

Continuous Integration with GitHub

Table of Contents

1. What for?	2
1.1. Continuous code verification.....	2
1.2. Environment	2
1.3. eXtreme Programming	2
1.4. Principles	2
1.5. YAML	3
1.6. Usage	4
2. Concrete illustration (GitHub).....	4
2.1. Processus type	4
2.2. Example (this repo: page generation, source: here).....	4
2.3. ⚠ Danger Zone	5
2.4. Solution	6
3. HelloWorld example	6
3.1. Java code	7
3.2. Testing with a main	7
3.3. Manual compilation.....	8
3.4. Ant build	8
3.5. Manual build	9
3.6. Improvements	9
3.7. Manual build (improved).....	10
3.8. Tests	11
3.9. Manual build (again).....	13
3.10. Eclipse	15
3.11. Continue Integration	19
4. Tips	19
4.1. How NOT to run CI	19
4.2. To check YAML syntax.....	20
5. GitHub CI = Actions	20
5.1. Click and install	20
5.2. Lots of help and documentation	21
5.3. Where to check/tune?	22
6. 2021 (awesome!) examples	22
7. Useful links	24

1. What for?

1.1. Continuous code verification

Add intro.adoc		9ea171a	<>
Improve tests		ef91c04	<>
Improve links check		4efec66	<>
Clean files		91dbdeb	<>
Add reqs html		73d2e3b	<>

Figure 1. Traced and visible results

1.2. Environment

We illustrate here the possibilities linked with [GitHub Actions](#), but others can be used:

- [circleci](#)
- [Jenkins](#) (previously Hudson)
- [Cruise Control](#)
- [Travis CI](#)
- [Gitlab CI](#)



Here is a complete list: <https://github.com/ligurio/awesome-ci>, and here is a comparison: [Wikipedia](#)

1.3. eXtreme Programming

Continuous Integration is a software development practice where members of a team integrate their work frequently, [...] leading to multiple integrations per day.

— Martin Fowler

1.4. Principles

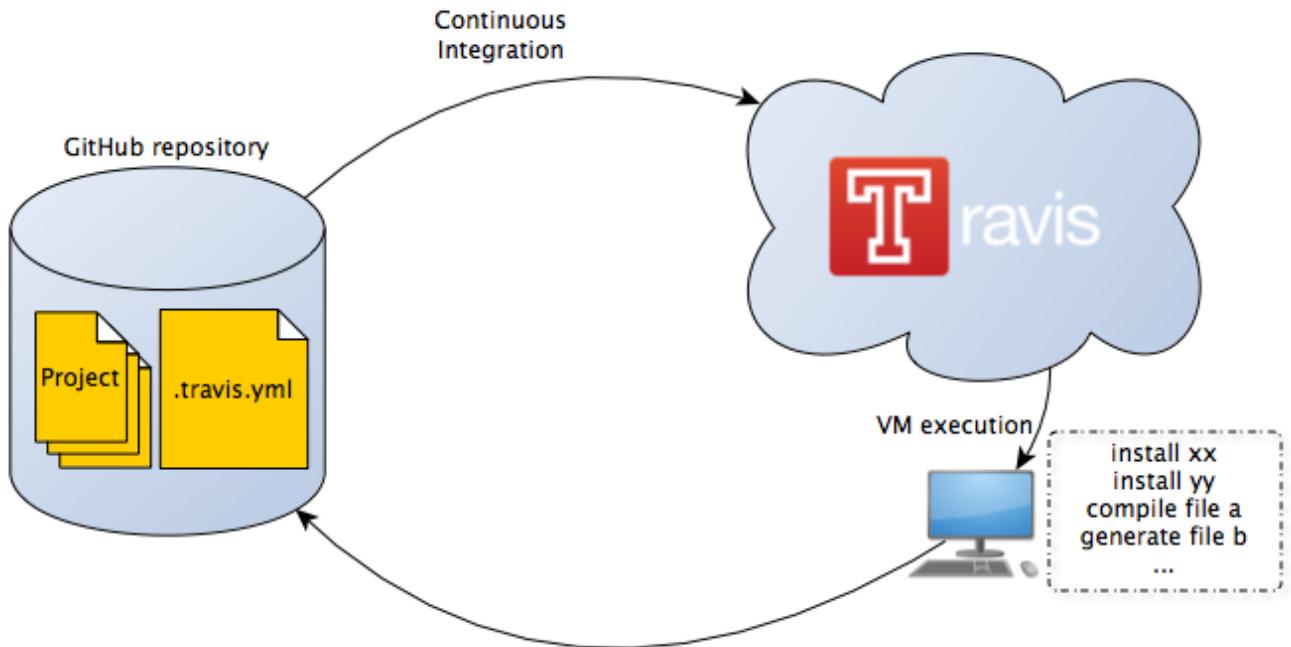


Figure 2. Typical architecture (e.g., *github-travis*)

