Rational Build Forge

# Tutorial: Getting started with Rational Build Forge

*Version 7.1.3*

**Note**
Before using this information and the product it supports, read the information in "Notices," on page 33.

This edition applies to version 7.1.3 of Rational Build Forge and to all subsequent releases and modifications until otherwise indicated in new editions.

# Contents

# Chapter 1. Tutorial 1: Creating, running, and scheduling a Rational Build Forge project

Use this tutorial to learn how to create a basic "Hello World" project in IBM® Rational® Build Forge®.

The tutorial shows you how to do the following tasks:

- Create a server authentication
- Create a selector and server
- Create a project
- Run a project
- Schedule a project

Time required: 30 minutes

## Creating a server authentication

When a server is authenticated, the same login information can be used for running Rational Build Forge jobs from more than one computer.

1. Start Rational Build Forge and log in.
2. Click the **Console** tab.
3. In the menu on the left, click the **Server** and then click **Server Auth**.
4. In the **Name** field, type `localhost auth` or a name of your choice.
5. In the **Access** list, set an access level. For more information, see Chapter 3, “Tutorial 3: Rational Build Forge administration: Controlling user access,” on page 9.
6. Enter the login information for this computer. This information is the credentials you normally use to log in to the computer.
7. Click **Save Server Authentication**. The login information is saved so that a project can later be run on other computers by using the same login information.

## Creating a selector and server

Specifying a selector allows you to pick the appropriate server on which to run your project. It can specify a server by name or by a property that a collector collects and stores in the manifest.

1. In the menu on the left, click **Server** and then click **Selectors**.
2. In the **Name** field, type `build selector` or a name of your choice.
3. Click **Save**.
4. In the **Name** field, type `BF_NAME`.
5. Set **Required** to **Yes**.
6. Set **Operator** to **EQ** (equals).
7. Set **Value** to **build_server**.
8. Click **Save**.

You have specified that a server with the name BF_NAME can be used as a build server.

## Creating a project

A project is a set of command-line commands that is run on the selected servers.

The only difference between a library (see Chapter 2, “Tutorial 2: Creating and using Rational Build Forge libraries,” on page 5) and a project is that a project uses a selector to find the correct server on which to run the commands.

1. In the menu on the left, click**Projects**.
2. In the **Name** field, type "Hello World".
3. In the **Selector** list, select the build selector you created in "Creating a selector and server." Selectors are used for matching the configuration information, which is found using a Collector, of a particular server with the information specified by you. In this example, you are matching any server with the BF_NAME of build_server.
4. In the rest of the fields, leave the default settings.
5. Click **Save**.
6. On the Details page for **Add New Step**, in the **Name** field, type a name. For example, type "echo".
7. In the **Command** field, type "echo Hello World".
8. Click **Save Step**.

You have created a new project named Hello World that specifies a step named echo. Each project holds a collection of steps. A project uses a certain selector to decide on which servers it should be run. Steps are one or more system commands that run in the shell of the selected servers.

## Running the project

1. In the menu on the left, click **Projects**.
2. Next to Hello World, click the blue Play button. In the yellow bar at the top, a message is displayed: "Job Build_1 Started." The build number changes depending on how many times you have run the project. The first time is Build_1, the second Build_2, and so on.
3. Select the **Jobs** tab. The Hello World project e build was successful.
4. To see more details about the execution of the build, click on the build name ("Build_X", where x is the build number) and then select a step.

## Scheduling a project

With Rational Build Forge, you can schedule your jobs to run at regular intervals, such as every day or every week at a particular time. Suppose you want to run this program every day at 1:30 p.m. To do so, use the Rational Build Forge scheduling capability.

1. In the menu on the left, select **Schedules**.
2. In the **Description** field type a name, such as "Hello World Daily Scheduler".
3. In the **Project** list, select the "Hello World" project that you created earlier.
4. In the **Selector** list, select the "build selector" selector that you created earlier.
5. In the **Minutes** field, type "30".
6. In the **Hours** field, type "13". The entry must be a value in 24 hour time.

7. Click **Save Schedule**.

**Tip:** To see a visual representation of when your scheduled projects will run, click the **Calendar** tab.

A number on each date on the calendar indicates the number of projects running on that day. If you move your mouse over this number, a list of the projects scheduled for that day is shown.

## Tutorial summary

Now that you have completed this tutorial, you can:

- Create selectors that determine which servers run your Rational Build Forge projects.
- Create a Rational Build Forge project with a specific collection of steps.
- Schedule the projects to run at regular intervals.

# Chapter 2. Tutorial 2: Creating and using Rational Build Forge libraries

A *library* is an IBM Rational Build Forge project that does not specify a selector. Libraries are useful for extracting repeated code so that the code does not have to be reentered as a step or series of steps in each project. Instead, the project can simply utilize the library.

The tutorial shows you how to do the following tasks:

- Create a library
- Copy a project
- Inline a library
- Use a library as part of a pass/fail chain

Time required: 30 minutes

## Creating a library

Begin by creating a library.

1. Start Rational Build Forge and log in.
2. In menu on the left, click **Libraries**.
3. Click **Add Library**.
4. In the **Library: Add Library** pane at bottom, on the Library Details page, in the **Name** field, type `basicLibrary`.
5. Leave the default values for the rest of the fields.
6. Click **Save**.
7. Click **Add Step**.
8. In the **Name** field, type a name for the step. For example, type `echo`.
9. In the **Command** field, type `echo basicLibrary`.
10. Click **Save Step**.

You created a library called basicLibrary with a step named echo. Libraries use the selector of any step that calls them. If a calling step does not have a selector, the library uses the selector of the step's project. Libraries are typically called by other projects as an Inline for a step or as a Pass chain or Fail chain for a step.

## Copying a project

You can copy each project as many times as you like. All commands and settings are copied identically. In a production environment, you might want to copy a project to provide a test bed for making changes to the original project.

1. In the menu on the left, click **Projects**.
2. Next to the "Hello World" project that you created in Chapter 1, “Tutorial 1: Creating, running, and scheduling a Rational Build Forge project,” on page 1, click the edit button  .
3. Click **Copy Project**.

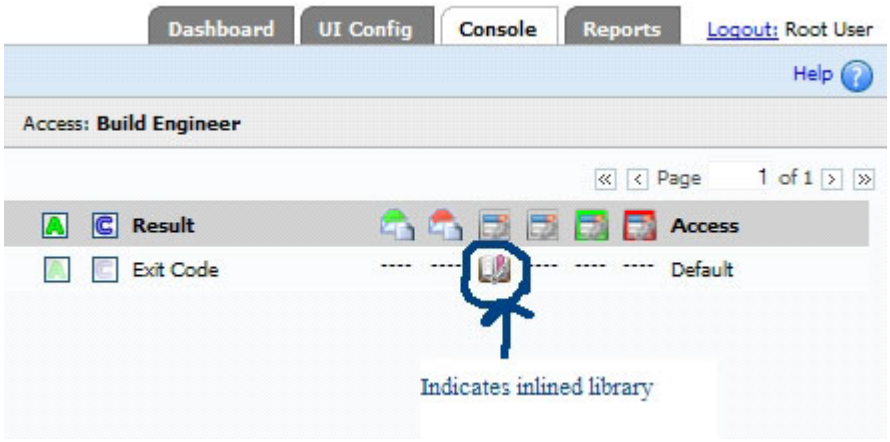
A new project is created, named "Hello World Copy", which is identical to "Hello World."

## Inlining a library

By associating a step with an inlined library, the code in the step **Command** field is run; then the steps in the library run as if they were *inline* with the rest of the project code.

1. Click the edit button next to the Hello World Copy project and then, in the **Name** field, enter a new name. For example, type `Hello World with Library`.
2. Click the **Hello World with Library** project. The Steps page is shown.
3. Click the **echo** step.
4. In the **Inline** field, select the library you created earlier, **basicLibrary**.
5. Click **Save Step**.

You have now inlined a library. The icons on the right side of the page show that a library is being used, as shown in the following screen capture.

