switch locale



Solution

1. **Define** business value for the features
2. **Prioritize** features by their business value
3. **Describe** them with readable scenarios
4. And only then - **implement** them

prioritize...

- 1) Feature: News admin panel
- 2) Feature: I18n
- 3) Feature: News list API

Solution

1. **Define** business value for the features
2. **Prioritize** features by their business value
3. **Describe** them with readable scenarios
4. And only then - **implement** them

Feature: News admin panel

In order to maintain a list of news

As a site administrator

I need to be able to edit news

Scenario: Add new article

Given I am on the `"/admin/news"` page

When I click `"New Article"`

And I fill in `"Title"` with `"Learned BDD"`

And I press `"Save"`

Then I should see `"A new article was added"`

Scenarios

Scenario: Add new article

Given I am on the "/admin/news" page

When I click "New Article"

And I fill in "Title" with "Learned BDD"

And I press "Save"

Then I should see "A new article was added"

Given

Defines the initial state of the system for the scenario

Scenarios

Scenario: Add new article

Given I am on the "/admin/news" page

When I click "New Article"

And I fill in "Title" with "Learned BDD"

And I press "Save"

Then I should see "A new article was added"

When

Describes the action taken by the **person/role**

Scenarios

Scenario: Add new article

Given I am on the "/admin/news" page

When I click "New Article"

And I fill in "Title" with "Learned BDD"

And I press "Save"

Then I should see "A new article was added"

Then

Describes the **observable** system state after the action has been performed

Scenarios

Scenario: Add new article

Given I am on the "/admin/news" page

When I click "New Article"

And I fill in "Title" with "Learned BDD"

And I press "Save"

Then I should see "A new article was added"

And/But

Can be added to create multiple

Given/When/Then lines

Example #2

Scenario: List available articles

Given there are 5 news articles

And I am on the `"/admin"` page

When I click `"News Administration"`

Then I should see 5 news articles

Gherkin

gives us a consistent language
for describing **features** and
their **scenarios**

... now let's turn them into
tests!

Act 3

*Those tests ain't gonna
write themselves,*

Behatch

t-shirt: <http://bit.ly/behatch-t>

Having a standard way of
describing features is cool...

... executing those sentences
as functional tests is just
awesome

What is Behat?

Behat does one simple thing:

It Maps Each step to a PHP Callback**

Behat “executes” your scenarios, reading each step and calling the function associated with it

** each line in a scenario is called a “step”

Installing Behat

Behat is just a library that can be installed easily in any project via **Composer**

New to Composer? Free screencast cures it!

KnpUniversity.com/screencast/composer

In your project directory...

1) Download **Composer**

```
$> curl -s http://getcomposer.org/installer | php
```

2) Create (or update) **composer.json** for Behat

<http://bit.ly/behat-composer>

```
{  
  "require": {  
    "behat/behat": "2.4.*@stable"  
  },  
  "minimum-stability": "dev",  
  "config": {  
    "bin-dir": "bin/"  
  }  
}
```

<http://bit.ly/behat-composer>

In your project directory...

1) Download **Composer**

```
$> curl -s http://getcomposer.org/installer | php
```

2) Create (or update) **composer.json** for Behat

```
http://bit.ly/behat-composer
```

3) **Download Behat** libraries

```
$> php composer.phar install
```


\o/ Woo!

The most important product
of the installation is an
executable **bin/behata** file

```
weaverryan@~/Sites/behat$ php bin/behat --help
```

Usage:

```
behat [--init] [-f|--format="..."] [--out="..."] [--lang="..."] [--[no-]ansi] [--no-]snippets-paths] [--[no-]multiline] [--[no-]expand] [--story-syntax] [-d|--detach="..."] [--strict] [--dry-run] [--rerun="..."] [--append-snippets] [features]
```

Arguments:

features

Feature(s) to run. Could be:

- a dir (**features/**)
- a feature (***.feature**)
- a scenario at specific line (***.feature:10**).
- all scenarios at or after a specific line (***.feature:10**)
- all scenarios at a line within a specific range (***.feature:10-20**)

Options:

--init

Create **features** directory structure.

