

Kofax RPA

Best Practices Guide for Robot Lifecycle Management

Version: 11.2.0

Date: 2021-06-01

The KOFAX logo is displayed in a bold, blue, sans-serif font. The letters are thick and closely spaced, with a consistent weight throughout the word.

© 2019–2021 Kofax. All rights reserved.

Kofax is a trademark of Kofax, Inc., registered in the U.S. and/or other countries. All other trademarks are the property of their respective owners. No part of this publication may be reproduced, stored, or transmitted in any form without the prior written permission of Kofax.

Table of Contents

Preface	4
Related Documentation.....	4
Training.....	5
Getting help with Kofax products.....	5
Chapter 1: Robot Lifecycle Management	7
Basic setup without Robot Lifecycle Management.....	7
Basic setup with Robot Lifecycle Management.....	8
Chapter 2: Basic setup with Robot Lifecycle Management	10
Select branching strategy.....	10
Recommended branching strategy.....	10
Other branching strategies.....	12
Create bare repository.....	14
Set up Management Consoles.....	15
Set up development Management Console.....	15
Set up production Management Console.....	16
Start synchronization.....	17
Promote and revert changes.....	18
Check synchronization result.....	19
Chapter 3: Access rights and prerequisites	20

Preface

This guide offers recommended methods and techniques to help you optimize performance and ensure success while using Kofax RPA.

Related Documentation

The documentation set for Kofax RPA is available here:¹

https://docshield.kofax.com/Portal/Products/RPA/11.2.0_ea1ydbmwk9/RPA.htm

In addition to this guide, the documentation set includes the following items:

Kofax RPA Release Notes

Contains late-breaking details and other information that is not available in your other Kofax RPA documentation.

Kofax RPA Technical Specifications

Contains information on supported operating systems and other system requirements.

Kofax RPA Installation Guide

Contains instructions on installing Kofax RPA and its components in a development environment.

Kofax RPA Upgrade Guide

Contains instructions on upgrading Kofax RPA and its components to a newer version.

Kofax RPA Administrator's Guide

Describes administrative and management tasks in Kofax RPA.

Help for Kofax RPA

Describes how to use Kofax RPA. The Help is also available in PDF format and known as *Kofax RPA User's Guide*.

Kofax RPA Getting Started with Robot Building Guide

Provides a tutorial that walks you through the process of using Kofax RPA to build a robot.

¹ You must be connected to the Internet to access the full documentation set online. For access without an Internet connection, see the *Installation Guide*.

Kofax RPA Getting Started with Document Transformation Guide

Provides a tutorial that explains how to use Document Transformation functionality in a Kofax RPA environment, including OCR, extraction, field formatting, and validation.

Kofax RPA Desktop Automation Service Configuration Guide

Describes how to configure the Desktop Automation Service required to use Desktop Automation on a remote computer.

Kofax RPA Developer's Guide

Contains information on the API that is used to execute robots on RoboServer.

Kofax RPA Integration API documentation

Contains information about the Kofax RPA Java API and the Kofax RPA .NET API, which provide programmatic access to the Kofax RPA product. The Java API documentation is available from both the online and offline Kofax RPA documentation, while the .NET API documentation is available only offline.

Note The Kofax RPA APIs include extensive references to RoboSuite, the original product name. The RoboSuite name is preserved in the APIs to ensure backward compatibility. In the context of the API documentation, the term RoboSuite has the same meaning as Kofax RPA.

Training

Kofax offers both classroom and computer-based training to help you make the most of your Kofax RPA solution. Visit the Kofax Education Portal at <https://learn.kofax.com/> for details about the available training options and schedules.

Also, you can visit the Kofax Intelligent Automation SmartHub at <https://smarthub.kofax.com/> to explore additional solutions, robots, connectors, and more.

Getting help with Kofax products

The [Kofax Knowledge Base](#) repository contains articles that are updated on a regular basis to keep you informed about Kofax products. We encourage you to use the Knowledge Base to obtain answers to your product questions.

To access the Kofax Knowledge Base, go to the [Kofax website](#) and select **Support** on the home page.

Note The Kofax Knowledge Base is optimized for use with Google Chrome, Mozilla Firefox or Microsoft Edge.

The Kofax Knowledge Base provides:

- Powerful search capabilities to help you quickly locate the information you need.
Type your search terms or phrase into the **Search** box, and then click the search icon.

