cloudbees UNIVERSITY

Intro to Plugin Development

Table of Contents

Introduction

These lab exercises should be completed in sequence.

If you encounter problems with your software installation or if you do not understand any of the instructions, please ask your instructor for help.

Getting Started (Exercise)

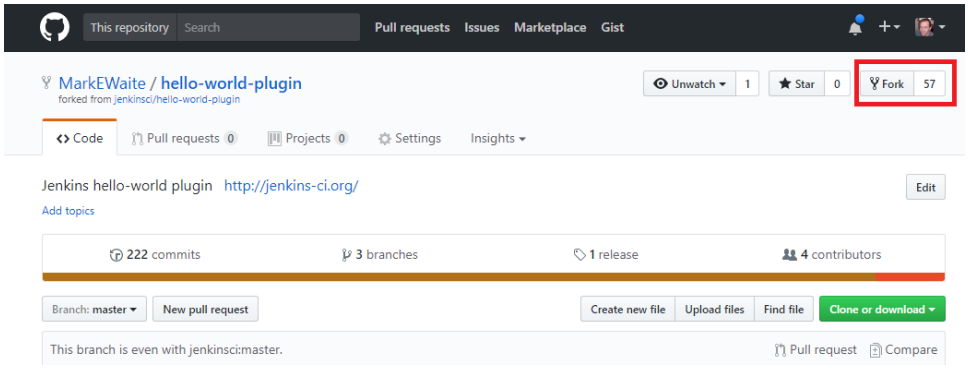
Pre-requisites

- Java Development Kit (JDK) 8
- Apache Maven 3.3.9.0 or later
- Git 1.9 or later
- Integrated Development Environment (IDE) (Netbeans or IntelliJ preferred) with Jenkins plugins (if available)
  - Netbeans users install stapler/jenkins plugin for Netbeans
- GitHub account

Create the project - fork

A GitHub fork is a copy of a repository that is placed into your personal area. It is yours to modify, alter, damage, or destroy as you see fit. Changes to your forked copy of the repository will not appear in anyone else's repository unless they include them in their repository. That is usually done with a "pull request" which you submit, asking them to include changes from you.