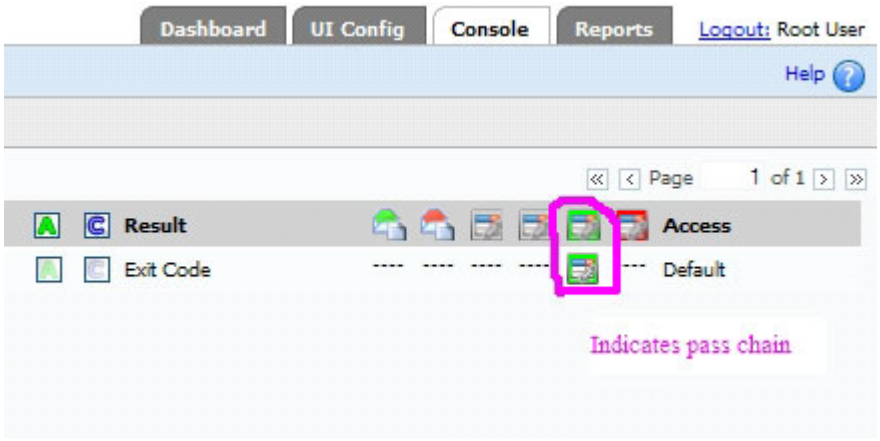



## Using a library as part of a pass or fail chain

If the commands of a particular step run without an error (the commands pass) and if there is a library or project selected under the pass chain, the step will run. Otherwise, if the commands fail, the pass library or project will not run; instead, the library or project selected under the fail chain will run.

1. Repeat the steps to copy a project.
2. Click the new project name, **Hello World with Library Chain**.
3. In the **Pass Chain** field, select **basicLibrary**.

Now the basicLibrary commands will run only if the echo step commands *pass*, which means to run without errors.




**Note:** The **Fail Chain** field is still empty. Also, there is a library/project to run on the pass chain.

## Tutorial summary

Now that you have completed this tutorial, you can create your own libraries and add them to pass or fail chains.

# Chapter 3. Tutorial 3: Rational Build Forge administration: Controlling user access

Users must have permission to view, change, or create objects in the system. Their permissions are controlled through access groups.

- Users belong to one or more access groups
- An access group for access has a set of permissions
- An access group for notification consists only of members

The tutorial shows you how to do the following tasks:

- Learn about the relationship between users, access groups, and permissions
- Learn how to use subgroups to set up hierarchical access levels

Time required: 45 minutes

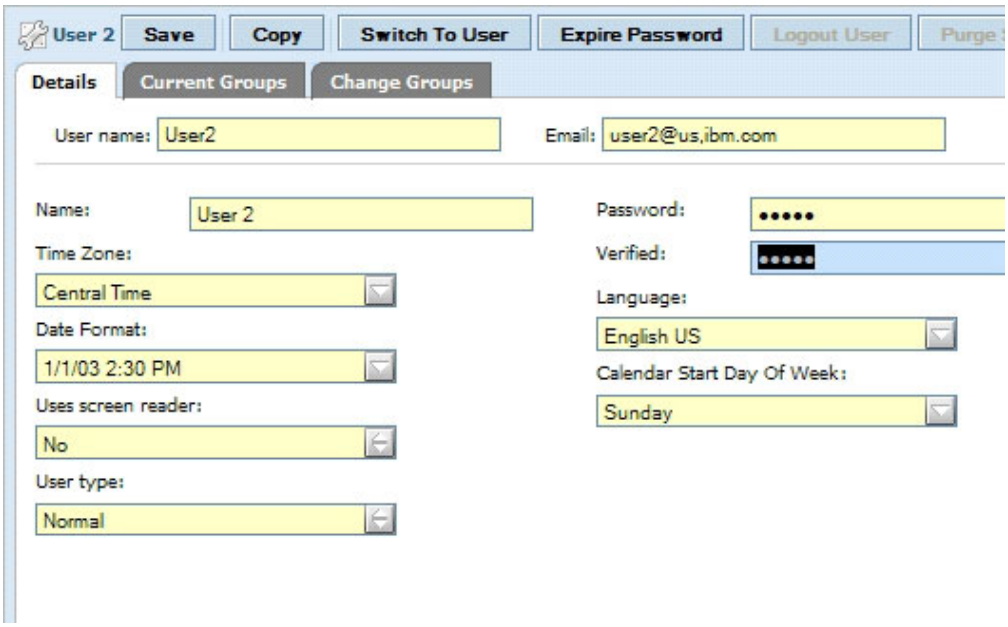
## Creating users and access groups

Begin by creating users and access groups and then assign the users to the access groups.

### 1. Creating a user

1. Start Rational Build Forge and log in.
2. In the menu on the left, click **Administration** > **Users**.
3. In the **User name** field, type `User2`.
4. In the **Email** field, type the email address to be used to send system-generated notifications to the user, for example `user2@mycorp.com`.
5. In the **Name** field, type the given name of the user.
6. In the **Password** and **Verified** fields, type the password.
7. Click **Save**.

The following example image shows a screen capture of the Details page for "User 2."




## 2. Creating an access group for notifications

In this lesson, you create an access group for notifying all users whose names begin with User. You also create an access group called "modify project" to create an access group used exclusively for notification.

First, create the access group for notification:

1. In the menu on the left, click **Administration** > **Access Groups**.
2. Click **Add Group**.
3. In the **Name** field, type `user_notify`.
4. Select the **Default** check box. Any new user will now be a member of this group.
5. In the **Owner** drop-down list, select an access group that can modify group access permissions.
6. Click **Save**.

The following example image shows a screen capture of the Details page for "user_notifiy" access group.

Now, add users to the user_notify group:

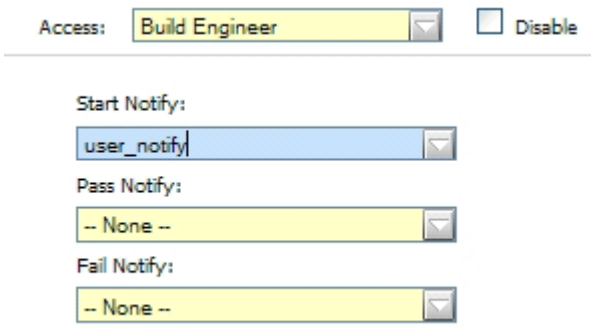
1. In the lower-center pane, click the **Users** tab.
2. Select each user to add and then click **Add**. Because this lesson is about a notification to all users who have names starting with "User", add **User 1** and **User 2**.

   **Tip:** To remove a user, select the user name and then click **Remove**.
3. Click **Save**.

Finally, add the notification group to a project:

1. Click **Projects**.
2. Click the edit button next to a project. For example, edit Hello World.
3. At the lower right of the Project Details page, assign the notification group you created earlier to one of the notification properties. You have three options for notification, represented by the **Start Notify**, **Pass Notify**, and **Fail Notify** fields, in which you can specify the user group:
   - **Start Notify**: Notifies the selected group when the project starts.
   - **Pass Notify**: Notifies the selected group when the project runs without errors.
   - **Fail Notify**: Notifies the selected group when the project fails with errors.

   In the following example image, the user_notify group is assigned to **Start Notify**.




## 3. Creating an access group for editing projects

An editing access group is similar to a notification access group, but includes additional sets of permissions.

1. Create an access group named "modify projects" by repeating steps 1 on page 10 through 6 on page 10 in the first part of "Creating an access group" above.
2. Add User2 to the modify projects group by repeating steps 1 through 3 in the second part of "Creating an access group" above.
3. In the menu on the left, click **Permissions**.
4. In the **Permissions** pane, click the **AddProject** permission.
5. On the Details page, select the **modify projects** group and then click **Add**. You can also add any other necessary permissions.

## Using subgroups to set up hierarchical access

In this lesson, consider an example software product called Widget. There are groups of users who have different roles in working with the project that builds Widget.