- Product information, configuration details and documentation, including release news.
Scroll through the Kofax Knowledge Base home page to locate a product family. Then click a product family name to view a list of related articles. Please note that some product families require a valid Kofax Portal login to view related articles.
- Access to the Kofax Customer Portal (for eligible customers).
Click the **Customer Support** link at the top of the page, and then click **Log in to the Customer Portal**.
- Access to the Kofax Partner Portal (for eligible partners).
Click the **Partner Support** link at the top of the page, and then click **Log in to the Partner Portal**.
- Access to Kofax support commitments, lifecycle policies, electronic fulfillment details, and self-service tools.
Scroll to the **General Support** section, click **Support Details**, and then select the appropriate tab.

Chapter 1

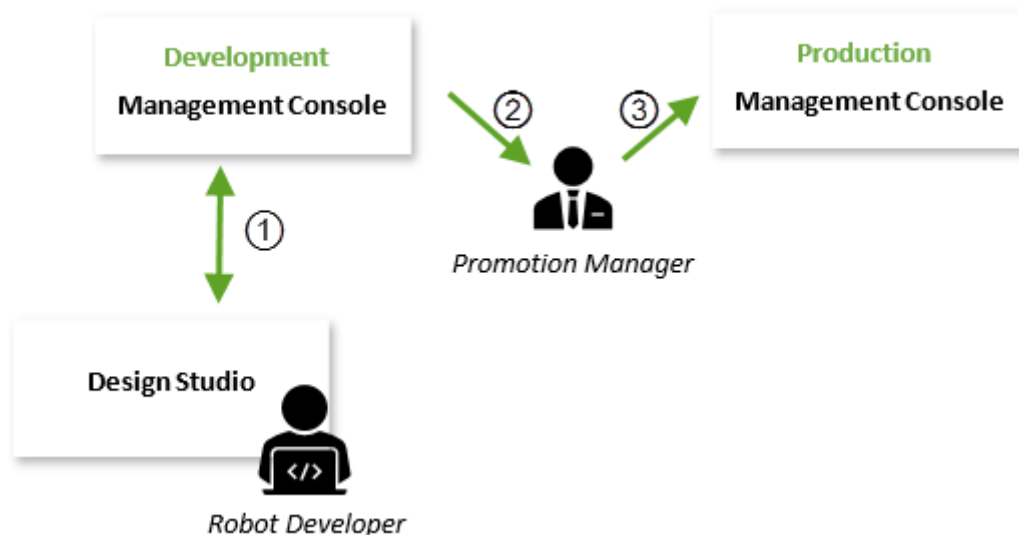
Robot Lifecycle Management

The Robot Lifecycle Management feature enables you to control work objects of various types in a version control system such as Git. Using this functionality, you can compare and synchronize the state of objects between the Management Console and your repository with the help of the Kofax RPA Synchronizer included in your installation. You can synchronize the following objects: robots, types, snippets, resources, schedules, and OAuth.

These best practices are written with the assumption that you have working knowledge of the Git version control system.

Basic setup without Robot Lifecycle Management

The following diagram illustrates a basic recommended Kofax RPA setup without Robot Lifecycle Management. This setup consists of two Management Consoles: one for development and one for production.



In this example setup, the Robot Developer establishes the synchronization **(1)** with a project stored in the development Management Console, creates new objects or makes changes to the existing objects, and then synchronizes the project again with that Management Console.

When the Promotion Manager considers a project for production, the manager makes a backup of that project **(2)** or particular objects from the development Management Console and then uploads the backup **(3)** to the production Management Console.