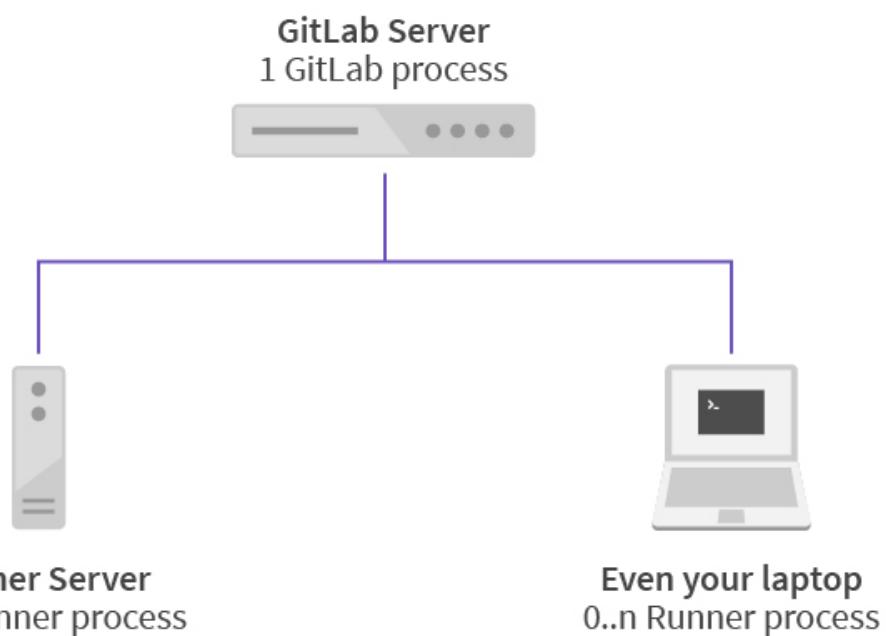


Figure 3. Architecture GitLab-CI (<https://about.gitlab.com/gitlab-ci/>)

1.5. YAML

YAML: YAML Ain't Markup Language

Example of .yml file

```
---  
receipt: Oz-Ware Purchase Invoice  
date: 2012-08-06  
customer:  
  first_name: Dorothy  
  family_name: Gale
```



Use spaces but not tabs.

1.6. Usage

- Compile source code ⇒ **build**
- Execute **tests** suites (Junit, Audit de code source, test IHM, ...)
- Create archives
- Do some git operations (pull, checkout, push)
- Deploy code on a production machine
- Notify results (mail, RSS)
- etc.

2. Concrete illustration (GitHub)

2.1. Processus type

1. Add some specific files at a specific place
2. Describe what needs to be done in those files
3. "push" in the repo
4. Check results

2.2. Example (this repo: page generation, source: [here](#))

```

name: Jekyll site CI ①

on: ②
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

jobs:
  build:

    runs-on: ubuntu-latest ③

    steps:
      - uses: actions/checkout@v2
      - name: Build the site in the jekyll/builder container
        run: | ④
          docker run \
            -v ${{ github.workspace }}:/srv/jekyll -v ${{ github.workspace }}/_site:/srv/jekyll/_site \
              jekyll/builder:latest /bin/bash -c "chmod 777 /srv/jekyll && jekyll build --future"

```

① Name of the CI (to check on github)

② Specification of which branches are concerned

③ Name of the "runner" (virtual machine)

④ Instructions (here running jekyll pages generation)

Add menus trick in asciidoc

master · 569fb4b

> Tests on: push	GitHub Pages / Page Build succeeded 6 days ago in 0s
> Checks on: push	GitHub Pages successfully built your site.
> SonarCloud	Your website is ready at https://jmbruel.github.io/teachingMaterials
✓ GitHub Pages	View more details on GitHub Pages
✓ Page Build	

Figure 4. Result from previous script

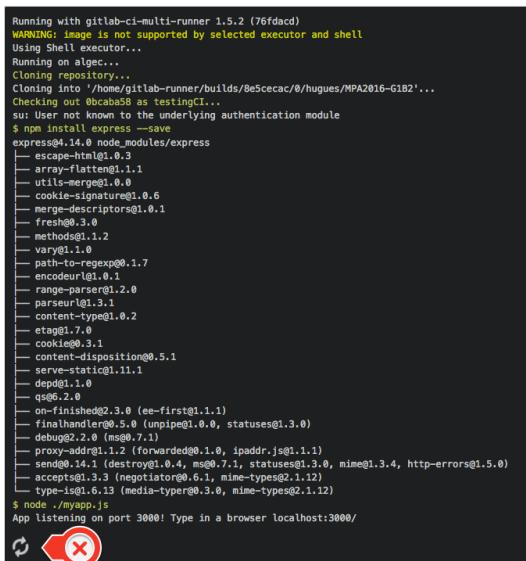
2.3. ⚡ Danger Zone

```

image: node:4.2.2

all_tests:
  script:
    - npm install express --save
    - node ./myapp.js

```



```

Running with gitlab-ci-multi-runner 1.5.2 (76fdacd)
WARNING: image is not supported by selected executor and shell
Using Shell executor...
Running on algec...
Cloning repository...
Cloning into '/home/gitlab-runner/builds/Be5cecac/0/hugues/MPA2016-G1B2'...
Checking out 0cabab58 as testingCI...
su: User not known to the underlying authentication module
$ npm install express --save
express@4.14.0 node_modules/express
├── escape-html@1.0.3
├── array-flatten@1.1.1
├── utils-merge@1.0.0
├── cookie-signature@1.0.6
├── merge-descriptors@1.0.1
├── fresh@0.3.0
└── method-override@1.2
  └── parseurl@1.3.1
    └── content-type@1.0.2
      └── etag@1.7.0
        └── cookie@0.3.1
          └── content-disposition@0.5.1
            └── serve-static@1.11.1
              └── depd@1.1.0
                └── qs@6.2.0
                  └── on-finished@2.3.0 (ee-first@1.1.1)
                    └── finalhandler@0.5.0 (unpipe@1.0.0, statuses@1.3.0)
                      └── debug@2.2.0 (ms@0.7.1)
                        └── proxy-addr@1.1.2 (forwarded@0.1.0, ipaddr.js@1.1.1)
                          └── send@0.14.1 (destroy@1.0.4, ms@0.7.1, statuses@1.3.0, mime@1.3.4, http-errors@1.5.0)
                            └── accept@1.3.3 (negotiator@0.6.1, mime-types@2.1.12)
                              └── type-is@1.6.13 (media-type@0.3.0, mime-types@2.1.12)
$ node ./myapp.js
App listening on port 3000! Type in a browser localhost:3000/

