



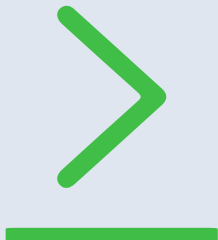
# JFROG CLI CHEAT SHEET



## WHAT IS JFROG CLI?

JFrog CLI is an open-source project, written in Golang. It is a compact and intelligent client that provides a simple interface to automate access to JFrog products, such as JFrog Artifactory, Xray and Distribution.

JFrog CLI works with the [JFrog Platform](#) to make scripts more reliable and efficient by enabling developers and DevOps teams to work in parallel using simple, easy-to-use commands for uploads, downloads and uploading the most current [build-info](#) publication. Running the [JFrog REST API](#) behind the scenes provides many advantages. For example, JFrog CLI enables uploading a full directory to a repository in Artifactory or downloading files from Artifactory according to specific criteria.



## INSTALLING JFROG CLI

Get the latest version of JFrog CLI using one of the following [installation commands](#):



DEBIAN

```
...apt install -y jfrog-cli-v2-jf;
```



HOME BREW

```
brew install jfrog-cli
```



NPM

```
...npm install -g jfrog-cli-v2-jf
```



GO

```
...go get github.com/jfrog/jfrog-cli...
```



RPM

```
...yum install -y jfrog-cli-v2-jf;
```



CURL

```
...install-cli.jfrog.io... Install
```



CHOCOLATEY

```
choco install jfrog...
```



DOCKER

```
docker run (slim)...  
docker run (full)...
```

## KEY FEATURES

### Advanced upload and download capabilities

JFrog CLI allows [uploading and downloading](#) artifacts to the [JFrog Platform](#) concurrently using a configurable number of threads that help automate and speed up the build process. For large artifacts, files can be divided into chunks that enable multiple downloads in parallel.

### Support for popular package managers and build tools

JFrog CLI offers comprehensive support for popular package managers and build tools. It seamlessly integrates with package managers such as [npm](#), [Maven](#), [NuGet](#), [Docker](#), and more, allowing you to easily manage and publish packages.

### Source code and binaries scanning

JFrog CLI empowers robust scanning capabilities to ensure the security and compliance of source code and software artifacts, including containers. It also integrates with [JFrog Xray](#), enabling scanning and analysis of projects and packages - including containers - for vulnerabilities, license compliance, and quality issues.

### Support for Build-Info

[Build-Info](#) is a comprehensive metadata [Software Bill of Materials \(SBOM\)](#) that captures detailed information about the components used in a build. It serves as a vital single source of truth, including version history, artifacts, project modules, dependencies, and other crucial data collected during the build process.

## BUILD PACKAGE INTEGRATION

JFrog CLI includes integration with different build packages such as [Maven](#), [Gradle](#), [PyPi](#) and [Docker](#). The example below lists the commands for a Maven build, that resolves dependencies and deploys build artifacts from and to JFrog Artifactory, while collecting and storing the build-info.

Define build name  
`export JFROG_CLI_BUILD_NAME=my-build-name`

Define build number  
`export JFROG_CLI_BUILD_NUMBER=1`

Set project root directory  
`cd root/of/project`

Set Maven repositories  
`jf mvn-config`

Run maven build  
`jf mvn install`

Publish build-info  
`jf rt build-publish`

## TOP 10 MOST POPULAR COMMANDS

### CLI authentication

```
jf login
```

Example: Authenticate JFrog CLI with the JFrog Platform using the web browser.

### Add configured servers

```
jf c add
```

Example: Configure the connection details of the JFrog Platform.

### Show configured servers

```
jf c show
```

Example: Show the details of the configured JFrog Platform.

### Downloading files from Artifactory

```
jf rt dl "my-local-repo/*.zip"
```

Example: Download all zip files located at the root of the my-local-repo repository to the current directory.

### Uploading files to Artifactory

```
jf rt u "build/*.zip" my-local-repo/zipFiles/
```

Example: Collect all the zip files located under the build directory (including subdirectories), and upload them to the my-local-repo repository, under the zipFiles folder, while maintaining the original names of the files.

### Running cURL with Artifactory

```
jf rt curl -XGET api/system/version
```

Example: Execute the cURL client, to send a GET request to the /api/system/version endpoint to the default configured Artifactory server.

### Running cURL with Xray

```
jf xr curl -XGET /api/v1/system/version
```

Example: Execute the cURL client, to send a GET request to the /api/system/version endpoint to the default configured Xray server.

### Scanning binaries for security vulnerabilities

```
jf s "path/to/files/*.tgz" --project "project-1"
```

Example: Scans all the tgz files located at the path/ti/files/ path in Artifactory for the "project-1" JFrog Prject.

### Auditing source projects for security vulnerabilities

```
jf audit --mvn --npm
```

Example: Audit the project in the current directory. Show all known vulnerabilities, regardless of the policies defined in Xray. Show only maven and npm vulnerabilities.

### Publishing Build-Info

```
jf rt bp my-build-name 18
```

Example: Publishes to Artifactory all the build-info collected for build my-build-name with build number 18.