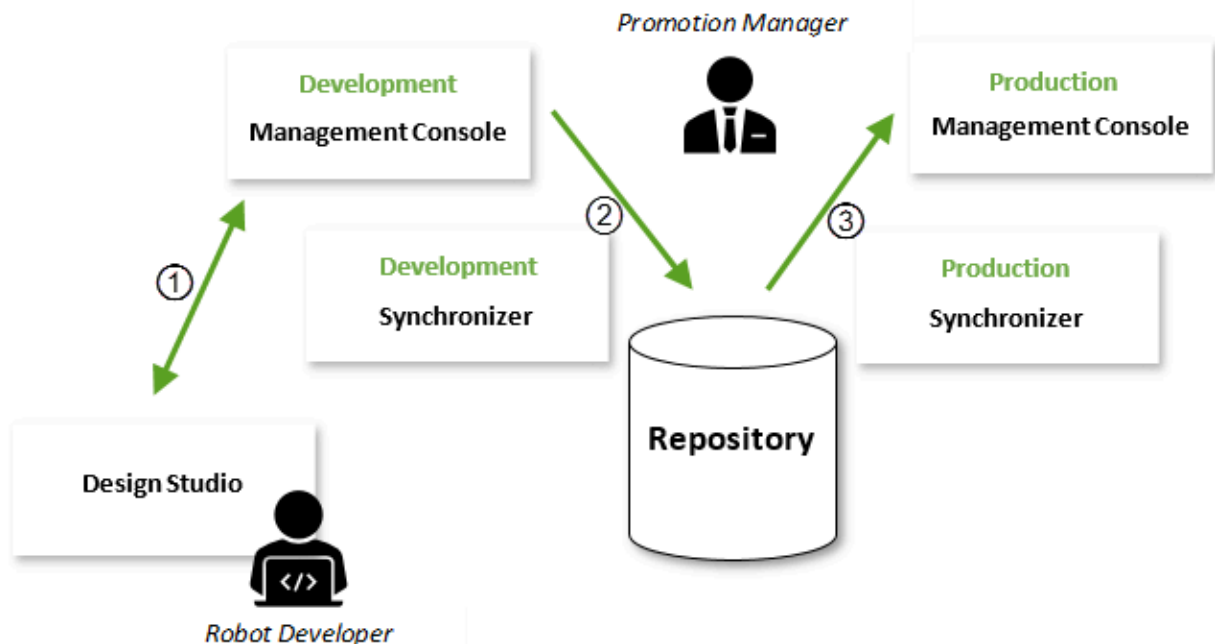
Without configuring Robot Lifecycle Management, this setup does not support the following features:

- Accurate version history for each object in a project.
- Ability to know which version is currently in production, to see the author of a change, the date an object was last changed, and a message explaining the changes.
- Ability to promote a specific version of an object from development to production.
- Ability to quickly revert to a previous object version in production.
- Ability to revert breaking changes in the development environment.

To include all of these features for use in your Kofax RPA environment, you can configure a basic setup with Robot Lifecycle Management as shown in the next topic.

Basic setup with Robot Lifecycle Management

The following diagram illustrates a basic recommended Kofax RPA setup with Robot Lifecycle Management. This setup shows two instances of Management Console that share a single repository: one instance for development and one instance for production.



In this example, the Robot Developer establishes synchronization **(1)** with a project stored in the development Management Console.

Whenever a Robot Developer updates the project in the development Management Console, the changes are automatically synchronized **(2)** with the shared repository; they are pushed as a change commit on the specified branch. The responsibility of the Production Manager is to point to the branch to use in production.

At this point, two methods to promote the change are possible:

- Merge, re-base, cherry pick, or pull the changes to the branch specified in the production Management Console.
- Simply specify the production Management Console to the new version.

When the Promotion Manager approves the changes, the production Synchronizer takes the changes **(3)** from the repository and then pushes them to the production Management Console.

In the following topics, we will recreate this setup step-by-step.

Chapter 2

Basic setup with Robot Lifecycle Management

Use the following procedure to recreate [the basic setup with Robot Lifecycle Management](#). The procedure gives step-by-step details that demonstrate how to synchronize in and out of a simple bare, file-based Git repository that is commonly referred to as a remote repository.

The Synchronizer supports all types of Git repositories to be the remote repository as long as they adhere to the Git protocol standards to transfer data. Connection to a remote repository through HTTP can only be established if the repository does not require authorization.

Tip You can check the Git online reference material for detailed information on using each type of repository. On the git-scm.com website, search for articles on the protocols, such as "Git on the Server - The Protocols." The Synchronizer supports the following Git protocol standards to transfer data: HTTP(S), SSH, and Git. The main difference is, while a simple bare Git repository resides locally on a file system, the other solutions can be connected to remotely. This choice has an impact on the physical requirements and container compositions.