```

Figure 5. Non terminating script

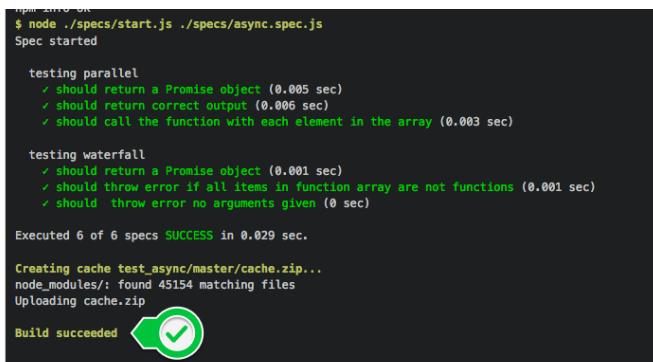
2.4. Solution

Here is a correct `.gitlab-ci.yml` (executing tests):

```

- npm install express --save
- node ./specs/start.js ./specs/async.spec.js

```



```

$ node ./specs/start.js ./specs/async.spec.js
Spec started

testing parallel
  ✓ should return a Promise object (0.005 sec)
  ✓ should return correct output (0.006 sec)
  ✓ should call the function with each element in the array (0.003 sec)

testing waterfall
  ✓ should return a Promise object (0.001 sec)
  ✓ should throw error if all items in function array are not functions (0.001 sec)
  ✓ should throw error no arguments given (0 sec)

Executed 6 of 6 specs SUCCESS in 0.029 sec.

Creating cache test_async/master/cache.zip...
node_modules/: found 45154 matching files
Uploading cache.zip

Build succeeded

```

Figure 6. Build success

3. HelloWorld example

1. Java Code
2. Main to test

3. Manual Compilation

4. ant Build

5. Improvements

6. Tests

7. Eclipse

8. Continue Integration

3.1. Java code



Tutorial for [ant here](#).

src/HelloWorld.java

```
package org.jmb;
public class HelloWorld
{
    private String name = "";
    public String getName()
    {
        return name;
    }
    public String getMessage()
    {
        if (name == "")
        {
            return "Hello!";
        }
        else
        {
            return "Hello " + name + "!";
        }
    }
    public void setName(String name)
    {
        this.name = name;
    }
}
```

3.2. Testing with a main

src/Main.java

```
package org.jmb;

public class Main {
    public static void main(String[] args) {
        org.jmb.HelloWorld h = new org.jmb.HelloWorld();
        h.setName("JMB");
        System.out.println(h.getMessage());
    }
}
```

3.3. Manual compilation

```
$ javac -sourcepath src -d bin/ src/Main.java
$ ls bin
HelloWorld.class  Main.class
$ java -cp bin Main
Hello JMB!
```

3.4. Ant build

```
<project>
    <target name="clean">
        <delete dir="bin"/>
    </target>

    <target name="build">
        <mkdir dir="bin"/>
        <javac srcdir="src" destdir="bin"/>
    </target>

    <target name="jar">
        <mkdir dir="bin/jar"/>
        <jar destfile="bin/jar/HelloWorld.jar" basedir="bin">
            <manifest>
                <attribute name="Main-Class" value="Main"/>
            </manifest>
        </jar>
    </target>

    <target name="run">
        <java jar="bin/jar/HelloWorld.jar" fork="true"/>
    </target>

</project>
```

3.5. Manual build

```
$ ant clean build jar
$ ant run
Buildfile: /Users/bruel/HelloWorld/build.xml

run:
    [java] Hello JMB!

BUILD SUCCESSFUL
Total time: 0 seconds
```

3.6. Improvements

build.xml (improved)

```
<project name="HelloWorld" basedir=". " default="main">

    <property name="src.dir"      value="src"/>
    <property name="build.dir"    value="bin"/>
    <property name="classes.dir" value="${build.dir}/classes"/>
    <property name="jar.dir"      value="${build.dir}/jar"/>

    <property name="main-class"   value="Main"/>

    <target name="clean">
        <delete dir="${build.dir}"/>
    </target>

    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="${jar.dir}"/>
        <jar destfile="${jar.dir}/${ant.project.name}.jar" basedir="${classes.dir}">
            <manifest>
                <attribute name="Main-Class" value="${main-class}"/>
            </manifest>
        </jar>
    </target>

    <target name="run" depends="jar">
        <java jar="${jar.dir}/${ant.project.name}.jar" fork="true"/>
    </target>

    <target name="clean-build" depends="clean,jar"/>

    <target name="main" depends="clean,run"/>

</project>
```

3.7. Manual build (improved)

```

$ ant
Buildfile: /Users/bruel/HelloWorld/build.xml

clean:
[delete] Deleting directory /Users/bruel/HelloWorld/bin

compile:
[mkdir] Created dir: /Users/bruel/HelloWorld/bin/classes
[javac] Compiling 2 source files to /Users/bruel/HelloWorld/bin/classes

jar:
[mkdir] Created dir: /Users/bruel/HelloWorld/bin/jar
[jar] Building jar: /Users/bruel/HelloWorld/bin/jar/HelloWorld.jar

run:
[java] Hello JMB!

main:

BUILD SUCCESSFUL
Total time: 1 second

```

3.8. Tests

src/TestHelloWorld.java (failing)

```

package org.jmb;
public class TestHelloWorld extends junit.framework.TestCase {

    public void testNothing() {
    }

    public void testWillAlwaysFail() {
        fail("An error message");
    }
}