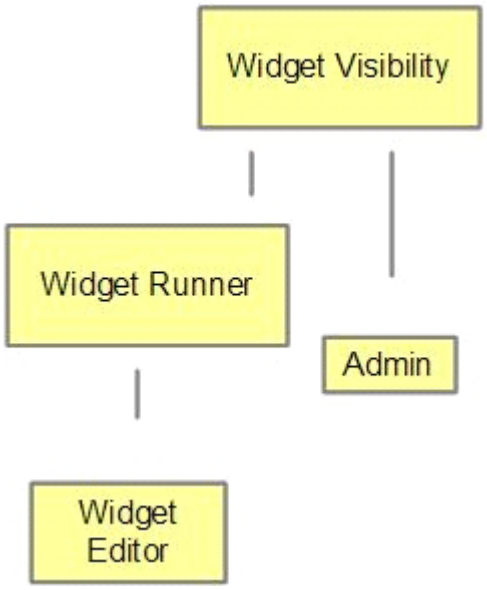
- There are users who should be able to run projects (for example, tests) but should not be able to edit them. You will call this group "Widget Runner".
- There are users who should be able to both run and edit projects. You will call this group "Widget Editor".
- There are users who are administrators who control which users are in which groups. Administrators do not need to run or edit projects. They only need to be able to view all users and projects, edit group membership, and do other administrative tasks. You will call this group "Widget Administrator".

You can create a hierarchy of access groups to represent these tiers of access permissions. The hierarchy passes permissions downward. A subgroup inherits all permissions allowed by the group that contains it. Therefore, the top-level container group has the least permissions. Each subgroup adds permissions.

This is the hierarchy of access groups needed to reflect the tiered access requirements stated above:

- Widget Visibility: the top-level access group. No users are assigned to it. It contains permissions to view objects that are needed by all subgroups.
- Widget Administrators: a subgroup of the Widget Visibility group. It contains permissions to define groups and perform other administrative actions.
- Widget Runner: another subgroup of the Widget Visibility access group. It contains permissions to run projects. Users who only run projects are assigned to it.
- Widget Editor: a subgroup of the Widget Runner Group. It contains permissions to edit projects. Users who need to both run and edit projects are assigned to it.

The advantage of using subgroups is that you do not have to set all permissions explicitly for each type of access. Subgroups inherit the permissions of the parent groups. Additionally, any changes made to a parent group are automatically inherited by its subgroups.




To create an access group hierarchy:

1. Create the access groups
2. Edit the access groups to set up the subgroup relationships

## 1. Creating access groups

1. Create an access group named "Widget Visibility" with view permissions. Do not add users to the group.
2. Create an access group named "Widget Runner" with run permissions. Add the users who need to run projects to the group.
3. Create an access group named "Widget Editor" with edit permissions. Add the users who need to edit projects.
4. Create an access group named "Widget Administrator" with administrative permissions. Add the users who need to perform administrative tasks.

## 2. Editing access groups to specify subgroup relationships

1. Specify that the Widget Runner and Widget Administrator groups:
   a. Click **Widget Visibility**.
   b. Click the **Subgroups** tab.
   c. Click the **Add** button and add Widget Runner and Widget Administrator as subgroups.
2. Specify that the Widget Editor group is a subgroup of the Widget Runner group:
   a. Click **Widget Runner**.
   b. Click the **Subgroups** tab.
   c. Click the **Add** button and add Widget Editor as a subgroup.

The groups are now arranged into subgroups, providing each group with the proper access permissions.

## Tutorial summary

Now that you have completed this tutorial, you can:

- Create users
- Create access groups with specified permissions and notifications
- Assign users to access groups
- Define inherited levels of permissions by specifying subgroup relationships among groups

# Chapter 4. Tutorial 4: Using log filters in Rational Build Forge for Ant builds

In IBM Rational Build Forge, log filters specify the success criteria for a step by using regular expression matching.

This tutorial utilizes an Ant file. A common use case for log filters is the execution of commands regarding Ant files, because these commands do not always return useful exit codes.

The tutorial shows you how to do the following tasks:

- Create a log filter
- Use a log filter in a project
- Set a project to continue even when a step fails

Time required: 30 minutes

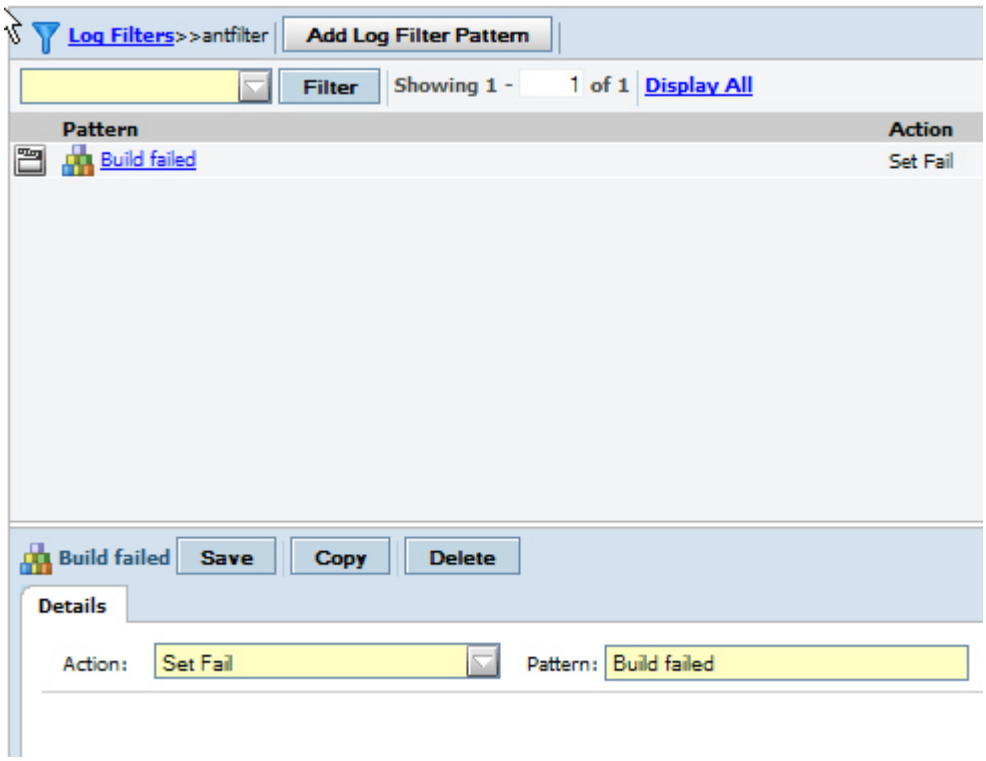
## Creating a log filter

With Ant, the system return code is not always indicative of the success or failure of the build. A log filter is needed to determine whether the step actually worked.

1. Create an Ant file. A sample Ant file named build.xml is included in the e-kit, in the AntSample sample.
2. In the Rational Build Forge Console, in the menu on the left, click **Projects** > **Log Filters**.
3. In the **Name** field, type `antfilter`.
4. Click **Save**.
5. Click **Add Log Filter Pattern**.
6. On the Details page, in the **Action** drop-down list, select **Set Fail**. This means that if the regular expression entered in the Pattern field is found, the step will have the result of fail.
7. In the **Pattern** field, enter `Build failed`. If the Ant build command does not work, these words will be displayed in the log.
8. In the **Notify** list, select a group to notify, for example **Build Engineer**.
9. Click **Save**.

You have created a log filter that notifies the Build Engineer user group when a build fails. In the next step, you will use the log filter in a project.

The following image shows the Build failed pattern in the antfilter log filter.




## Using a log filter in a project

1. In the menu on the left, click **Projects**.
2. Create a project named "anttutorial."
3. Create a step, named "failbuild," and then in the **Command** field, type the following code:

   ```
   ant -f C:temp/buildDNExml compile
   ```

   where *buildDNExml* is a build file that does not exist. Later in the tutorial, when you run the project, you will see the result of this command.
4. Scroll down in the bottom-center pane, beneath the field in which you entered the command.
5. In the **Result** list, select **antfilter**. You might need to scroll down the page, past the **Command** field.

   **Note:** Setting the **Result** field ensures that the Rational Build Forge step uses the log filter named antfilter that you created earlier.
6. Set the **On Fail** field to **Continue**.

   **Note:** Setting the **On Fail** field ensures that the project keeps running and runs the subsequent steps, even if this step fails. You are doing this because you expect this step to fail (since the file buildDNExml does not exist) and you are going to put the working build step as the next step.
7. Create a step, named "passbuild", and then in the **Command** field, type the following code:

   ```
   ant -f <path to buildfile>/build.xml compile
   ```

This command assumes that you have a build.xml file with a compile target.