To perform a basic setup that includes Robot Lifecycle Management, complete the following steps:

1. [Select an applicable branching strategy](#)
2. [Create a bare, file-based Git repository](#)
3. [Configure the Management Consoles](#)
4. [Start the Synchronizers](#)
5. [Promote objects to production and revert changes](#)
6. [Check the synchronization result](#)

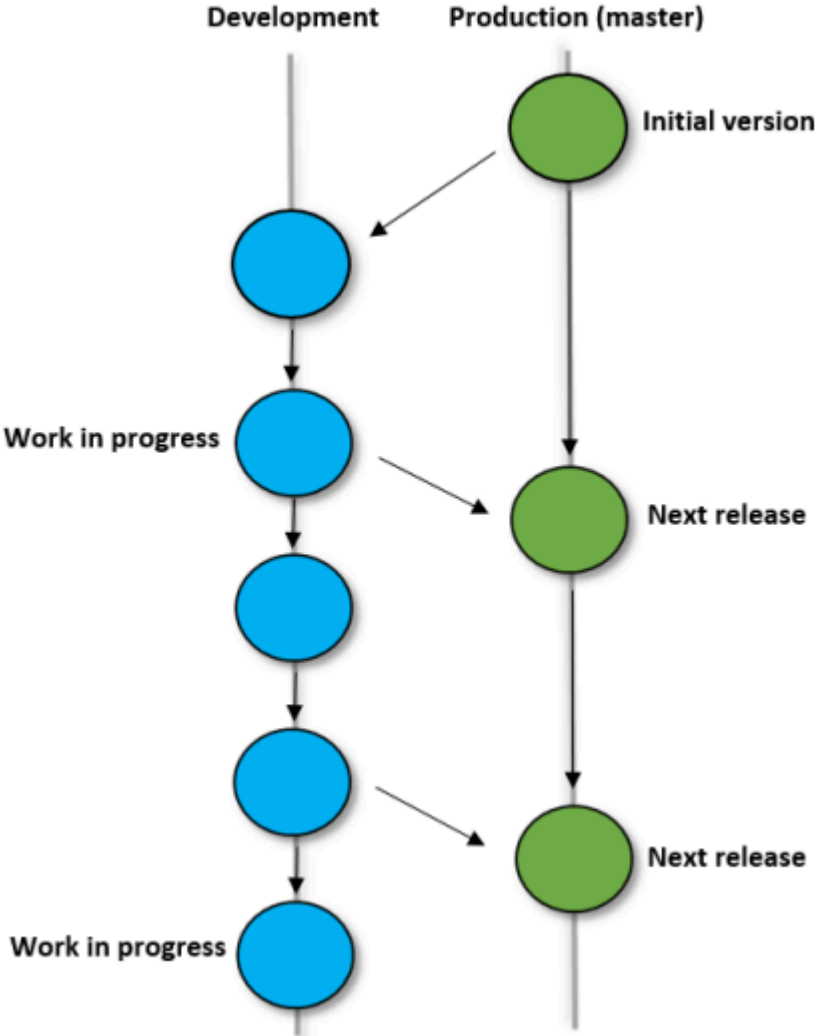
Select branching strategy

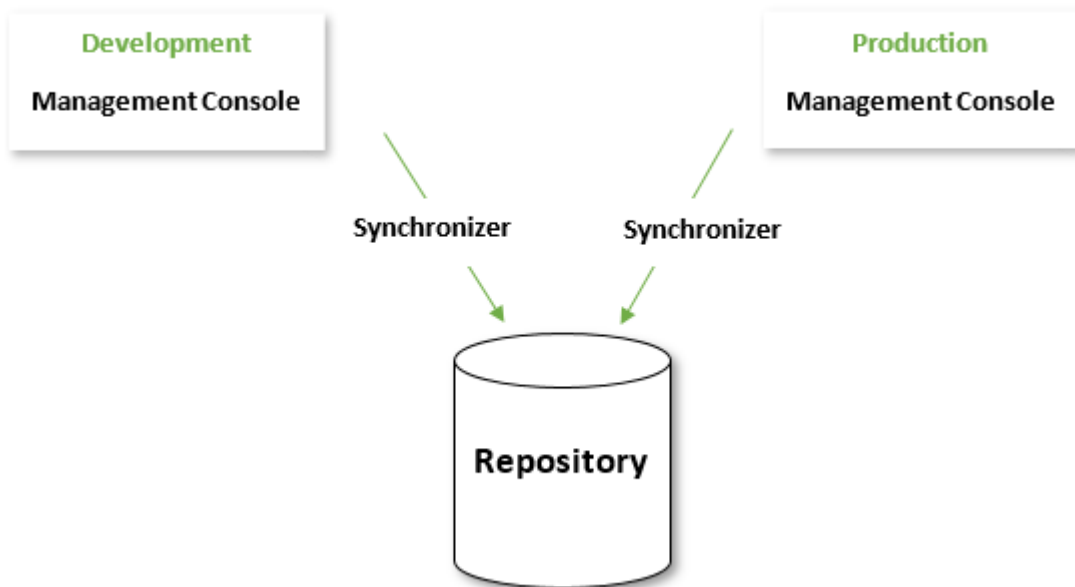
Many approaches are available to manage your work objects with a Git repository. We recommend that you follow the strategy and recommendations presented in the following topic.