```

build.xml (librairies extérieures)

```

<project name="HelloWorld" basedir=". " default="main">

    <property name="src.dir"      value="src"/>
    <property name="build.dir"    value="bin"/>
    <property name="classes.dir"  value="${build.dir}/classes"/>
    <property name="jar.dir"      value="${build.dir}/jar"/>

    <property name="main-class"   value="Main"/>

```

```

<property name="lib.dir"      value="lib"/>

<path id="classpath">
    <fileset dir="${lib.dir}" includes="**/*.jar"/>
</path>

<target name="clean">
    <delete dir="${build.dir}"/>
</target>

<target name="compile">
    <mkdir dir="${classes.dir}"/>
    <javac srcdir="${src.dir}" destdir="${classes.dir}"
classpathref="classpath"/>
</target>

<target name="jar" depends="compile">
    <mkdir dir="${jar.dir}"/>
    <jar destfile="${jar.dir}/${ant.project.name}.jar" basedir="${classes.dir}">
        <manifest>
            <attribute name="Main-Class" value="${main-class}"/>
        </manifest>
    </jar>
</target>

<path id="application" location="${jar.dir}/${ant.project.name}.jar"/>

<target name="run" depends="jar">
    <java fork="true" classname="${main-class}">
        <classpath>
            <path refid="classpath"/>
            <path refid="application"/>
            <path location="${jar.dir}/${ant.project.name}.jar"/>
        </classpath>
    </java>
</target>

<target name="junit" depends="jar">
    <junit printsummary="yes">
        <classpath>
            <path refid="classpath"/>
            <path refid="application"/>
        </classpath>

        <batchtest fork="yes">
            <fileset dir="${src.dir}" includes="TestHelloWorld.java"/>
        </batchtest>
    </junit>
</target>

```

```
<target name="clean-build" depends="clean,jar"/>  
  
<target name="main" depends="clean,run"/>  
  
</project>
```

3.9. Manual build (again)

```
$ ant junit  
Buildfile: /Users/bruel/HelloWorld/build.xml  
  
compile:  
  
jar:  
  
junit:  
  [junit] Running TestHelloWorld  
  [junit] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0,003 sec  
  [junit] Test TestHelloWorld FAILED  
  
BUILD SUCCESSFUL  
Total time: 1 second
```

```
package org.jmb;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class TestHelloWorldReal {

    private org.jmb.HelloWorld h;

    @Before
    public void setUp() throws Exception
    {
        h = new org.jmb.HelloWorld();
    }

    @Test
    public void testHelloEmpty()
    {
        assertEquals(h.getName(),"");
        assertEquals(h.getMessage(),"Hello!");
    }

    @Test
    public void testHelloWorld()
    {
        h.setName("World");
        assertEquals(h.getName(),"World");
        assertEquals(h.getMessage(),"Hello World!");
    }
}
```