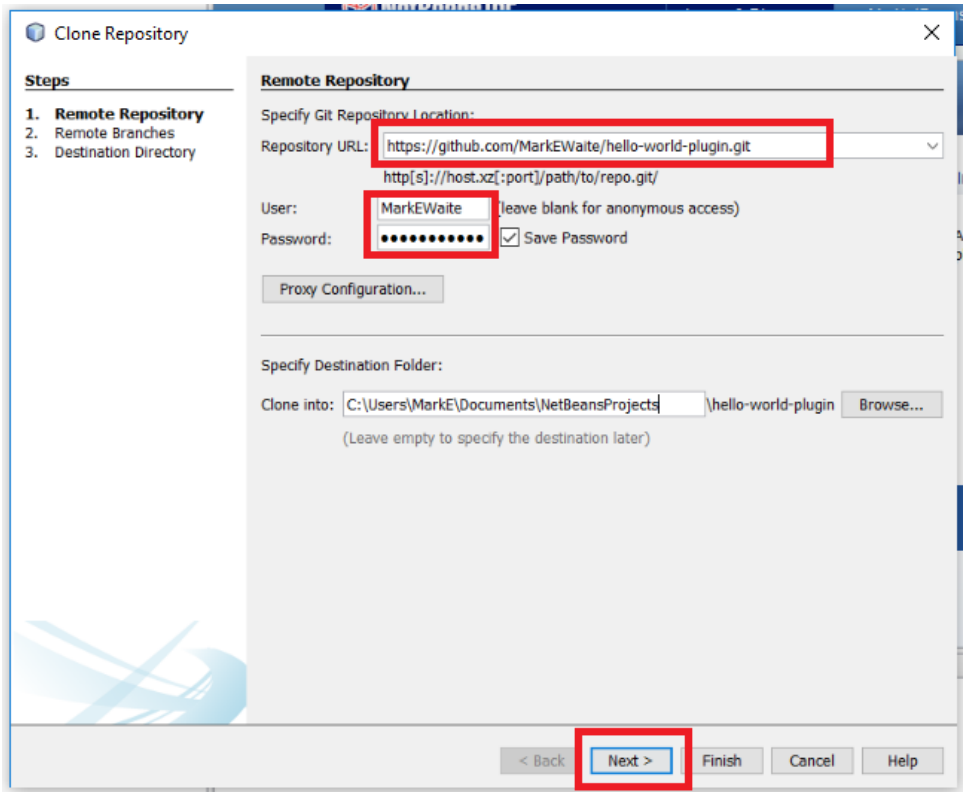
- Fork hello world plugin on GitHub



Create the project - clone

A local clone is where you will make changes and commit those changes. Changes on a local clone are not visible to anyone until they are pushed to a publicly visible location.

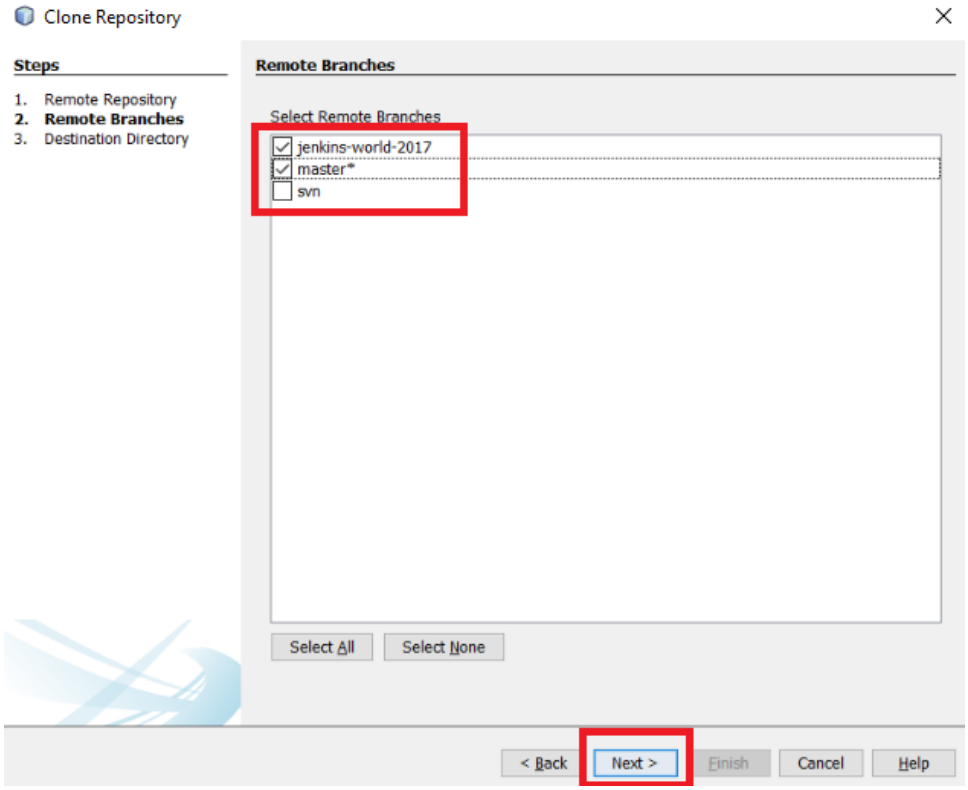
- Clone your fork locally



Create the project - choose branches

Netbeans allows you to select the branches to clone. Be sure you include `jenkins-world-2017` as one of the branches you clone.