Recommended branching strategy

According to this strategy, the head of the master branch always contains the current version running in production. You can add another branch to the master (production) branch, such as for development, and have the development Management Console synchronized with that branch.

The following figures demonstrate the minimal recommended branching strategy and setup. The blue and green circles are Git heads containing current versions in the development and production branches, respectively.





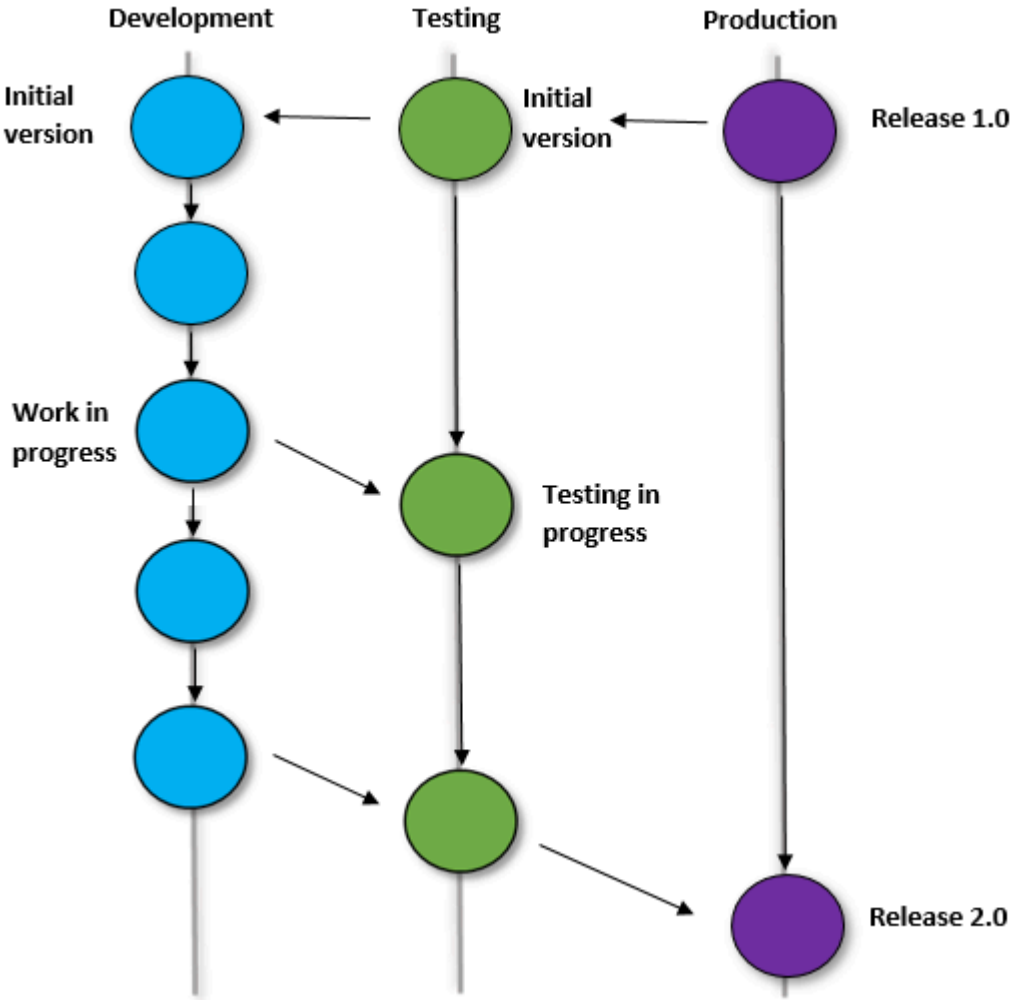
Recommendation 1: Always use the master branch for production.

Recommendation 2: When merging to the master branch, use the `--no-ff` flag, which prevents Git from executing in a "fast-forward" manner if it detects that your current head is an ancestor of the commit you are merging. It is useful to have the merge commits on your production branch to track the exact date and time of the merge.