8. In the **Result** list, select **antfilter**.
9. Run the **anttutorial** project and view the results.


10. Click **failbuild**. At the end of the log, you should see something like the following image:

```
SCRIPT  ant -f /home/emredmil/antFiles/ElissaAntSample/ElissaAntSample/buildDNExml compi
EXEC    start [/home/emredmil/anttutorial/BUILD_6@rightmeow]
EXEC    Buildfile: /home/emredmil/antFiles/ElissaAntSample/ElissaAntSample/buildDNExml c
EXEC    Build failed
EXEC    end [/home/emredmil/anttutorial/BUILD_6@rightmeow]
RESULT  1 (1)
```

11. Click **passbuild**. At the end of the log, you should see something like the following image:

```
SCRIPT  ant -f /home/emredmil/antFiles/ElissaAntSample/ElissaAntSample/build.xml compile
EXEC    start [/home/emredmil/anttutorial/BUILD_6@rightmeow]
EXEC    Buildfile: /home/emredmil/antFiles/ElissaAntSample/ElissaAntSample/build.xml
EXEC
EXEC    init:
EXEC
EXEC    compile:
EXEC        [javac] Compiling 1 source file to /home/emredmil/antFiles/ElissaAntSample/E
EXEC
EXEC    BUILD SUCCESSFUL
EXEC    Total time: 1 second
EXEC    end [/home/emredmil/anttutorial/BUILD_6@rightmeow]
RESULT  0 (0)
```

## Tutorial summary

Now that you have completed this tutorial, you can:

- Create a log filter
- Use a log filter in a project
- Set a project to continue On Fail

# Chapter 5. Tutorial 5: Environment variables in Rational Build Forge

An *environment* is a named set of variables. You can use environment variables in IBM Rational Build Forge to abstract common parts of code, such as paths to binary files, addresses to code repositories, and version numbers.

In this tutorial, you will use environment variables to create two "sentences" using the following command: `echo ${HELLO}, ${WELCOME} build forge`.

In the first environment, the value of the variables will be:

- HELLO = Hello
- WELCOME = welcome to

In the second environment, the value of the variables will be:

- HELLO = Goodbye
- WELCOME = leaving

By changing the environment at project run time, you will see the differences in the output of the same echo command.

The tutorial shows you how to do the following tasks:

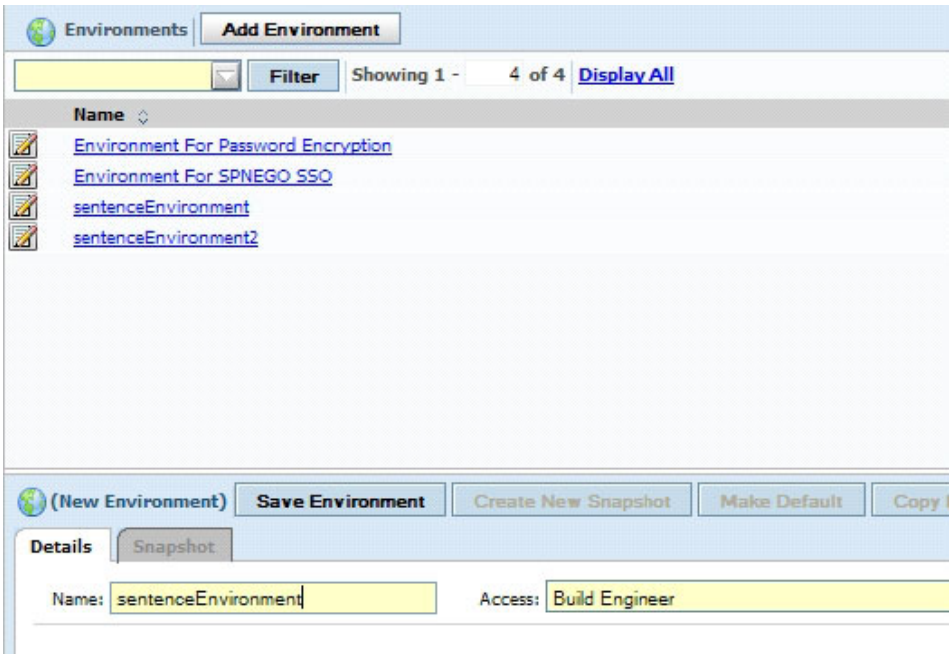
- Create an environment with standard variables
- Run a project with standard environment variables
- Create an environment with pulldown variables
- Run a project with pulldown environment variables

Time required: 20 minutes

## Creating an environment with standard variables

Standard variables are predefined and populated by the system every time it creates the environment for a step.

1. Open the Rational Build Forge Console.
2. In the menu on the left, click **Environments**.
3. Click **Add Environment**.
4. In the **Name** field, type `sentenceEnvironment`; then select the appropriate access group.
5. Click **Save Environment**. The following image shows an example of a newly created environment named sentenceEnvironment.

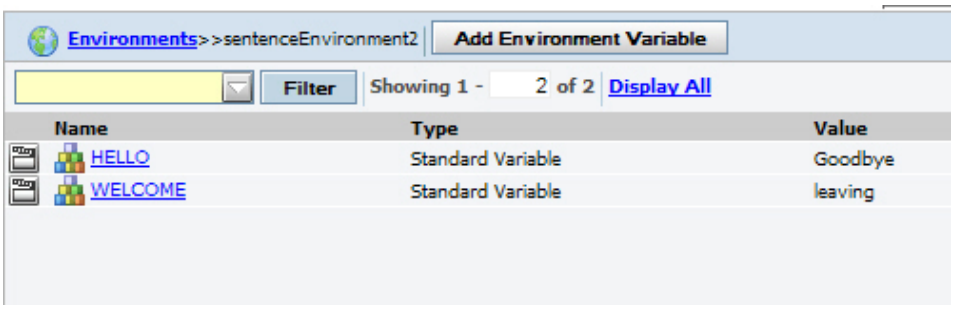



6. Create an environment variable by completing the following steps:
   a. In the **Environments** list, click the edit button next to **sentenceEnviroment**.
   b. Click **Add Environment Variable**.
   c. In **Name**, type `HELLO`.
   d. In **Value**, type `Hello`.
   e. Leave **Type** set as **Standard Variable**.

      **Learn more about variable types:** There are *Include* variables, in which you can include another environment and its variables. There are also *Pulldown* variables, in which you can specify different possible options for each variable.

   f. Click **Save Variable**.
7. Create another environment variable named "WELCOME" by repeating the steps above, this time with the value set to "welcome to".
8. Click **Save Variable**.
9. Click **Environments**.
10. Create another environment, named "sentenceEnvironment2" and save the environment.
11. Create two environment variables, HELLO and WELCOME, with values "Goodbye" and "leaving".