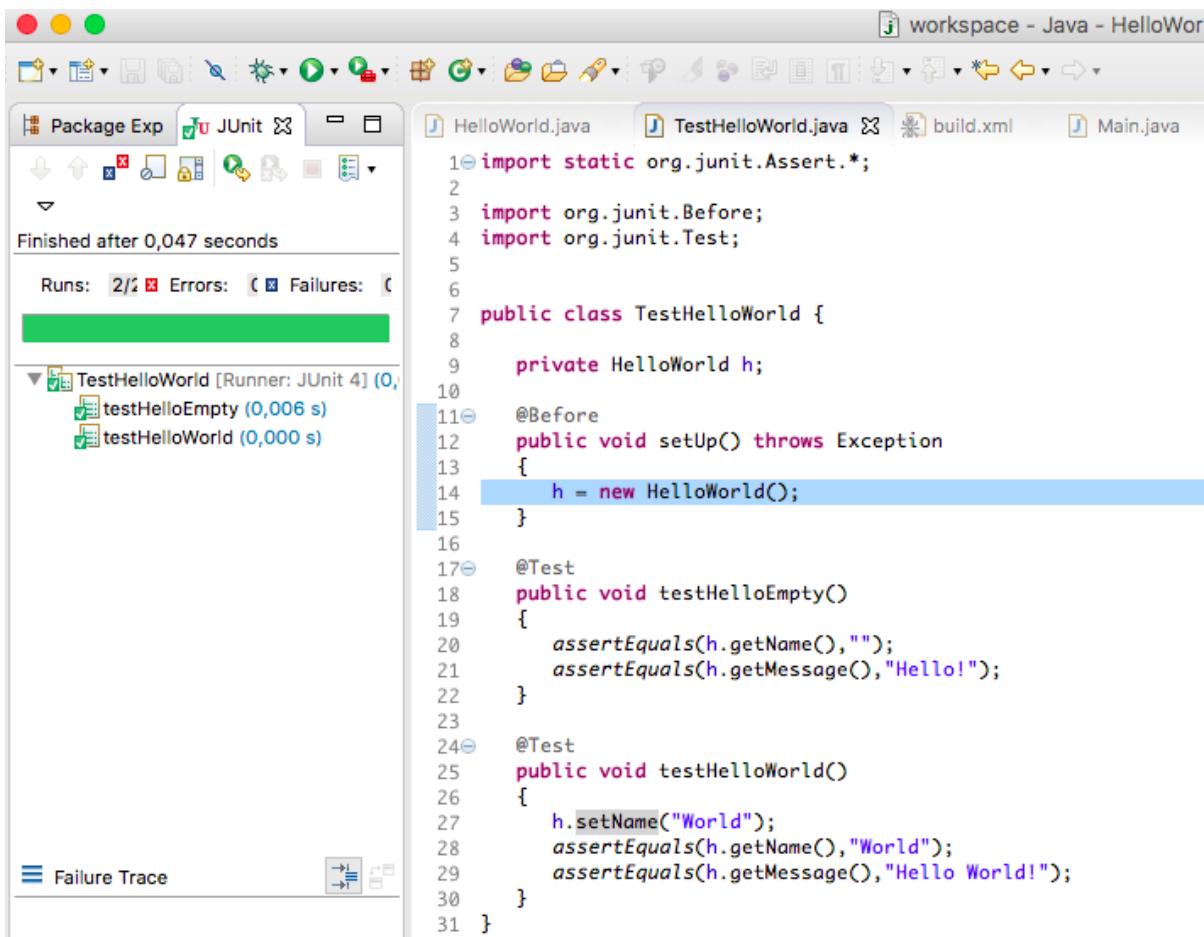


Figure 7. Test under eclipse

Do not hesitate to use the [infinitest](#) plugin.

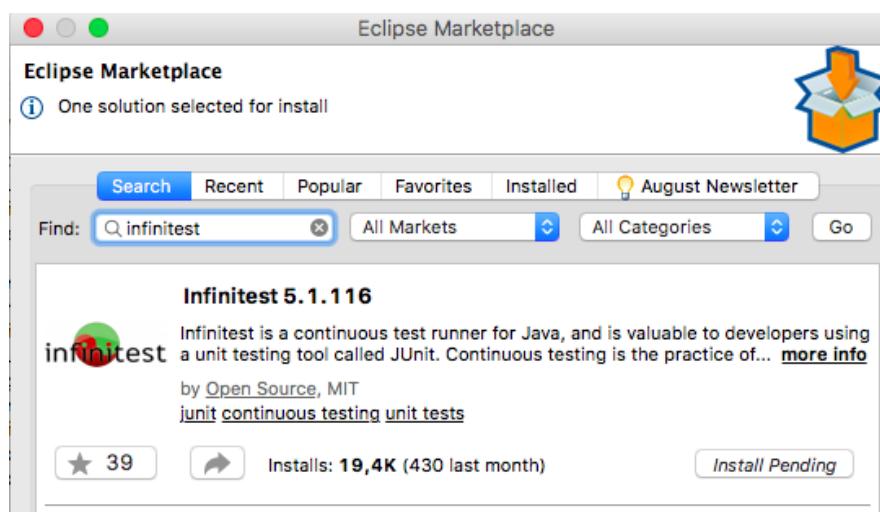


Figure 8. Infinitest plugin

3.10. Eclipse

Directly generate the build [ant](#) with [Eclipse!](#)