--format (-f)

How to format features. **pretty** is default.

Default formatters are:

- **pretty**: Prints the feature as is.
- **progress**: Prints one character per step.
- **html**: Generates a nice looking HTML report.
- **junit**: Generates a report similar to Ant+JUnit.
- **failed**: Prints list of failed scenarios.
- **snippets**: Prints only snippets for undefined steps.

Can use multiple formats at once (splitted with ",")

--out

Write formatter output to a file/directory

Behat in a project

To use **Behat** in a project you need:

- 1) Actual *.feature files to be executed
- 2) A FeatureContext.php file that holds the PHP callbacks for each step
- 3) (optional) A behat.yml configuration file

```
$> php bin/behata --init
```

```
weaverryan@~/Sites/behata$ php bin/behata --init  
+d features - place your *.feature files here  
+d features/bootstrap - place bootstrap scripts and static files here  
+f features/bootstrap/FeatureContext.php - place your feature related code here
```

```
<?php
// features/bootstrap/FeatureContext.php

use Behat\Behat\Context\BehatContext,
    Behat\Behat\Exception\PendingException;

class FeatureContext extends BehatContext
{
}
```

```
<?php
```

```
// features/bootstrap/FeatureContext.php
```

```
use Behat\Behat\Context\Context;
    Behat\Behat\Exception\PendingException;

    Nothing Interesting here
    yet...
```

```
class FeatureContext extends BehatContext
{
}
}
```

Pretend you're testing the
“ls” program

1) Describe your Feature

Feature: `ls`

In order to `see the directory structure`

As `a UNIX user`

`I need to be able to list the current directory's contents`

`features/ls.feature`

2) Your First Scenario

“If you have two files in a directory, and you're running the command - you should see them listed.”

2) Write Your First Scenario

**** Write in the natural voice of “a UNIX user”**

Scenario: List 2 files in a directory

Given I have a file named "foo"

And I have a file named "bar"

When I run "ls"

Then I should see "foo" in the output

And I should see "bar" in the output

features/ls.feature

3) Run Behat

```
$> php bin/behat
```

```
weaverryan@~/Sites/behats$ php bin/behats
```

```
Feature: ls
```

```
  In order to see the directory structure
```

```
  As a UNIX user
```

```
  I need to be able to list the current directory's contents
```

```
Scenario: List 2 files in a directory
```

```
  Given I have a file named "foo"
```

```
  And I have a file named "bar"
```

```
  When I run "ls"
```

```
  Then I should see "foo" in the output
```

```
  And I should see "bar" in the output
```

```
1 scenario (1 undefined)
```

```
5 steps (5 undefined)
```

```
0m0.018s
```

Behat tries
to find a
method in
FeatureContext
for each step

You can implement step definitions for undefined steps with these snippets:

```
/**
 * @Given /^I have a file named "([^"]*)"$/
 */
public function iHaveAFileNamed($arg1)
{
    throw new PendingException();
}
```

Matching is
done via
regex

```
/**
 * @When /^I run "([^"]*)"$/
 */
public function iRun($arg1)
{
    throw new PendingException();
}
```

For each step that
doesn't have a matching
method, Behat prints
code to copy into
FeatureContext

```
/**
 * @Then /^I should see "([^"]*)" in the output$/
 */
public function iShouldSeeInTheOutput($arg1)
{
    throw new PendingException();
}
```

4) Copy in the new Definitions

```
class FeatureContext extends BehatContext
{
```

```
    /** @Given /^I have a file named "([^"]*)"$/ */
    public function iHaveAFileNamed($arg1)
```

```
{
```

```
    throw new PendingException();
```

```
}
```

```
    /** @When /^I run "([^"]*)"$/ */
    public function iRun($arg1)
```

```
{
```

```
    throw new PendingException();
```

```
}
```

```
// ...
```

```
}
```