    **Remember:** Save each variable.
    The following image shows a screen capture of the sentenceEnvironment2 variables.



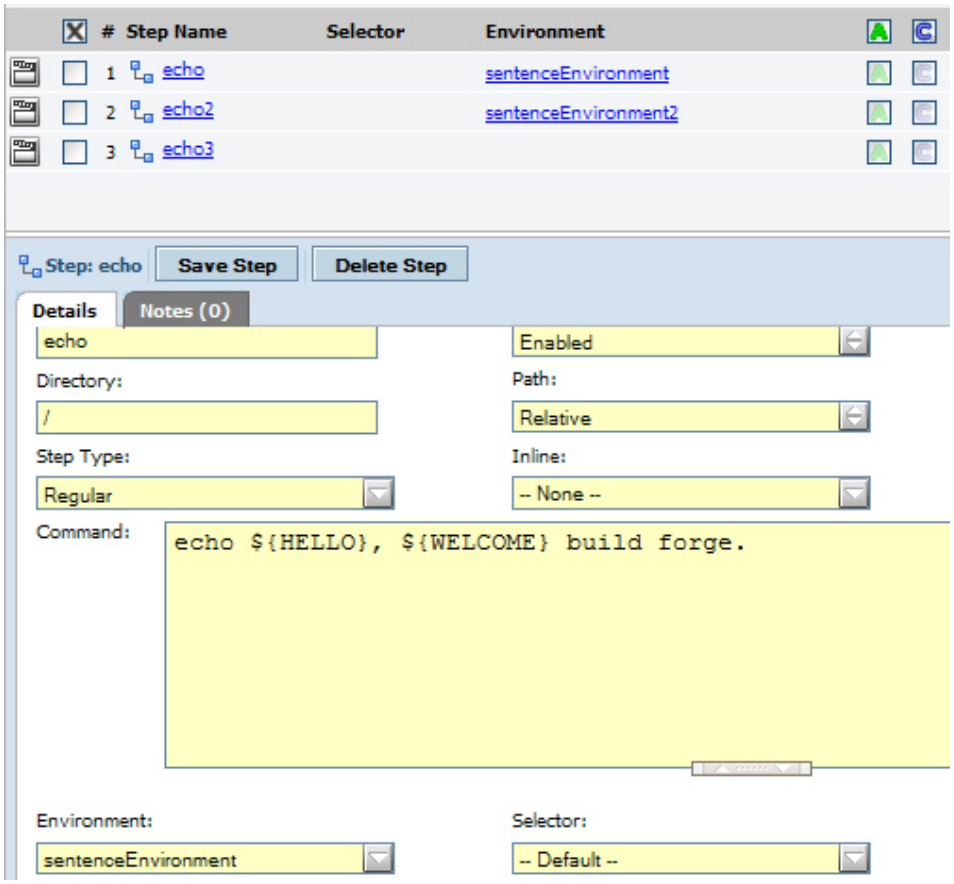

## Running a project with different environment settings

1. In the menu on the left, click **Projects**.
2. Create a project called "environmentSentence."
3. Create a step for the project and call it "echo."
4. On the Details page, in the **Command** field, type the following code:

```
echo ${HELLO}, ${WELCOME} build forge
```

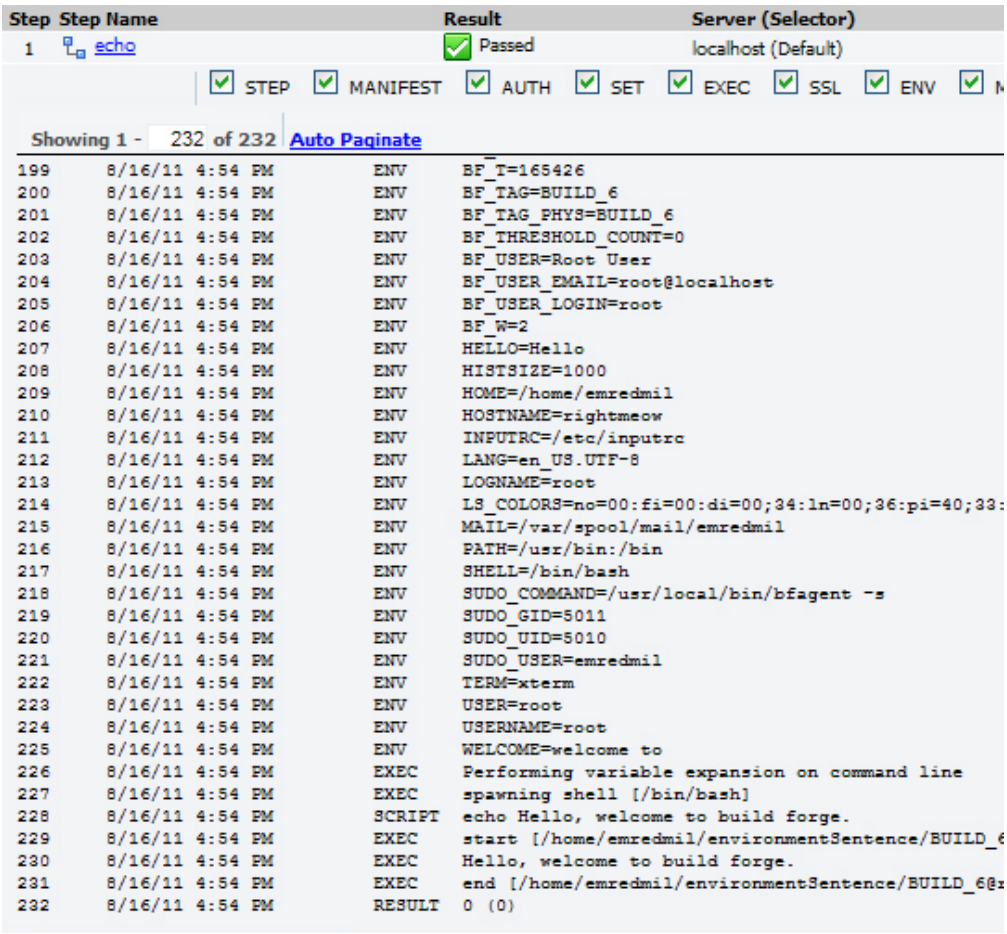
**Note:** You can use either a UNIX-style or Windows-style variable syntax in step commands or environment variables definitions. The system uses a preprocessor to interpret both UNIX-style ($VAR) or Windows-style (%VAR%) syntax into an appropriate format for the server where the step is run. The preparsing can enable a step to run on either a Windows-based server or a UNIX-based server.

The following image shows an example of the settings for the echo step.



5. Click **Save Step**.
6. In the **Environment** list, select **sentenceEnvironment**.
7. Create a new step called “echo2.”
8. On the Details page, in the **Command** field, type the following code:
   `echo ${HELLO}, ${WELCOME} build forge`
9. In the **Environment** list, select **sentenceEnvironment2**.
10. Run the project and view the output.

The following image shows an example of the output for echo.



| Step | Step Name | Result | Server (Selector) |
|---|---|---|---|
| 1 | echo | Passed | localhost (Default) |

STEP MANIFEST AUTH SET EXEC SSL ENV

Showing 1 - 232 of 232 Auto Paginate

```
199     8/16/11 4:54 PM         ENV     BF_T=165426
200     8/16/11 4:54 PM         ENV     BF_TAG=BUILD_6
201     8/16/11 4:54 PM         ENV     BF_TAG_PHYS=BUILD_6
202     8/16/11 4:54 PM         ENV     BF_THRESHOLD_COUNT=0
203     8/16/11 4:54 PM         ENV     BF_USER=Root User
204     8/16/11 4:54 PM         ENV     BF_USER_EMAIL=root@localhost
205     8/16/11 4:54 PM         ENV     BF_USER_LOGIN=root
206     8/16/11 4:54 PM         ENV     BF_W=2
207     8/16/11 4:54 PM         ENV     HELLO=Hello
208     8/16/11 4:54 PM         ENV     HISTSIZE=1000
209     8/16/11 4:54 PM         ENV     HOME=/home/emredmil
210     8/16/11 4:54 PM         ENV     HOSTNAME=rightmeow
211     8/16/11 4:54 PM         ENV     INPUTRC=/etc/inputrc
212     8/16/11 4:54 PM         ENV     LANG=en_US.UTF-8
213     8/16/11 4:54 PM         ENV     LOGNAME=root
214     8/16/11 4:54 PM         ENV     LS_COLORS=no=00:fi=00:di=00;34:ln=00;36:pi=40;33:
215     8/16/11 4:54 PM         ENV     MAIL=/var/spool/mail/emredmil
216     8/16/11 4:54 PM         ENV     PATH=/usr/bin:/bin
217     8/16/11 4:54 PM         ENV     SHELL=/bin/bash
218     8/16/11 4:54 PM         ENV     SUDO_COMMAND=/usr/local/bin/bfagent -s
219     8/16/11 4:54 PM         ENV     SUDO_GID=5011
220     8/16/11 4:54 PM         ENV     SUDO_UID=5010
221     8/16/11 4:54 PM         ENV     SUDO_USER=emredmil
222     8/16/11 4:54 PM         ENV     TERM=xterm
223     8/16/11 4:54 PM         ENV     USER=root
224     8/16/11 4:54 PM         ENV     USERNAME=root
225     8/16/11 4:54 PM         ENV     WELCOME=welcome to
226     8/16/11 4:54 PM         EXEC    Performing variable expansion on command line
227     8/16/11 4:54 PM         EXEC    spawning shell [/bin/bash]
228     8/16/11 4:54 PM         SCRIPT  echo Hello, welcome to build forge.
229     8/16/11 4:54 PM         EXEC    start [/home/emredmil/environmentSentence/BUILD_6
230     8/16/11 4:54 PM         EXEC    Hello, welcome to build forge.
231     8/16/11 4:54 PM         EXEC    end [/home/emredmil/environmentSentence/BUILD_6@r
232     8/16/11 4:54 PM         RESULT  0 (0)
```

## Pulldown environment variables

Pulldown environment variables are single variables with a set of options.