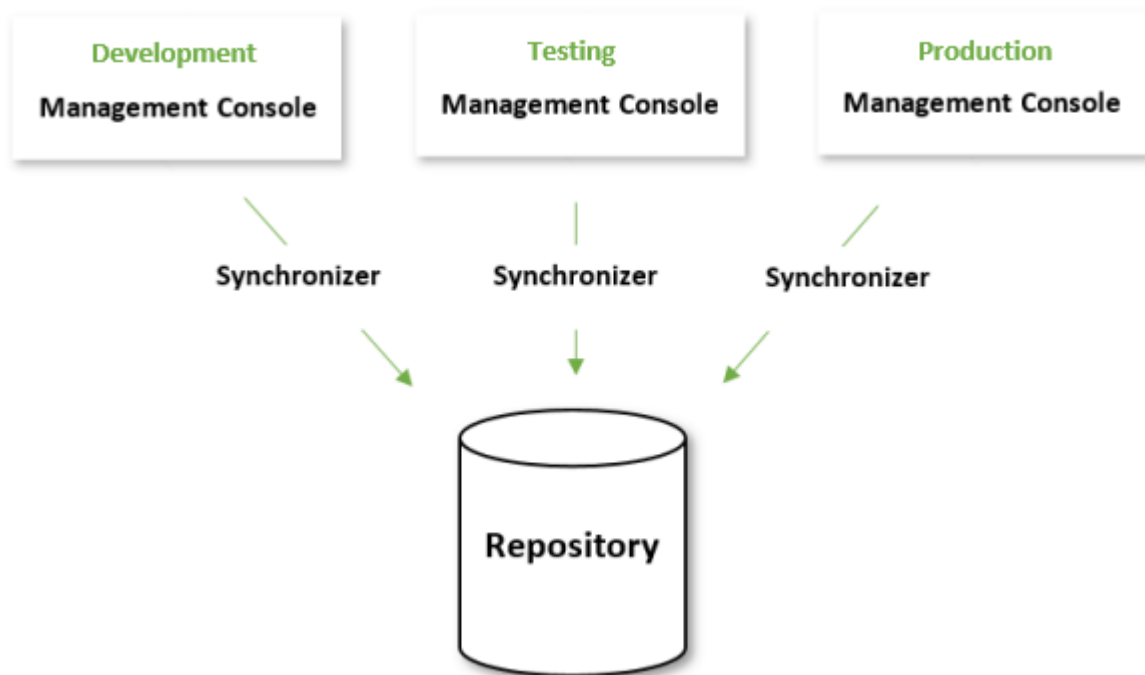
Tip To learn more about the strategy described above, see [A successful Git branching model](#).

Other branching strategies

If you prefer a more complex setup, the minimal branching strategy shown in the previous topic can be easily expanded to cover a larger setup with three branches: development, testing, and production.



In this setup, you have three Management Consoles for development, testing, and production, synchronized with three branches in a Git repository through Synchronizers.



Create bare repository

Before starting the Synchronizers and setting up the Management Consoles for synchronization, you need to initialize your Git repository. The method for creating a new bare repository depends on the third-party tool that you are using.

To create a bare repository and a development branch, you can execute the following commands.


```
# mkdir example.git
# cd example.git/
# git init --bare
Initialized empty Git repository in /gitrepos/example.git/
# cd ..
# git clone example.git
Cloning into 'example'...
warning: You appear to have cloned an empty repository.
done.
# git commit --allow-empty -m 'initial commit'
[master (root-commit) b96fdb8] initial commit
# git push origin
Everything up-to-date
# git checkout -b development
Switched to a new branch 'development'
# git push -u origin development:development
```

Tip Alternatively, you can use the Synchronizer to automatically create a bare repository. When you start the Synchronizer, it automatically creates a bare repository on the specified location if it was not already created, with an empty initial commit. However, the Synchronizer does not create a branch if it is not already present.

Set up Management Consoles

After creating a repository, you need to set up two Management Consoles: one for development and the other for production. They are used to synchronize with the development and production (master) branches in your file-based repository.

Set up development Management Console

1. Start the development Management Console.
2. On the menu, select **Admin > Projects**, click the  content menu for the project to synchronize with the repository, and then click **Edit**.
3. In the new dialog box, select the **Repository** tab.
 - a. In the **URL** property, type the path to the repository that you created [in the previous topic](#): **/gitrepos/example.git/**
 - b. In the **Branch** property, type the branch to use: **development**.
 - c. To enable the configuration specified above, select **Enable configuration**.
 - d. Under **Objects to synchronize**, select the objects to include in the synchronization: **Robots, types, and snippets**.