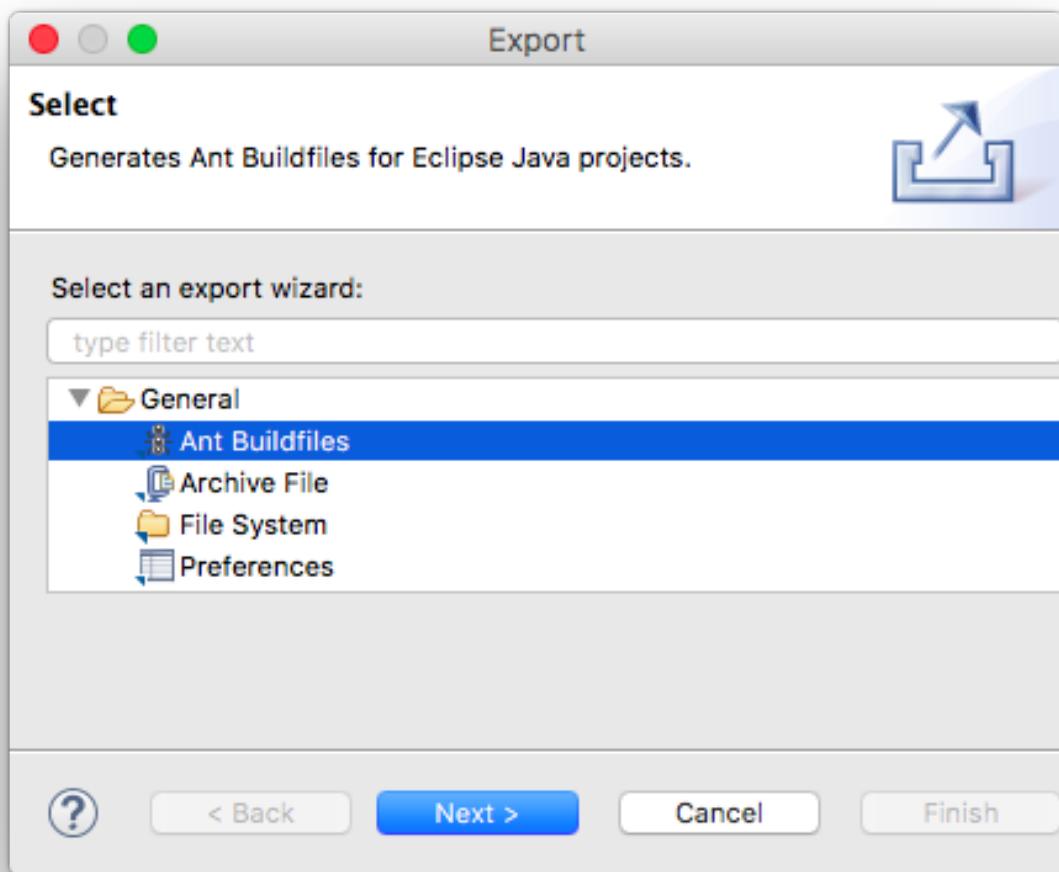


Figure 9. Export with eclipse

build.xml (export from eclipse)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- WARNING: Eclipse auto-generated file.
Any modifications will be overwritten.
To include a user specific buildfile here, simply create one in the same
directory with the processing instruction <?eclipse.ant.import?>
as the first entry and export the buildfile again. --><project
basedir="."
default="build"
name="HelloWorld">
<property environment="env"/>
<property name="junit.output.dir" value="junit"/>
<property name="debuglevel" value="source,lines,vars"/>
<property name="target" value="1.8"/>
<property name="source" value="1.8"/>
<path id="JUnit 4.libraryclasspath">
    <pathelement
location="../../../../p2/pool/plugins/org.junit_4.12.0.v201504281640/junit.jar"/>
    <pathelement
location="../../../../p2/pool/plugins/org.hamcrest.core_1.3.0.v201303031735.jar"/>
</path>
```

```

<path id="HelloWorld.classpath">
    <pathelement location="bin"/>
    <path refid="JUnit 4.libraryclasspath"/>
</path>
<target name="init">
    <mkdir dir="bin"/>
    <copy includeemptydirs="false" todir="bin">
        <fileset dir="src">
            <exclude name="**/*.launch"/>
            <exclude name="**/*.java"/>
        </fileset>
    </copy>
</target>
<target name="clean">
    <delete dir="bin"/>
</target>
<target depends="clean" name="cleanall"/>
<target depends="build-subprojects,build-project" name="build"/>
<target name="build-subprojects"/>
<target depends="init" name="build-project">
    <echo message="${ant.project.name}: ${ant.file}"/>
    <javac debug="true" debuglevel="${debuglevel}" destdir="bin"
includeantruntime="false" source="${source}" target="${target}">
        <src path="src"/>
        <classpath refid="HelloWorld.classpath"/>
    </javac>
</target>
<target description="Build all projects which reference this project. Useful to
propagate changes." name="build-refprojects"/>
<target name="Main">
    <java classname="org.jmb.Main" failonerror="true" fork="yes">
        <classpath refid="HelloWorld.classpath"/>
    </java>
</target>
<target name="TestHelloWorld">
    <mkdir dir="${junit.output.dir}"/>
    <junit fork="yes" printsummary="withOutAndErr">
        <formatter type="xml"/>
        <test name="org.jmb.TestHelloWorld" todir="${junit.output.dir}"/>
        <jvmarg line="-ea"/>
        <classpath refid="HelloWorld.classpath"/>
    </junit>
</target>
<target name="junitreport">
    <junitreport todir="${junit.output.dir}">
        <fileset dir="${junit.output.dir}">
            <include name="TEST-*.xml"/>
        </fileset>
        <report format="frames" todir="${junit.output.dir}"/>
    </junitreport>
</target>

```

```
</project>
```

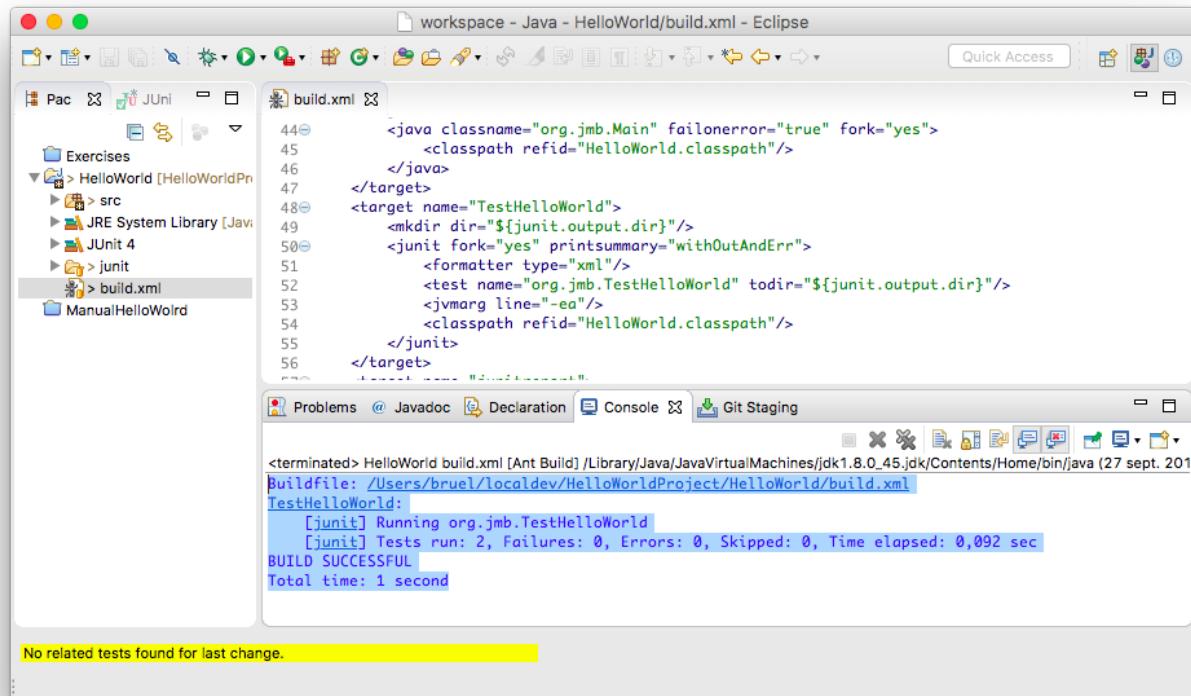


Figure 10. Run ant with eclipse

Another advantage of junit web doc generation ([junit](#) repo, open the `index.html` file).

The screenshot shows a web browser window titled "Unit Test Results." The URL is `file:///Users/bruel/localdev/HelloWorldProject/HelloWorld/junit`. The left sidebar has navigation links for "Home", "Packages" (with options "<none>" and "org.jmb"), "org.jmb", and "Classes" (with option "TestHelloWorld"). The main content area is titled "Unit Test Results." and includes a note "Designed for use with [JUnit](#) and [Ant](#)." It shows two tables: one for the "Class org.jmb.TestHelloWorld" and another for the "Tests".

Name	Tests	Errors	Failures	Skipped	Time(s)	Time Stamp	Host
TestHelloWorld	2	0	0	0	0.102	2016-09-27T09:10:09	dhcp-14-62.laas.fr

Name	Status	Type	Time(s)
testHelloEmpty	Success		0.001
testHelloWorld	Success		0.000

[Properties »](#)

Figure 11. Testing results

3.11. Continue Integration

!.gitlab-ci.yml (added on a GitLab project)