1. Click **Environments** and then create an environment variable, “pullDownSentenceEnvironment”.
2. Set **Type** to **Pulldown List**.
3. In **Name**, type `HELLOoptions`.
4. Click **Save Variable**.
5. Click the **Pulldown Options** tab.
6. In **Name**, type **HELLO1**. and value “Hello”
7. In **Value**, type `Hello`.
8. Click **Create**.

9. Repeat steps 6 on page 22 through 8 on page 22, substituting "HELLO2" for the name and "Goodbye" for the value.

## Running a project with pulldown environment variables

1. Click **Projects** and then select the **sentenceEnvironment** project.
2. Set **Environment** to **pullDownSentenceEnvironment**.
3. Create a step, called "echo3":
   a. Click **Add Step**.
   b. In **Name**, type `echo3`.
   c. In **Command**, type the following code:
      ```
      echo ${HELLOoptions}, ${WELCOMEoptions} build forge
      ```
   d. In the **Environment** list, select **None**.
   e. Click **Save Step**.
4. Click **Start Project**.
5. On the Job Details page, in the **Project Environment** section, you select the HELLOoptions and WELCOMEoptions variable values to use. For this lesson, select **HELLO1** and **WELCOME1**. The following image shows a screen capture of the sample **Project Environment** section.


6. Click **Execute** and view the job output.
7. Click **Start Project**.
8. This time, in **HELLOoptions** select **HELLO2** and in **WELCOMEoptions** select **WELCOME**.
9. Click **Execute** and view the job output.

Changing the environment variables changed the output for the project build.

## Tutorial summary

Now that you have completed this tutorial, you can:

- Create an environment with standard or pulldown variables
- Run a project which specifies the standard or pulldown environments

# Chapter 6. Tutorial 6: Dot commands in Rational Build Forge

With Dot commands, you can access the capabilities and functions of the system. You can use multiple dot commands in an IBM Rational Build Forge step and intersperse them with ordinary commands.

The tutorial shows you how to use the following commands:

- "Changing build tags with the .retag command"
- "Accessing system information with the .date command" on page 26
- "Moving files with the .put and .get commands" on page 28
- "Changing a selector with the .bset command" on page 29
- "Exporting a project with the .export command" on page 29
- "Using a .source command in an environment variable" on page 30

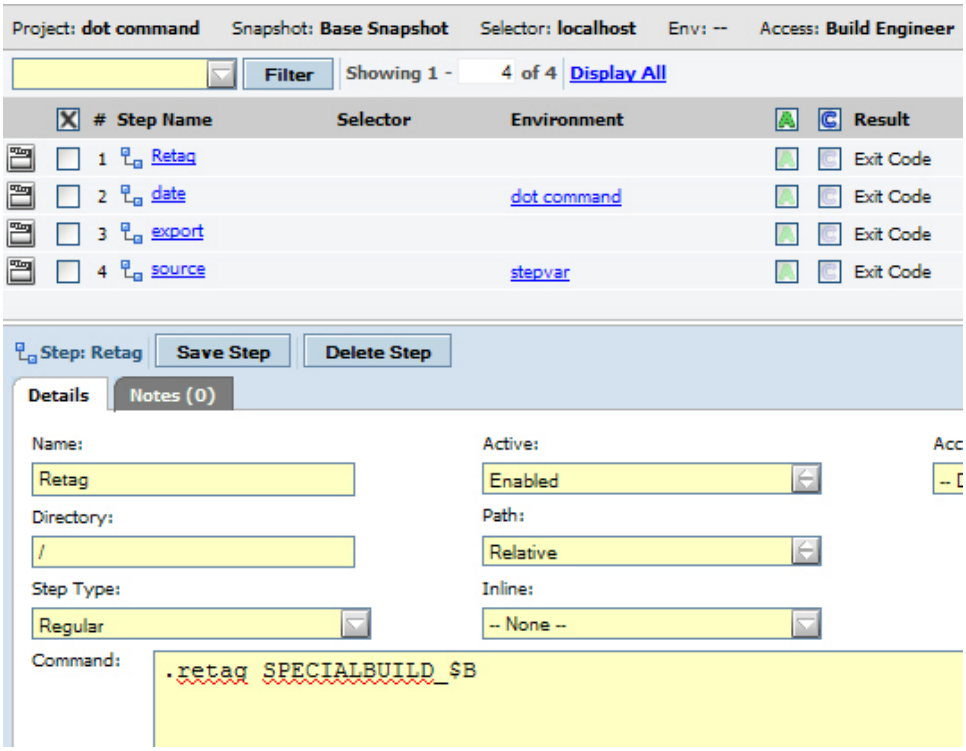
Time required: 45 minutes

## Changing build tags with the .retag command

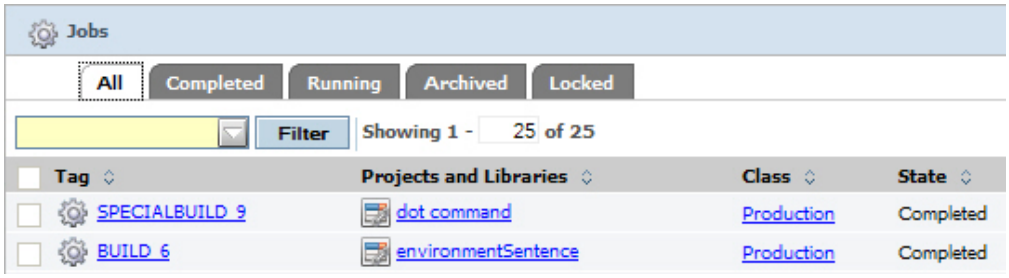
The default build tag that is displayed in the Jobs list is Build_N where N is the current build number. The **.retag** command changes the build tag for the given project. With this command, you can specify another name, such as SPECIALBUILD_1, SPECIALBUILD_2, and so on.

1. Create a project, named `dot commands`, with the appropriate selector and access groups. You learned how to create projects and steps in Chapter 1, "Tutorial 1: Creating, running, and scheduling a Rational Build Forge project," on page 1. For information about access groups, see Chapter 3, "Tutorial 3: Rational Build Forge administration: Controlling user access," on page 9.
2. Create a step named `Retag`.
3. On the Details page, in the **Command** field, type the following code:

   ```
   .retag SPECIALBUILD_$B
   ```




4. Click **Save Step**.
5. Run the *dot commands* project and view the results.

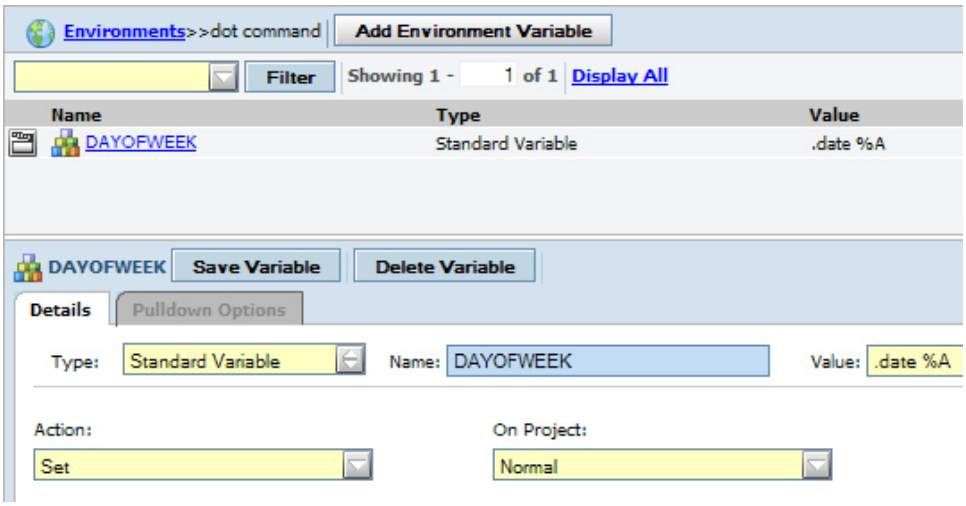



## Accessing system information with the .date command

With the **.date** command, you can access system information about the current day of the week, date, and time.