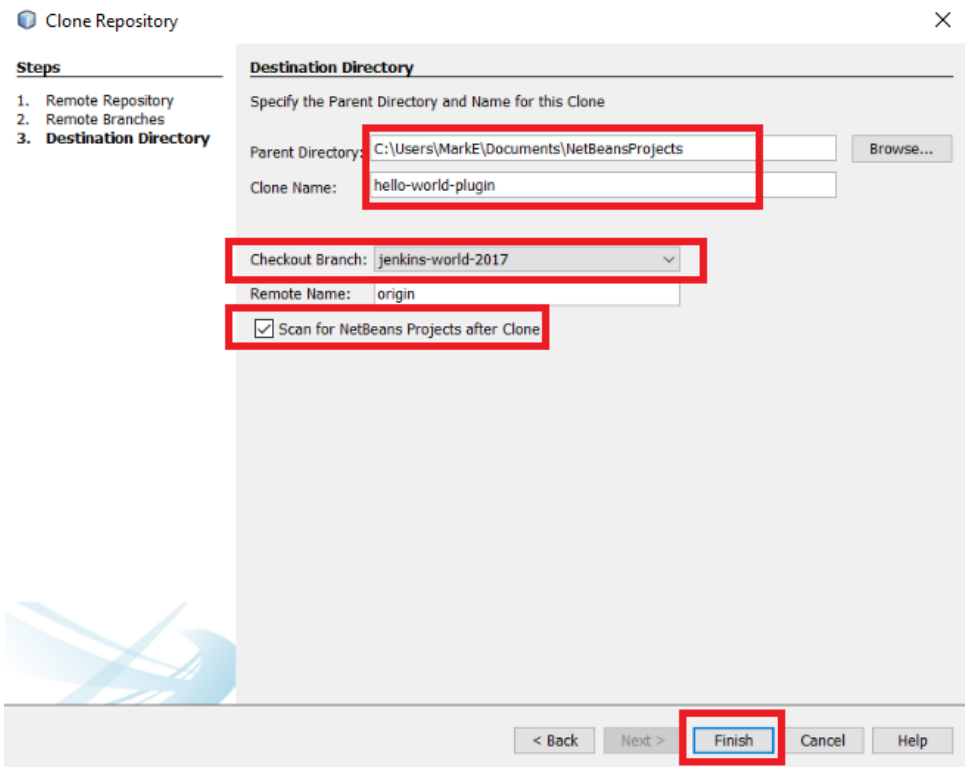
- Choose the branches



Create the project - choose destination

Netbeans allows you to select the destination directory where your project will be stored. Choose a directory you'll remember.

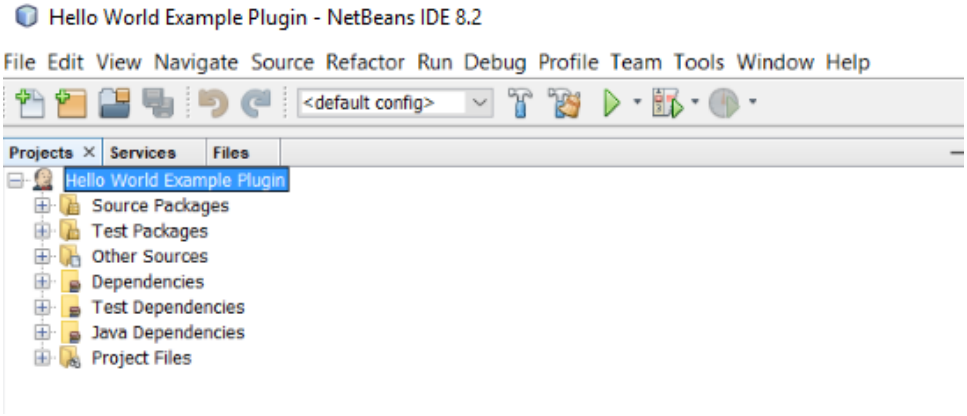
- Choose destination directory



Create the project - done

Once those steps are complete, the project is on your local computer, ready for development. You have a git repository locally which will track your history, and you have a GitHub repository which can share your work with others.