Enable configuration

URL
/gitrepos/example.git

Branch
development

Read-only

Objects to synchronize

Schedules


Robots, types, and snippets

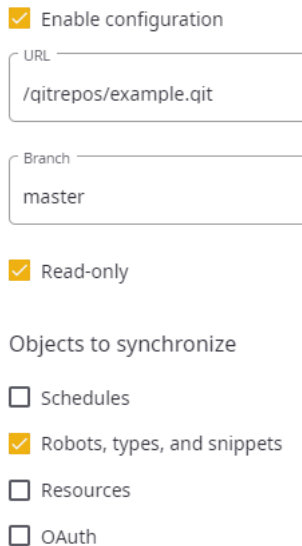
Resources

OAuth

- e. Save the changes.

Set up production Management Console

1. Start the production Management Console.
2. On the menu, select **Admin > Projects**, click the  content menu for the project to synchronize with the repository, and then click **Edit**.
3. In the new dialog box, select the **Repository** tab.
 - a. In the **URL** property, type the path to the repository that you created in [in the previous topic: /gitrepos/example.git/](#)
 - b. In the **Branch** property, type the branch to use: **master**. We recommend that you always use the master branch for production.
 - c. To make the repository the only source for object changes, select **Read-only**.
We recommend that you select this option to avoid any changes to objects belonging to the synchronized project in the production Management Console.
 - d. To enable the configuration specified above, select **Enable configuration**.
 - e. Under **Objects to synchronize**, select the objects to include in the synchronization: **Robots, types, and snippets**.



The screenshot shows a configuration dialog box with the following elements:

- Enable configuration
- URL:
- Branch:
- Read-only
- Objects to synchronize
 - Schedules
 - Robots, types, and snippets
 - Resources
 - OAuth

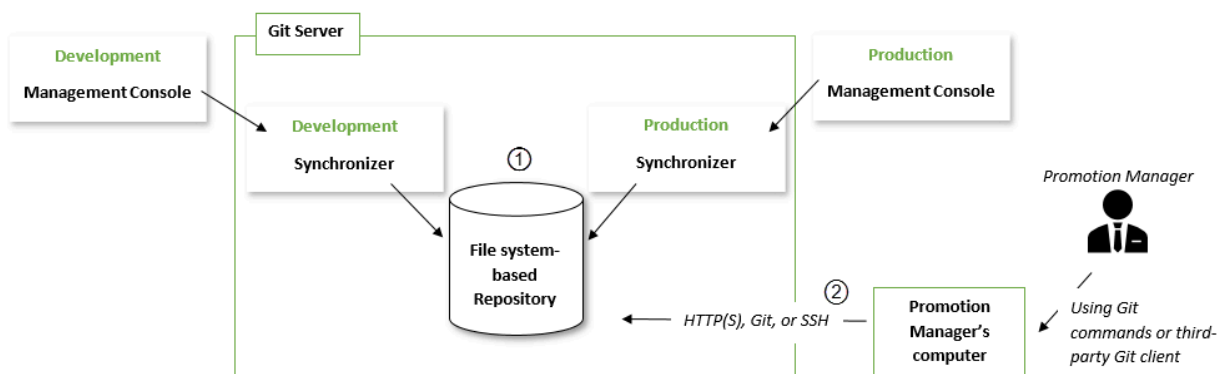
- f. Save the changes.

You have now set up your Management Consoles and are ready to start the synchronization.

Start synchronization

When using a file-based Git repository, both instances of the Synchronizer, one for the development Management Console and the other for the production Management Console, require direct access to the file system where the repository is located.

We recommend that you run both instances of the Synchronizer on the same computer where the repository is stored **(1)**. With this setup, the Promotion Manager can use one of the supported Git protocol standards **(2)** to access the repository and push changes to production.



Recommendation 1: When synchronizing with a bare file-based repository, multiple Synchronizers can run on the same computer.

Recommendation 2: For convenience, the Promotion Manager can use a third-party Git client such as SourceTree instead of Git commands.

Recommendation 3: If Docker is used for deployment, the Synchronizer containers can share a volume.

Start the Synchronizers

To connect the two Management Consoles and the Git repository, follow these steps.

1. Point the development Synchronizer to the development Management Console by configuring the following properties in the **synchronizer.settings** file.

In the Command Prompt window, first specify the `-c` property for the **synchronizer.exe** file to ignore the default `synchronizer.settings` file. Then specify the other properties that you require.