Quoted text
maps to a
method
argument

5) Make the definitions do
what they need to


```
/**
 * @Given /^I have a file named "([^"]*)"$/
 */
public function iHaveAFileNamed($file) {
    touch($file);
}
```

```
/**
 * @Given /^I have a directory named "([^"]*)"$/
 */
public function iHaveADirectoryNamed($dir) {
    mkdir($dir);
}
```

```
/**
 * @When /^I run "([^"]*)"$/
 */
public function iRun($command) {
    exec($command, $output);
    $this->output = trim(implode("\n", $output));
}

/**
 * @Then /^I should see "([^"]*)" in the output$/
 */
public function iShouldSeeInTheOutput($string) {
    assertContains(
        $string,
        explode("\n", $this->output)
    );
}
```

```
weaverryan@~/Sites/behat$ php bin/behat
```

```
weaverryan@~/Sites/behat$ php bin/behat
```

```
Feature: ls
```

```
    In order to see the directory structure
```

```
    As a UNIX user
```

```
    I need to be able to list the current directory's contents
```

```
Scenario: List 2 files in a directory
```

```
    Given I have a file named "foo"
```

```
    And I have a file named "bar"
```

```
    When I run "ls"
```

```
    Then I should see "foo" in the output
```

```
    And I should see "bar" in the output
```

```
1 scenario (1 passed)
```

```
5 steps (5 passed)
```

```
0m0.024s
```

```
weaverryan@~/Sites/behat$ █
```



```
weaverryan@~/Sites/behat$ php bin/behat
```

```
Feature: ls
```

```
    In order to see the directory structure
```

```
    As a UNIX user
```

```
    I need to be able to list the current directory's contents
```

```
Scenario: List 2 files in a directory
```

```
    Given I have a file named "foo"
```

```
    And I have a file named "bar"
```

```
    When I run "ls"
```

```
    Then I should see "foo" in the output
```

```
    And I should see "bar" in the output
```

```
1 scenario (1 passed)
```

```
5 steps (5 passed)
```

```
0m0.024s
```

```
weaverryan@~/Sites/behat$ █
```

See the full FeatureContext
class:

<http://bit.ly/behata-ls-feature>

What Behat *does*

Scenario Step

Given I have a file named “foo”

regex

```
@Given /^I have a file named "([^"]*)"$/
```

Definition

```
public function iHaveAFileNamed($file) {
```

do work

```
touch($file);
```

Pass/Fail: Each step is a “test”, which passes
unless an exception is thrown

Creating files and directories in
FeatureContext is nice...

but wouldn't it be really cool
to **command a browser**, fill out
forms and check the output?

Act 4

Mink

Mink!

- A standalone library to use PHP to command a “browser”
- One easy API that can be used to command Selenium, Goutte, ZombieJS, etc

A sample of Mink

```
use Behat\Mink\Driver\GoutteDriver;  
use Behat\Mink\Session;
```

```
// change *only* this line to run  
// in Selenium, etc
```

```
$driver = new GoutteDriver();  
$session = new Session($driver);
```

```
// visit a page
$session->visit('http://behat.org');

echo 'Status: ' . $session->getStatusCode();

echo 'URL : ' . $session->getCurrentUrl();
```

```
$page = $session->getPage();  
  
// drill down into the page  
$ele = $page->find('css', 'li:nth-child(4) a');  
  
echo 'Link text is: ' . $ele->getText();  
echo 'href is: ' . $ele->getAttribute('href');  
  
// click the link  
// (you can also fill out forms)  
$ele->click();
```


Mink inside FeatureContext



Dangerous Combo for Functional Testing

Integration



- An “Extension” is like a Behat plugin
- The MinkExtension makes using Mink inside Behat a matter of configuration

Install Mink & MinkExtension

- Update composer.json to include
 - * Mink
 - * MinkExtension
 - * Goutte and Selenium2 Drivers for Mink

<http://bit.ly/behat-mink-composer>
- Update the vendor libraries

```
$> php composer.phar update
```

```
{  
  "require": {  
    "behat/mink": "1.4@stable",  
    "behat/mink-goutte-driver": "*",  
    "behat/mink-selenium2-driver": "*",  
    "behat/behat": "2.4@stable",  
    "behat/mink-extension": "*"  
  },  
  "minimum-stability": "dev",  
  "config": {  
    "bin-dir": "bin/"  
  }  
}
```

<http://bit.ly/behata-mink-composer>

Goal: To easily use Mink inside
FeatureContext

Bootstrap MinkExtension

```
# behat.yml
default:
  extensions:
    Behat\MinkExtension\Extension:
      goutte: ~
      selenium2: ~
      # The base URL to app you're testing
      base_url: http://en.wikipedia.org/
```

behat.yml is the Behat configuration file and can contain much more than you see here

Extend MinkContext

```
use Behat\MinkExtension\Context\MinkContext;  
  
class FeatureContext extends MinkContext
```

Extending MinkContext gives us 2 things...