- Project is ready



Compile plugin

- IDE (Netbeans - **Run › Build Project** or press `F11`)
- Command line

```
mvn clean compile
```

Run plugin tests

- IDE (Netbeans - **Run › Test Project** or press `Alt+F6`)
- Command line

```
mvn clean test
```

Show your work

- Show your results to a neighbor

More Information - Getting Started

- Using NetBeans, IntelliJ, Eclipse, or Eclipse with less maven for Jenkins plugin development
- Jenkins.io plugin tutorial, Jenkins wiki custom build step plugin tutorial
- Maven hpi plugin

- Git command line, and graphical interfaces
- GitHub help including pull requests, reviews, and merges
- Jenkins groovy script console
- Jenkins internet relay chat (IRC), users mailing list and developers mailing list

Go back to slides

Build and Run (Exercise)

Verify HelloWorldBuilder in Pipeline

- Run Jenkins
- Install the Pipeline plugin (**Manage Jenkins** › **Manage Plugins** › **Available** › **Pipeline**), then restart Jenkins
- Create a new Pipeline project (**New Item** › **Pipeline**, enter the name and press [**OK**])
- Enter Pipeline definition

```
pipeline {
    agent any
    stages {
        stage('Stage 1') {
            steps {
                echo 'Stage 1 started'
                step([$class: 'HelloWorldBuilder', name: 'Your name'])
                helloWorld 'another name'
                echo 'Stage 1 ending'
            }
        }
    }
}
```

- Execute Pipeline, confirm console output includes "Say hello"

Verify HelloWorldBuilder in FreeStyle

- Run Jenkins
- Open http://localhost:8080/jenkins/
- Create a new Freestyle Project with "New Item"
  - Add the "Say Hello World" build step to the project
  - Save the project
  - Run the project
  - Confirm "Hello world" is in the console output
- Stop Jenkins

Create RootAction

- Create a new IntroRootAction class in your IDE that implements RootAction

```
@Extension
public class IntroRootAction implements RootAction {
    /** Returns icon file name.
      * @return icon file name
      */
    @Override
    public final String getIconFileName() {
        return "clipboard.png";
    }
    /** Returns user visible link text.
      * @return link display name
      */
    @Override
    public final String getDisplayName() {
        return "Intro Root Action";
    }
    /** Returns link destination URL.
      * @return link destination URL
      */
    @Override
    public final String getUrlName() {
        return "https://jenkins.io/";
    }
}
```

- Add the required methods

Verify RootAction

- Compile and run the plugin
- Confirm your new RootAction is visible

Show your work

- Show your neighbor the results of your work

More Information - Build and Run

- Jenkins Action and its Subtypes blog
- Jenkins list of all extensions, and core extensions
- Jenkins javadoc for core, Action, RootAction, Builder, Notifier, Publisher, Recorder, and Shell

Go back to slides

Collect User Input (Exercise)

This exercise shows how to make the plug-in gather more input from the user when the user defines the build step.

- Add a field to HelloWorldBuilder with @DataBoundSetter
- Build the plugin
- Confirm that the new field is in the project configuration page
- Confirm that values assigned to the field are preserved
- Explore out of bounds values (non-numeric values like "half" or "one" or "soon", negative values, floating point values)

Add a field

- Add a new field `long sleepTime` to the HelloWorldBuilder class.

```
/** Sleep duration in milliseconds */
private long sleepTime = 0;
```

- Sleep and log that time in the `perform` method

```
listener.getLogger().println("Sleeping " + (sleepTime / 1000.0) + " seconds");
Thread.sleep(sleepTime);
listener.getLogger().println("Awake after " + (sleepTime / 1000.0) + "
seconds");
```