1. Create an environment named `dot command`.
2. Create a new standard environment variable, named `DAYOFWEEK`, with a value of `.date %A`. This value returns the full name of the current day of the week.

   **Tip:** For information on other **.date** commands, see the topics in the Rational Build Forge information center.




3. In the *dot commands* project, create a new step named `date`.
4. On the Details page, in the **Command** field, type the following code:
   ```
   echo The day of the week today is ${DAYOFWEEK}
   ```
5. Set **Environment** to **dot command**.
6. Run the *dot commands* project.

The following image shows the output from the *dot commands* project.

| Step | Step Name | Result | Server (Selector) |
|---|---|---|---|
| 2 | date | Passed | localhost (Default) |

STEP MANIFEST AUTH SET EXEC SSL ENV

Showing 1 - 188 of 188 Auto Paginate

```
155     8/17/11 11:02 AM        ENV     BF_SUID=51B616F6-C8EA-11E0-9FC7-95BF6CA7
156     8/17/11 11:02 AM        ENV     BF_T=110230
157     8/17/11 11:02 AM        ENV     BF_TAG=SPECIALBUILD_9
158     8/17/11 11:02 AM        ENV     BF_TAG_PHYS=SPECIALBUILD_9
159     8/17/11 11:02 AM        ENV     BF_THRESHOLD_COUNT=0
160     8/17/11 11:02 AM        ENV     BF_USER=Root User
161     8/17/11 11:02 AM        ENV     BF_USER_EMAIL=root@localhost
162     8/17/11 11:02 AM        ENV     BF_USER_LOGIN=root
163     8/17/11 11:02 AM        ENV     BF_W=3
164     8/17/11 11:02 AM        ENV     DAYOFWEEK=Wednesday
165     8/17/11 11:02 AM        ENV     HISTSIZE=1000
166     8/17/11 11:02 AM        ENV     HOME=/home/emredmil
167     8/17/11 11:02 AM        ENV     HOSTNAME=rightmeow
168     8/17/11 11:02 AM        ENV     INPUTRC=/etc/inputrc
169     8/17/11 11:02 AM        ENV     LANG=en_US.UTF-8
170     8/17/11 11:02 AM        ENV     LOGNAME=root
171     8/17/11 11:02 AM        ENV     LS_COLORS=no=00:fi=00:di=00;34:ln=00;36:
172     8/17/11 11:02 AM        ENV     MAIL=/var/spool/mail/emredmil
173     8/17/11 11:02 AM        ENV     PATH=/usr/bin:/bin
174     8/17/11 11:02 AM        ENV     SHELL=/bin/bash
175     8/17/11 11:02 AM        ENV     SUDO_COMMAND=/usr/local/bin/bfagent -s
176     8/17/11 11:02 AM        ENV     SUDO_GID=5011
177     8/17/11 11:02 AM        ENV     SUDO_UID=5010
178     8/17/11 11:02 AM        ENV     SUDO_USER=emredmil
179     8/17/11 11:02 AM        ENV     TERM=xterm
180     8/17/11 11:02 AM        ENV     USER=root
181     8/17/11 11:02 AM        ENV     USERNAME=root
182     8/17/11 11:02 AM        EXEC    Performing variable expansion on command
183     8/17/11 11:02 AM        EXEC    spawning shell [/bin/bash]
184     8/17/11 11:02 AM        SCRIPT  echo The day of the week today is Wednes
185     8/17/11 11:02 AM        EXEC    start [/home/emredmil/dot_command/SPECIA
186     8/17/11 11:02 AM        EXEC    The day of the week today is Wednesday
187     8/17/11 11:02 AM        EXEC    end [/home/emredmil/dot_command/SPECIALB
188     8/17/11 11:02 AM        RESULT  0 (0)
```

## Moving files with the .put and .get commands

The **.put** and **.get** commands are used to send and retrieve files between two servers.

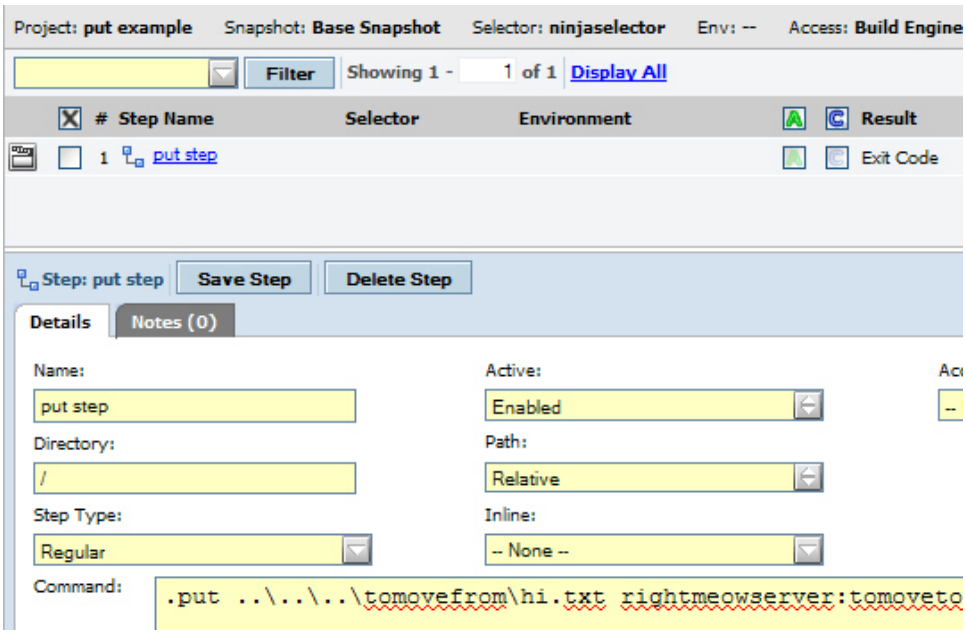
To complete this lesson, you need access to two working servers. The example uses *ninjaserver* as the server containing the file to be moved and *rightmeowserver* as the server that is the target of the moved file.

**Restriction:** Do not use these commands to transfer large files.

1. On *ninjaserver*, create a blank file called `hi.txt`.
2. In Rational Build Forge, create a project named `put`.
3. Create a step, named `putstep`, and in the **Command** field type the following code:

   ```
   .put ..\..\..\tomovefrom\hi.txt rightmeowserver:tomoveto/hi.txt
   ```

   **Note:** The path is specific to the target location of the file to be moved. A `put` command uses the following syntax: `.put [<relative_path>/]file server:[[<relative_path>/]file]`




4. Run the project.
5. Navigate to the `tomoveto` directory on the *rightmeow* server and see that the `hi.txt` file has been moved there. The following image shows an example.