Example

```
Synchronizer.exe -c --mc-url http://127.0.0.1:8080/ManagementConsole --username
admin --password pass --interval 10 --no-host-key false --private-key $USER_HOME
\.ssh\id_rsa -s
```

- `-c, --command-line` Uses settings specified in the command-line and ignores the settings file.
 - `-e, --environment` Uses settings from the environment and ignores the settings file.
 - `-g, --generate-ssh-keys <argument>` Generates a key-pair for SSH authentication and saves it to the specified folder. For example: `-g C:\Work\MyKeys`
 - `--mc_url <argument>` *Required.* URL to connect to the development or "production" Management Console containing the protocol and a port number. Only used when `-c` is specified.
 - `--username <argument>` *Required.* Management Console user. Only used when `-c` is specified.
 - `--password <argument>` *Required.* Password for the Management Console user. Only used when `-c` is specified.
 - `--interval <argument>` *Required.* Interval in seconds between synchronization runs. If set to a value equal to or less than 0 or a non-numeric value, the Synchronizer runs once and exits. Only used when `-c` is specified.
 - `--no-host-key <argument>` *Optional.* Disables strict SSH host-key checking. Default is `false`. Only used when `-c` is specified.
 - `--private-key <argument>` *Required.* Path to a file containing the private SSH key to connect to a remote repository. When connecting to a local repository, this attribute is ignored, but any value must be specified. Only used when `-c` is specified.
 - `-r, --reset-hard` Resets the version information and purges the entire local cache.
 - `-s, --save` Saves the configuration settings in the `synchronizer.settings` file and exits.
 - `-v, --version` Prints version information and exits.
 - `-h, --help` Prints the description of properties and exits.
2. Repeat the previous step for the production Synchronizer and the production Management Console and change the properties as applicable.

You have now synchronized the Management Consoles and the Git repository and are ready to promote objects to production.

Promote and revert changes

Robots, types and snippets can now be synchronized with the development Management Console from Design Studio using the upload function or from the Management Console interface using the Robots tab. At this point, you can make changes to the production Management Console only by promoting an object version from development to production.

Use the following command to merge from the development branch into the production (master) branch.

```
# cd example
# git checkout development
# git pull
# git checkout master
# git merge --no-ff development
# git push
```

If you need to revert changes, do so on the development branch by reverting commits to a previous good state and then merging the reverted commits to the production (master) branch.

Check synchronization result

To check the synchronization result, see the respective log file, which by default resides in: `home\AppData\Local\Kofax RPA\version\Logs`. The log file location can be configured with `log4j2_synchronizer.properties`, which resides in: `home\AppData\Local\Kofax RPA\version\Configuration`.

If synchronization is successful, the log contains the name of the synchronized project and a commit ID of the synchronization. The synchronization is run at the specified interval, and object changes are automatically synchronized between the Management Console and the repository once the changes appear.

If conflicting changes occur, the repository object version has a priority over changes in the Management Console. In this case, the conflicting Management Console changes are ignored.

Note Currently, it is not supported to rename and change the location of synchronized files from the repository because it references to these files in the Management Console become incorrect.

A successful synchronization includes:

- commit ID. Denotes a commit of the repository object changes synchronized with the Management Console.
- timestamp. All objects modified in the Management Console before this point in time are synchronized with the repository.

On the Management Console interface, to see the author of a newly added object, commit message, object revision, and date of the last modification, ensure that the following columns are added to the respective view: **Modified by**, **Commit message**, **Revision number**, and **Last modified**, respectively.

Chapter 3

Access rights and prerequisites

By default, only users with the **admin**, **Administrator**, **Project Administrator**, or **VCS Service User** role can use Robot Lifecycle Management.

While all of these roles include the right to start the Synchronizer, **VCS Service User** is designed specifically for this purpose. The user with this role cannot make changes to a Management Console such as setting synchronization settings for a project. Therefore, you may assign this role to a user whose responsibilities do not include project management and primarily relate to synchronization with a version control system.

To allow a new user access to Robot Lifecycle Management, in the Management Console, the user must be added to a group that has one of these roles. For more information on user roles, see "Manage Users and Groups" in *Help for Kofax RPA*.

If the Synchronizer operates on a file system that has specific non-standard limitations on characters and, therefore, may apply some restrictions on object names, it may cause issues during the synchronization of those objects, even if the file system itself is supported. To ensure successful synchronization, check that your file system allows characters used in the synchronized object names.