```
image: asciidoctor/docker-asciidoc

variables:
  GIT_SSL_NO_VERIFY: "1"

stages:
  - build
  - test

html:
  stage: build
  script:
    - asciidoctor README.adoc -o index.html
  artifacts:
    paths:
      - index.html

pdf_preview:
  stage: test
  when: manual
  environment:
    name: preview/$CI_COMMIT_REF_NAME
  except:
    - /master/
  artifacts:
    paths:
      - README.pdf
    expire_in: 1 week
  script:
    - asciidoctor-pdf README.adoc
```



The `build.xml` might have to be updated because sometimes too linked to [Eclipse](#).

4. Tips

4.1. How NOT to run CI

```
git commit -m "Blabla... [ci skip]"
```

4.2. To check YAML syntax

<https://docs.gitlab.com/ee/ci/lint.html>

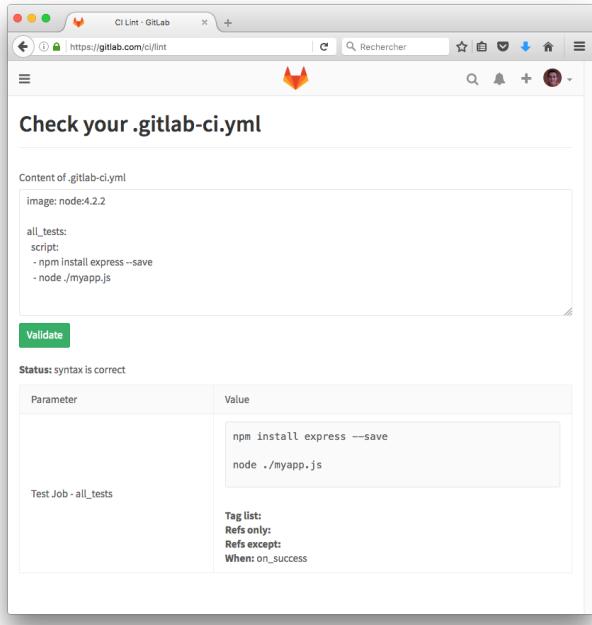


Figure 12. Check of YAML syntax

5. GitHub CI = Actions

5.1. Click and install

Choose a workflow template

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow template to get started.

Skip this and [set up a workflow yourself →](#)

Workflows made for your C# repository Suggested

.NET

By GitHub Actions

Build and test a .NET or ASP.NET Core project.

[Set up this workflow](#)

```
dotnet restore  
dotnet build --no-restore  
dotnet test --no-build --verbosity normal
```

 actions/starter-workflows

C# 

.NET Desktop

By GitHub Actions

Build, test, sign and publish a desktop application built on .NET.

[Set up this workflow](#)

```
dotnet test  
msbuild $env:Solution_Name /t:Restore /p:Configuration=$env:Configuration  
$pfx_cert_byte = [System.Convert]::FromBase64String("{{$secrets.Base64_Encoded_Pfx }}")
```

 actions/starter-workflows

C# 

Deploy your code with these popular services

Deploy Node.js to Azure Web App

By Microsoft Azure

Build a Node.js project and deploy it to an Azure Web App.

[Set up this workflow](#)

 actions/starter-workflows

Deployment 

Deploy to Alibaba Cloud ACK

By Alibaba Cloud

Deploy a container to Alibaba Cloud Container Service for Kubernetes (ACK).

[Set up this workflow](#)

 actions/starter-workflows

Deployment 

Deploy to Amazon ECS

By Amazon Web Services

Deploy a container to an Amazon ECS service powered by AWS Fargate or Amazon EC2.

[Set up this workflow](#)

 actions/starter-workflows

Deployment 

Build and Deploy to GKE

By Google Cloud

Build a docker container, publish it to Google Container Registry, and deploy to GKE.

[Set up this workflow](#)

 actions/starter-workflows

Deployment 

Deploy to IBM Cloud Kubernetes Service

By IBM

Build a docker container, publish it to IBM Cloud Container Registry, and deploy to IBM Cloud Kubernetes Service.

[Set up this workflow](#)

 actions/starter-workflows

Deployment 

OpenShift

By Red Hat

Build a Docker-based project and deploy it to OpenShift.

[Set up this workflow](#)

 actions/starter-workflows

Deployment 

Tencent Kubernetes Engine

By Tencent Cloud

This workflow will build a docker container, publish and deploy it to Tencent Kubernetes Engine (TKE).

[Set up this workflow](#)

 actions/starter-workflows

Deployment 

Terraform

By HashiCorp

Set up Terraform CLI in your GitHub Actions workflow.

[Set up this workflow](#)

 actions/starter-workflows

Deployment 

Continuous integration workflows

Ruby

By GitHub Actions

Build and test a Ruby project with Rake.

[Set up this workflow](#)

 actions/starter-workflows

Ruby 

Rust

By GitHub Actions

Build and test a Rust project with Cargo.

[Set up this workflow](#)

 actions/starter-workflows

Rust 

Java with Ant

By GitHub Actions

Build and test a Java project with Apache Ant.

[Set up this workflow](#)

 actions/starter-workflows

Ant 

Figure 13. Choose your weapon

5.2. Lots of help and documentation

Learn more about GitHub Actions

Getting started and core concepts 

New to Actions? Start here. Learn the core concepts and how to get started.

Configuring and managing workflows 

Create custom workflows to control your project's life cycle processes.

Language and framework guides 

Guides for projects written in many programming languages.

Figure 14. Tune your actions

5.3. Where to check/tune?

.github/workflows/

6. 2021 (awesome!) examples