1) Access to a Mink Session

```
class FeatureContext extends MinkContext
{
    public function doSomething()
    {
        $session = $this->getSession();
        $session->visit('http://behat.org');
    }

    // ...
}
```

Our custom definitions can now
command a browser!

2) We inherit a pile of great definitions

Before extending MinkContext:

```
weaverryan@~/Sites/behat$ php bin/behat -dl
Given /^I have a file named "([^"]*)"$/
Given /^I have a directory named "([^"]*)"$/
When /^I run "([^"]*)"$/
Then /^I should see "([^"]*)" in the output$/
weaverryan@~/Sites/behat$
```

the **-dl** option prints all current definitions

After extending MinkContext:

```
weaverryan@~/Sites/behat$ php bin/behat -dl
Given /^I have a file named "(?P<name>[^\s]*)"/
Given /^I have a directory named "(?P<name>[^\s]*)"/
When /^I run "(?P<command>[^\s]*)"/
Then /^I should see "(?P<text>[^\s]*)" in the output$/
Given /^(?:II) am on homepage$/
When /^(?:II) go to homepage$/
Given /^(?:II) am on "(?P<page>[^\s]*)"/
When /^(?:II) go to "(?P<page>[^\s]*)"/
When /^(?:II) reload the page$/
When /^(?:II) move backward one page$/
When /^(?:II) move forward one page$/
When /^(?:II) press "(?P<button>(?:[^\s]|\\")*)"/
When /^(?:II) follow "(?P<link>(?:[^\s]|\\")*)"/
When /^(?:II) fill in "(?P<field>(?:[^\s]|\\")*)" with "(?P<value>(?:[^\s]|\\")*)"/
When /^(?:II) fill in "(?P<value>(?:[^\s]|\\")*)" for "(?P<field>(?:[^\s]|\\")*)"/
When /^(?:II) fill in the following:$/
When /^(?:II) select "(?P<option>(?:[^\s]|\\")*)" from "(?P<select>(?:[^\s]|\\")*)"/
When /^(?:II) additionally select "(?P<option>(?:[^\s]|\\")*)" from "(?P<select>(?:[^\s]|\\")*)"/
When /^(?:II) check "(?P<option>(?:[^\s]|\\")*)"/
When /^(?:II) uncheck "(?P<option>(?:[^\s]|\\")*)"/
When /^(?:II) attach the file "(?P<file>[^\s]*)" to "(?P<field>(?:[^\s]|\\")*)"/
Then /^(?:II) should be on "(?P<page>[^\s]*)"/
Then /^the (?i)url(?-i) should match (?P<pattern>"(?P<url>[^\s]*)"/
Then /^the response status code should be (?P<code>\d+)/
Then /^the response status code should not be (?P<code>\d+)/
Then /^(?:II) should see "(?P<text>(?:[^\s]|\\")*)"/
Then /^(?:II) should not see "(?P<text>(?:[^\s]|\\")*)"/
Then /^(?:II) should see text matching (?P<pattern>"(?P<text>[^\s]*)"/
Then /^(?:II) should not see text matching (?P<pattern>"(?P<text>[^\s]*)"/
Then /^the response should contain "(?P<text>(?:[^\s]|\\")*)"/
Then /^the response should not contain "(?P<text>(?:[^\s]|\\")*)"/
Then /^(?:II) should see "(?P<text>(?:[^\s]|\\")*)" in the "(?P<element>[^\s]*)" element$/
Then /^(?:II) should not see "(?P<text>(?:[^\s]|\\")*)" in the "(?P<element>[^\s]*)" element$/
Then /^the "(?P<element>[^\s]*)" element should contain "(?P<value>(?:[^\s]|\\")*)"/
Then /^the "(?P<element>[^\s]*)" element should not contain "(?P<value>(?:[^\s]|\\")*)"/
Then /^(?:II) should see an? "(?P<element>[^\s]*)" element$/
Then /^(?:II) should not see an? "(?P<element>[^\s]*)" element$/
Then /^the "(?P<field>(?:[^\s]|\\")*)" field should contain "(?P<value>(?:[^\s]|\\")*)"/
Then /^the "(?P<field>(?:[^\s]|\\")*)" field should not contain "(?P<value>(?:[^\s]|\\")*)"/
Then /^the "(?P<checkbox>(?:[^\s]|\\")*)" checkbox should be checked$/
Then /^the "(?P<checkbox>(?:[^\s]|\\")*)" checkbox should not be checked$/
Then /^(?:II) should see (?P<num>\d+) "(?P<element>[^\s]*)" elements?$/
Then /^print last response$/
Then /^show last response$/
weaverryan@~/Sites/behat$
```