## Changing a selector with the .bset command

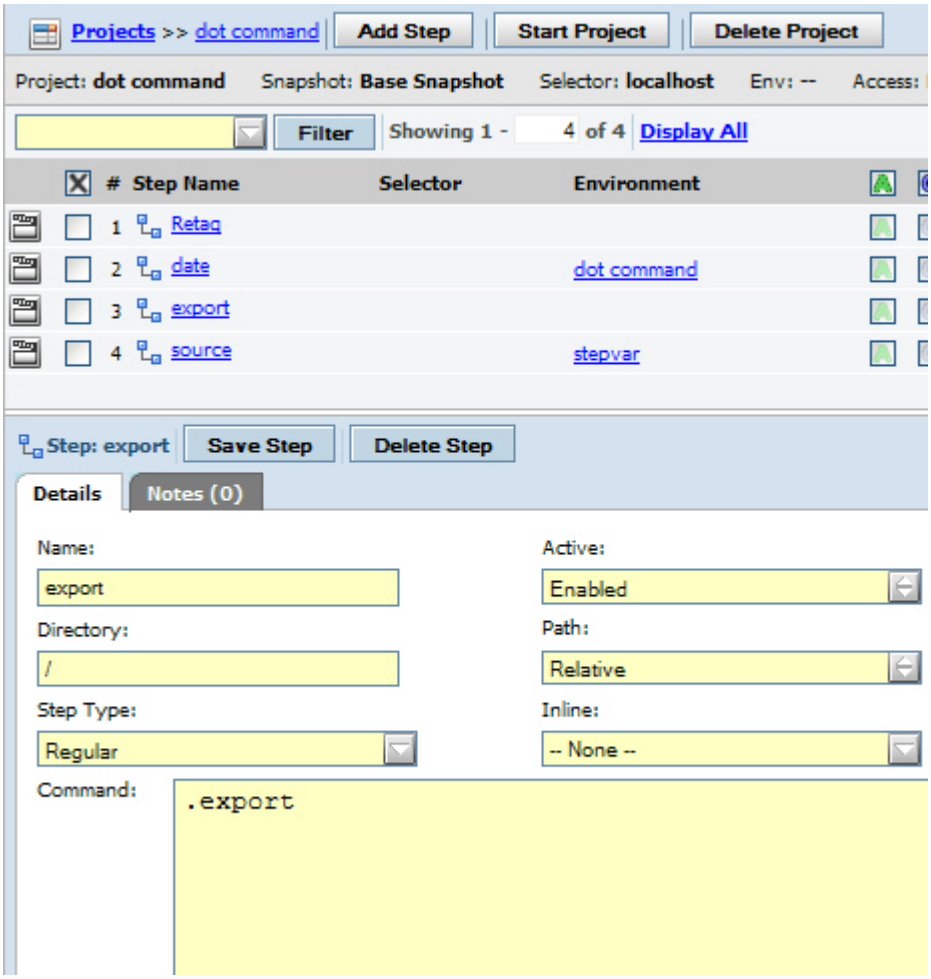
The **.bset** command is used to change the value of an environment variable or to change the server, selector, or buildserver used by the project. All steps after the step implementing **.bset** are affected by the changes made in that step.

1. In the *put* project, create a step named *bsetstep*:
   a. Click **Add Step**.
   b. In **Name**, type `bsetstep`.
   c. In **Command**, type the following code:
      ```
      .bset rightmeowselector
      ```
   d. Click **Save Step**.
2. Create another new step named `echostep`, in **Command** type `echo hi`, and then click **Save Step**. This Rational Build Forge step will show that the selector was changed.
3. Run the project and view the results.

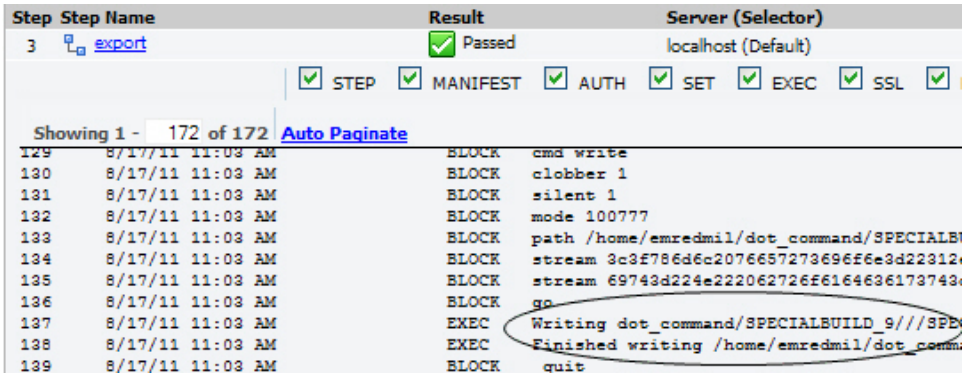
## Exporting a project with the .export command

The **.export** command saves the project definition for the calling project to an XML file located in the working directory of the step.

1. In the dot commands project, create a step named `export`, in **Command** type `.export`, and then click **Save Step**.




2. Run the project. In the log, the location of the file is listed, as shown in the following image.




## Using a .source command in an environment variable

You can use the **.source** command with both adapters and environments. We will address using the command with batch files. This lesson shows how to use the **.source** command in an environment variable for both Windows and Linux systems.

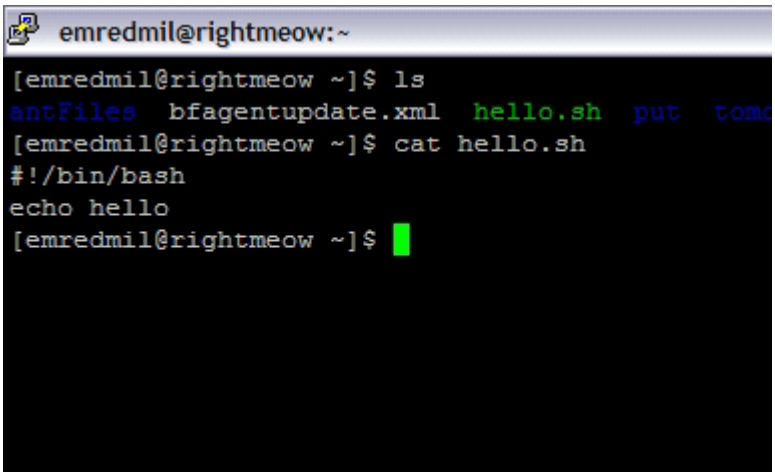
To implement .source in an environment variable on Windows systems:

1. Create a file, `hi.bat`, in the `C:/temp` directory. The file contains the command `echo hello`.
2. Create an environment named `stepvars`.
3. Create a standard environment variable:
   a. In **Name**, type `.source`.
   b. In **Value**, type `C:/temp/hi.bat`.
   c. Click **Save Variable**.
4. In the *dot commands* project, create a new step named `sourcestep`. Then, in **Command** type `echo goodbye` and in **Environment** select **stepvars**.
5. Run the project and view the result.

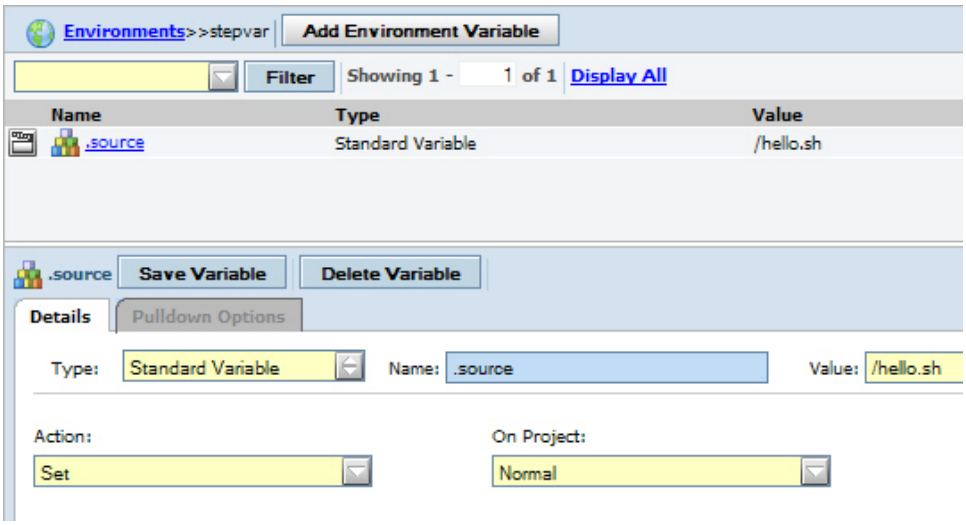
   **Note:** The batch script is executed before the step command.

To implement .source in an environment variable on Linux systems:

1. In your home directory, create a file named "hello.sh" with the commands shown below.




2. Create new environment named `stepvars`.
3. Create a standard environment variable:
   a. Click **Add Environment Variable**.
   b. In **Name**, type `.source`.
   c. In **Value**, type `/hello.sh`.
   d. Click **Save Variable**.




4. In the *dot commands* project, create a new step named `sourcestep`.
   a. Click **Add Step**.
   b. In **Name**, type `sourcestep`.
   c. In **Command**, type the following code: `echo goodbye`.
   d. In the **Environment** list, select **stepvars**.
   e. Run the project and view the results.

   **Note:** The bash script is run before the step command.

## Tutorial summary

Now that you have completed this tutorial, you can begin implementing Dot commands for your Rational Build Forge projects.

# Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 2011.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.html.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.