- Use the IDE to create getters and setters for the field

```
/** Return sleep time in milliseconds.
 * @return sleep time in milliseconds
 */
public final long getSleepTime() {
    return sleepTime;
}
/** Set sleep time in milliseconds.
 * @param sleepTime duration of sleep in milliseconds
 */
public void setSleepTime(final long sleepTime) {
    this.sleepTime = sleepTime;
}
```

- Annotate the setter with @DataBoundSetter

```
/** Return sleep time in milliseconds.
 * @return sleep time in milliseconds
 */
public final long getSleepTime() {
    return sleepTime;
}
/** Set sleep time in milliseconds.
 * @param sleepTime duration of sleep in milliseconds
 */
@DataBoundSetter
public void setSleepTime(final long sleepTime) {
    this.sleepTime = sleepTime;
}
```

Add a UI

- Add a new Jelly snippet for the field

```
<f:entry title="Sleep time" field="sleepTime">
  <f:textbox />
</f:entry>
```

- Add a help page for the field

```
<div>
  Number of milliseconds the job should wait during this build step.
</div>
```

Test the field

- Compile the modified class
- Run Jenkins from the IDE using the modified class
- Test the field
  - Allowed values
  - Illegal values
  - Nonsense values
  - Help text
- Note the problems you found while testing the field

Commit the changes

- Save the tested changes to git (Netbeans **Team › Commit**, then describe the change and press **[OK]**)

```
git add .
git commit -a -m "Added sleep time field"
```

More Information - Collect User Input

- Kohsuke's 2013 DataBoundSetter announcement
- See HelloWorldBuilder for "global.jelly"
- Guide to Jelly, Understanding Jelly, Jelly form controls, Jelly taglib reference
- See UI samples plugin for examples of bars, boxes, buttons, lists, notifications, and syntax highlighting

Go back to slides

Check User Input (Exercise)

Descriptors do checks

- doCheck method checks input
  - Takes @QueryParameter value argument
  - `FormValidation.ok()` accepts input
  - `FormValidation.warning("")` accepts input with warning message
  - `FormValidation.error("")` rejects input

Require integers

- Modify doCheckSleepTime in HelloWorldBuilder
  - Reject out of range characters with `FormValidation.error()`

```
/** Return ok(), warn(), or error() based on user input.
 * @param value user input to be validated
 * @return FormValidation matching user input
 */
public FormValidation doCheckSleepTime(@QueryParameter String value)
        throws IOException, ServletException {
    if (value == null || value.isEmpty()) {
        return FormValidation.ok(); // Null accepted
    }
    value = value.trim();           // Remove leading and trailing spaces
    if (value.isEmpty()) {
        return FormValidation.ok(); // Empty string accepted
    }
    long sleepTime;
    try {
        sleepTime = Long.parseLong(value);
    } catch (NumberFormatException nfe) {
        return FormValidation.error("Sleep time must be an integer");
    }
    return FormValidation.ok();
}
```

- Run it, confirm it works
  - Use the debugger, step through it, confirm that you understand it
  - Change it, try to break it with different input
- Commit to git

Reject negative input

- Add doCheckSleepTime() in HelloWorldBuilder
  - Reject ⍰ 0 input with `FormValidation.warning()`

```
    ...
    } catch (NumberFormatException nfe) {
        return FormValidation.error("Sleep time must be a positive integer");
    }
    if (sleepTime < 0) {
        return FormValidation.error("Sleep time must be a positive integer");
    }
    return FormValidation.ok();
}
```

- Run it, confirm that it works
- Commit to git

Reject really big input

- Modify doCheckSleepTime() in HelloWorldBuilder
  - Reject > 10 minutes input with `FormValidation.error()`

```
    ...
    } catch (NumberFormatException nfe) {
        return FormValidation.error("Sleep time must be a positive integer");
    }
    if (sleepTime < 0) {
        return FormValidation.error("Sleep time must be a positive integer");
    }
    if (sleepTime >= 10 * 60 * 1000) {
        return FormValidation.error("Sleep time must be less than 10 minutes");
    }
    return FormValidation.ok();
}
```