In other words:

**We can write some tests for
our app without writing any
PHP code**

Suppose we're testing
Wikipedia.org

features/wikipedia.feature

Feature: Search

In order to see a word definition

As a website user

I need to be able to search for a word

Scenario: Searching for a page that does exist

Given I am on "/wiki/Main_Page"

When I fill in "search" with "Behavior Driven Development"

And I press "searchButton"

Then I should see "agile software development"

These 4 definitions all come
packaged with MinkContext

Celebration!

```
weaverryan@~/Sites/behat$ php bin/behat features/wikipedia.feature
```

Feature: Search

In order to see a word definition

As a website user

I need to be able to search for a word

Scenario: Searching for a page that does exist

Given I am on **"/wiki/Main_Page"**

When I fill in **"search"** with **"Behavior Driven Development"**

And I press **"searchButton"**

Then I should see **"agile software development"**

1 scenario (1 passed)

4 steps (4 passed)

0m1.075s

```
weaverryan@~/Sites/behat$ █
```

Act 5

Behat in your application

Getting “under the hood”

- So far, we can do true “black-box” testing - using Mink to test any website (e.g. wikipedia)
- But if we’re testing our app, we don’t have access to it, we can’t:
 - a) access/clear/prepare the database
 - b) use any code in our application

When testing: you should
guarantee the starting
condition of your **environment**

How can we add nodes, add
users, and configure
permissions from inside
Behat?

Behat in your app

1) Install Behat, Mink, MinkExtension

<http://bit.ly/behat-mink-composer>

2) ??? Gain access to Drupal in Behat ???

3) Create nodes, users, etc so that you're testing against a predictable dataset

Introducing...

... a library made by the
Drupal community ...

... which I did not help with ...

DrupalExtension!



A plugin (extension) for Behat and Drupal

<http://bit.ly/drupal-extension>

DrupalExtension

- 1) Even more built-in sentences/definitions
- 2) Build nodes, add users, manage permissions inside Behat
- 3) Operating within Regions
- 4) Hooks to load more sentences/definitions from contrib modules

features/node_manage.feature

Background:

Given I am logged in as a user with the "administrator" role

Scenario: Edit Node

Given I am viewing a "page" node with the title "Cool beans!"

When I click "Edit" in the "Body" region

And I fill in the following:

Body	Ipsumm
------	--------

And I press "Save"

Then I should see "Ipsumm" in the "Body" region

Creates a user and
adds a role to it

Background:

Given I am logged in as a user with the "administrator" role

Scenario: Edit Node

Given I am viewing a "page" node with the title "Cool beans!"

When I click "Edit" in the "Body" region

And I fill in the following:

| Body | Ipsumm |

And I press "Save"

Then I should see "Ipsumm" in the "Body" region

Background:

Given I am logged in as an "administrator" role
Creates a "page" node in the database

Scenario: Edit Node

Given I am viewing a "page" node with the title "Cool beans!"