```
|── build.yml ①
|── doxygen.yml ②
|── linter.yml ③
|── tests.yml ④
|── uml.yml ⑤
```

- ① Build the app
- ② Generate documentation
- ③ Check code quality
- ④ Launch tests
- ⑤ Generate UML diagrams from code

```

Test Suites: 2 passed, 2 total
Tests: 8 passed, 8 total
Snapshots: 5 passed, 5 total
Time: 9.394 s

```

Figure 15. Tests through CI



Figure 16. Easy results checking through badges

The screenshot displays the Swagger UI for the 'Snippets API v3'. At the top, it shows the base URL: <https://project-les-frais.herokuapp.com/>. Below this, there's a dropdown for 'Schemes' set to 'HTTPS', and a user session with 'Django maxkors2014@gmail.com' and 'Authorize' buttons. A 'Filter by tag' input field is also present. The main area lists API endpoints under the 'api' category:

- GET /api/v1/categories/** (blue button) - api_v1_categories_list (green lock icon)
- POST /api/v1/categories/** (green button) - api_v1_categories_create (green lock icon)
- GET /api/v1/categories/{id}/** (blue button) - api_v1_categories_read (green lock icon)
- PUT /api/v1/categories/{id}/** (orange button) - api_v1_categories_update (green lock icon)
- PATCH /api/v1/categories/{id}/** (green button) - api_v1_categories_partial_update (green lock icon)
- DELETE /api/v1/categories/{id}/** (red button) - api_v1_categories_delete (red lock icon)
- GET /api/v1/currencies/** (blue button) - api_v1_currencies_list (green lock icon)

Figure 17. API documentation generation using [Swagger](#)

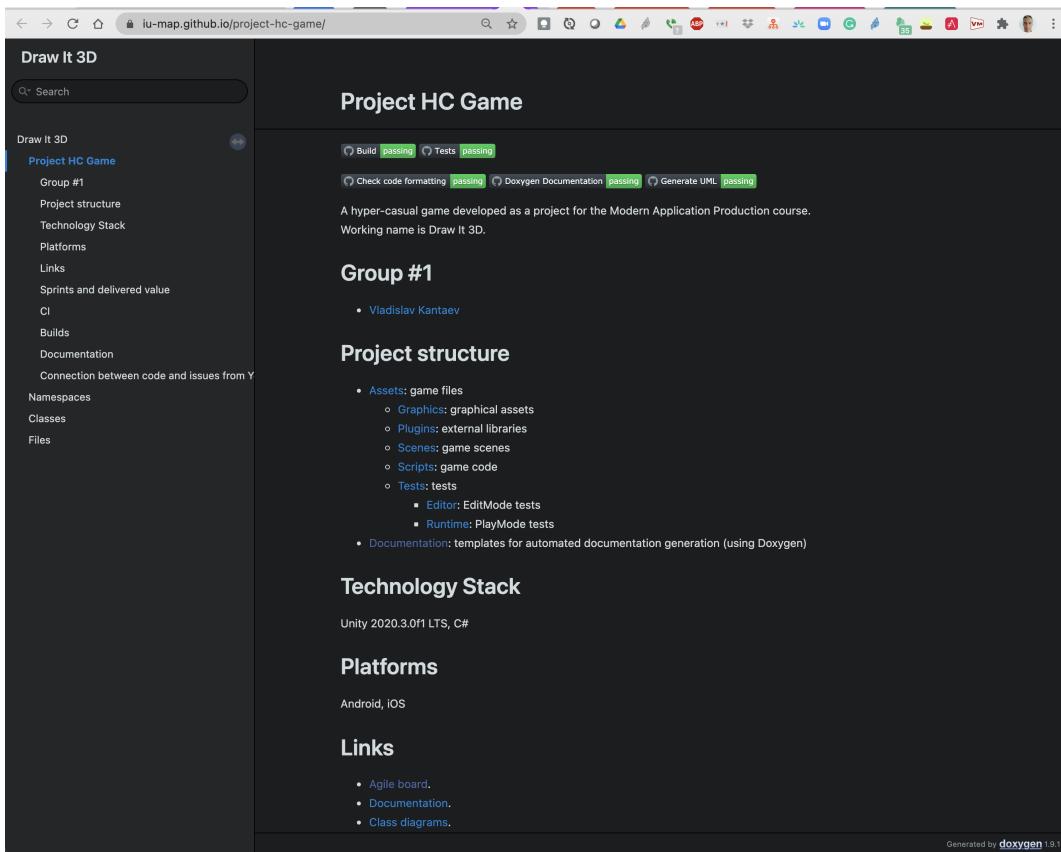


Figure 18. GitHub pages generation

7. Useful links

CI for Gitlab

[Gitlab CI](#)

CI for GitHub

[Actions](#)