- Run it, confirm that it works
- Commit to git

Warn on big input

- Modify doCheckSleepTime() in HelloWorldBuilder
  - Reject > 1 minute input with `FormValidation.warning()`

```
    ...
    } catch (NumberFormatException nfe) {
        return FormValidation.error("Sleep time must be a positive integer");
    }
    if (sleepTime < 0) {
        return FormValidation.error("Sleep time must be a positive integer");
    }
    if (sleepTime >= 10 * 60 * 1000) {
        return FormValidation.error("Sleep time must be less than 10 minutes");
    }
    if (sleepTime >=      60 * 1000) {
        return FormValidation.error("Sleep time should be less than 1 minute");
    }
    return FormValidation.ok();
}
```

- Run it, confirm that it works
- Commit to git

More information - Check User Input

- Jenkins form validation

- Guide to Jelly, Understanding Jelly, Jelly form controls, Jelly taglib reference
- See UI samples plugin for examples of bars, boxes, buttons, lists, notifications, and syntax highlighting

Go back to slides

Unit Testing (Exercise)

Create a unit test (IDE)

- Netbeans **Tools › Create/Update Tests** generates unit tests for the current Java class
- Generated tests are stubs
- Disable all the tests initially (`@Test` → `// @Test`)

```
// @Test
```

- Update one of the tests

```
private String builderName = "Builder name";
private HelloWorldBuilder builder = new HelloWorldBuilder(builderName);

@Test
public void testGetName() {
    assertThat(builder.getName(), is(builderName));
}
```

- Run the test

Fail a unit test

- Enable testGetDescriptor()

```
private String builderName = "Builder name";
private HelloWorldBuilder builder = new HelloWorldBuilder(builderName);

@Test
public void testGetDescriptor() {
    assertThat(builder.getName(), notNullValue());
}
```

- Run it, watch it fail with null pointer exception
- Use debugger to find null pointer exception

```
// Inside Builder.java
public Descriptor<Builder> getDescriptor() {
    return Jenkins.getInstance().getDescriptorOrDie(getClass());
}
```

Show someone the result

More information - Unit Testing

- Unit testing with Jenkins
- Assertions with AssertJ or Hamcrest
- Manage temporary folders with TemporaryFolder
- Assert exceptions with ExpectedException
- Skipping tests with Assume and @Ignore
- Timeout on tests with Timeout and DisableOnDebug
- Parameterized tests with Parameterized
- Mock objects with mockito and powermock

== Go back to slides

Better Testing (Exercise)

Testing expected exception

- Add unit tests for perform() and getDescriptor()

```
@Rule
public ExpectedException thrown = ExpectedException.none();

private String builderName = "My builder name";
private HelloWorldBuilder builder = new HelloWorldBuilder(builderName);

@Test
public void testPerform() {
    Run build = null;
    FilePath workspace = null;
    Launcher launcher = null;
    TaskListener listener = null;
    // perform() will throw null pointer exception
    // thrown.expect(NullPointerException.class);
    builder.perform(build, workspace, launcher, listener);
}

@Test
public void testGetDescriptor() {
    // getDescriptor() with throw null pointer exception
    // thrown.expect(NullPointerException.class);
    Descriptor builderDescriptor = BUILDER.getDescriptor();
    assertThat(BUILDER.getDescriptor(), notNullValue());
    assertThat(BUILDER.getDescriptor(),
IsInstanceOf.instanceOf(BuildStepDescriptor.class));
}
```