When I click "Edit" in the "Body" region

And I fill in the following:

| Body | Ipsumm |

And I press "Save"

Then I should see "Ipsumm" in the "Body" region

Background:

Given I am logged in as a user with the "administrator" role

Scenario: Edit Node

Given I am viewing a "page" node with the title "Cool beans!"

When I click "Body" region

And I fill

| Body |

And I press "save"

Looks for the text in
a CSS region you've
defined as "Body"

Then I should see "Ipsumm" in the "Body" region

And it's alive!

```
~/Sites/drupal$ ./bin/behat
```

```
Feature: Node Management
```

```
Background:
```

```
    Given I am logged in as a user with the "administrator" role
```

```
Scenario: Edit Node
```

```
    Given I am viewing a "page" node with the title "Cool beans!"
```

```
    When I click "Edit" in the "Content" region
```

```
    And I fill in the following:
```

```
        | Body | Ipsummm |
```

```
    And I press "Save"
```

```
    Then I should see "Ipsummm" in the "Content" region
```

```
1 scenario (1 passed)
```

```
6 steps (6 passed)
```

```
0m1.57s
```

```
~/Sites/drupal$ █
```

The 3 Modes of the DrupalExtension

- 1) **blackbox**: test an external server, no access to the database
- 2) **drupal**: Bootstraps Drupal's code and calls functions
- 3) **drush**: Interacts with Drupal via drush

and all of this works right
now, in Drupal 7... 6... and 8!

Contrib Modules

- Every module brings its own screens and user-flows
- What if every module brought its own Behat definitions?
 - * Given I have a view....
 - * Given I upload an image and crop it...

The DrupalExtension has a
hook to automatically load
these

... We just need to add these definitions into contrib :)

Want the test to run in
Selenium?

Add @javascript

...

@javascript

Scenario: Edit Node

Given I am viewing a "page" node with the title "Cool beans!"

When I click "Edit" in the "Body" region

And I fill in the following:

Body	Ipsumm
------	--------

And I press "Save"

Then I should see "Ipsumm" in the "Body" region

Add @javascript

...

Yep, that's all you do!

@javascript

Scenario: Edit Node

Given I am viewing a "page" node with the title "Cool beans!"

When I click "Edit" in the "Body" region

And I fill in the following:

Body	Ipsumm
------	--------

And I press "Save"

Then I should see "Ipsumm" in the "Body" region

Download and start Selenium

```
$> wget http://selenium.googlecode.com/files/  
selenium-server-standalone-2.31.0.jar
```

```
$> java -jar selenium-server-standalone-2.31.0.jar
```

Re-run the tests

```
~/Sites/drupal$ ./bin/behat
```

```
Feature: Node Management
```

```
Background:
```

```
    Given I am logged in as a user with the "administrator" role
```

```
@javascript
```

```
Scenario: Edit Node
```

```
    Given I am viewing a "page" node with the title "Cool beans!"
```

```
    When I click "Edit" in the "Content" region
```

```
1 scenario (1 passed)
```

```
3 steps (3 passed)
```

```
0m5.036s
```

```
—
```

Yes, add only **1 line** of code
to run a test in Selenium

Epilogue

You're Turn!

1) Install DrupalExtension

<http://bit.ly/drupal-extension>

This will also install Behat, Mink and MinkContext in your project

2) Write features for your app!

... and learn more about what you can do with
Mink: <http://mink.behat.org/>

3) high-five your teammates

... and do your homework

1) Behat & Mink Screencast:

KnpUniversity.com/screencast/behata

Coupon: HIPSTER

2) Talk with [OpenSourcery](#), read their blogs, give them jumping high-fives

3) Chat with [Melissa Anderson](#) (eliza411) - she'll make you think of the testing process holistically: from geek to biz person

Thanks...



Ryan Weaver
@weaverryan

SPECIAL thanks to
Jonathan Hedstrom
(jhedstrom)

KnpUniversity.com

PHP, Behat, Twig, OO, etc Tutorial Screencasts



... and we love you!



Ryan Weaver
@weaverryan

 **Knip** Labs

Coupon: HIPSTER



KnPUniversity.com

@weaveryan
@KnPUniversity