- Confirm tests fail with null pointer exceptions

Add a JenkinsRule

- Add a JenkinsRule

```
@Rule
public ExpectedException thrown = ExpectedException.none();

@Rule
public JenkinsRule jenkins = new JenkinsRule();

private String builderName = "My builder name";
private HelloWorldBuilder builder = new HelloWorldBuilder(builderName);

@Test
public void testPerform() {
    Run build = null;
    FilePath workspace = null;
    Launcher launcher = null;
    TaskListener listener = null;
    // perform() will throw null pointer exception
    thrown.expect(NullPointerException.class);
    builder.perform(build, workspace, launcher, listener);
}

@Test
public void testGetDescriptor() {
    // thrown.expect(NullPointerException.class);
    Descriptor builderDescriptor = BUILDER.getDescriptor();
    assertThat(BUILDER.getDescriptor(), notNullValue());
    assertThat(BUILDER.getDescriptor(),
IsInstanceOf.instanceOf(BuildStepDescriptor.class));
}
```

- Test output increased
- Test runtime increased
- No null pointer exception in testGetDescriptor
- Still a null pointer exception in testPerform

Test Declarative Pipeline

```
@Test
public void testDeclarativePipeline() throws Exception {
    String agentLabel = "my-agent";
    jenkins.createOnlineSlave(Label.get(agentLabel));
    WorkflowJob job = jenkins.createProject(WorkflowJob.class, "test-perform-
pipeline");
    String pipelineScript
            = "pipeline {\n"
            + "    agent any\n"
            + "    stages {\n"
            + "        stage('Stage 1') {\n"
            + "            steps {\n"
            + "                helloWorld 'My Name'\n"
            + "            }\n"
            + "        }\n"
            + "    }\n"
            + "}\n";
    job.setDefinition(new CpsFlowDefinition(pipelineScript, true));
    WorkflowRun completedBuild = jenkins.assertBuildStatusSuccess(job
.scheduleBuild2(0));
    String expectedString = "Hello, " + name + "!";
    jenkins.assertLogContains(expectedString, completedBuild);
}
```

- Test output increased
- Test runtime increased
- Declarative pipeline now tested

Test Scripted Pipeline

```
@Test
public void testScriptedPipeline() throws Exception {
    String agentLabel = "my-agent";
    jenkins.createOnlineSlave(Label.get(agentLabel));
    WorkflowJob job = jenkins.createProject(WorkflowJob.class, "test-perform-
pipeline");
    String pipelineScript
            = "node {\n"
            + "  helloWorld '" + name + "'\n"
            + "}";
    job.setDefinition(new CpsFlowDefinition(pipelineScript, true));
    WorkflowRun completedBuild = jenkins.assertBuildStatusSuccess(job
.scheduleBuild2(0));
    String expectedString = "Hello, " + name + "!";
    jenkins.assertLogContains(expectedString, completedBuild);
}
```

- Test output increased
- Test runtime increased
- Scripted pipeline now tested

Test FreeStyle Project

```
@Test
public void testFreeStyleProject() throws Exception {
    FreeStyleProject project = jenkins.createFreeStyleProject();
    project.getBuildersList().add(builder);
    FreeStyleBuild completedBuild = jenkins.assertBuildStatusSuccess(project
.scheduleBuild2(0));
    String helloString = "Hello, " + name + "!";
    jenkins.assertLogContains(helloString, completedBuild);
}
```

- Test output increased
- Test runtime increased
- Scripted pipeline now tested

More Information - Better Testing

- Javadoc for JenkinsRule,
- SauceLabs beyond unit testing blog posting

More Information - Conclusion

- Other tutorials, writing your third plugin, Extending Jenkins - 2015
- Logger console
- Acceptance test harness and How to write an acceptance test - 2014
- Credentials plugin and its user guide, consumer guide, and implementation guide
- Variable expansion on stackoverflow, and in Pipeline
- Internationalization and localization
- FindBugs in Plugins
- Plugin upgrades retaining backward compatibility
- Extending the remote access API from a plugin API
- Pipeline book, developers guide
- Project types (FreestyleProject, MatrixProject, etc.)
- Alistair Todd's plugin tutorial, part 1, part 2, part 3, and part 4

Go back to slides