Tools and commands

Apache Geronimo provides several tools for administering the server. These tools are available via command line and some via a Web based console. The Web based Geronimo console is explained in detail in the following section Geronimo Administration Console. The currently available command line tools are located in the **<geronimo_home>/bin** directory and are enumerated in the following list:

- geronimo
- startup
- shutdown
- deploy
  - General options
  - Security
  - Commands
    - Deploy
    - Login
    - Redeploy
    - Start
    - Stop
    - Undeploy
    - Distribute
    - Encrypt
    - Install-library
    - List-modules
    - List-targets
    - UnlockKeystore
    - Install-plugin
    - Search-plugins
- client
- jaxws-tools
  - wsgen
  - wsimport

Although the tools name is self-explanatory, it may not be the same case with the tool's parameters. The following sections explain more in detail these tools and commands usage.

geronimo

The **geronimo** command lets you perform two actions, that is start and stop the server in different modes depending on the parameters you specify. This command has the following syntax:

```
<geronimo_home>/bin/geronimo [options]
```

The available options are:

debug
This option will start the server in **JDB** debugger.

jpda run
This option will start the server in foreground under JPDA debugger

jpda start
This option will start the server in background under JPDA debugger.

run
This option will start the server in the current window.

start
This option will start the server in a separate window just like the startup command.

stop
This option will stop the server.

Both **start** and **stop** options for this command will have the same set of parameters as the startup and shutdown commands.

startup

The **startup** command starts the Apache Geronimo server. This command has the following syntax:

```
<geronimo_home>/bin/startup <options>
```

The available options are:

--quiet
Suppress the normal startup progress bar. This is typically used when redirecting console output to a file, or starting the server from an IDE or other tool.

--long
Write startup progress to the console in a format that is suitable for redirecting console output to a file, or starting the server from an IDE or other tool (does not use linefeeds to update the progress information that is used by default if you do not specify --quiet or --long).

-v --verbose
Sets the console log level to **INFO**, resulting in more console output than is normally present.

-vv --veryverbose
Sets the console log level to **DEBUG**, resulting in even more console output.

-override [*configId*]
Overrides the configurations in **<geronimo_home>/var/config.xml** such that only the configurations listed on the command line will be started. Note that many J2EE
features depend on certain configs being started, so you should be very careful what you omit. Any arguments after **-override** are assumed to be configuration names.

The **startup** command can also be started by using the **java -jar** command:

```
java -Djava.endorsed.dirs=lib/endorsed -javaagent:bin/jpa.jar -jar bin/server.jar <options>
```

shutdown

The **shutdown** command stops the Apache Geronimo server. This command has the following syntax:

```
<geronimo_home>/bin/shutdown [options]
```

The available options are:

--user [*user_name*]
Specifies the user name with the authority to stop the server. By default you would normally use **system** as the user name.

--password [*password*]
Specifies the password for the user name you just entered. By default you would normally use **manager** as the password.

--port [*port_number*]
Specifies the RMI naming port to connect to the server (for example JMX connection port). By default port **1099** is used.

--host [*host_name*]
Specifies the host name of the server you are trying to stop. By default **localhost** is used. *This parameter is only available in Geronimo 2.1.3 and greater.*

--secure
Use secure channel to communicate with JMX server, see #Security for details. *This parameter is only available in Geronimo 2.1.3 and greater.*

If you do not specify any parameters, this command will prompt you for a user name and password and will connect to port 1099 on localhost.

The **shutdown** command can also be started by using the **java -jar** command:

```
java -Djava.endorsed.dirs=lib/endorsed -jar bin/shutdown.jar <options>
```

deploy

The **deploy** command is used for installing, uninstalling, reinstalling, starting and stopping applications and modules and for installing and uninstalling configurations (for example some configuration specific deployment plans, security realms, database connection pools etc.)

This command has the following syntax:

```
<geronimo_home>/bin/deploy <general_options> <command> <command_options>
```

where **<general_options>** specify common options that apply to all commands and control how the application behaves, **<command>** is a command name that specifies the action to be performed, and **<command_options>** are options unique to the command specified.

The **deploy** command can also be started by using the **java -jar** command:

```
java -Djava.endorsed.dirs=lib/endorsed -jar bin/deployer.jar <general_options> <command> <command_options>
```

General options

This section lists all the available general options for the Geronimo deployer tool.

- **--uri** <identifier>
  Where <identifier> is a Universal Resource Identifier (URI) that specifies how the deployer is to contact the server. If this flag is not specified, the deployer will attempt to contact the server using the standard port on localhost. The identifier must have the following form:
  deployer:geronimo:jmx:rmi:///jndi/rmi:[//host[:port]]/JMXConnector
  where <host> is replaced with the host name or TCP/IP address of the system where the server is running and <port> is replaced with the port number where the server is listening. If unspecified, localhost and the default port will be used.

- **--host** <host>
  Where <host> is the host name of the server you are trying to deploy that application or resource. This option allows you to deploy resources and applications to a remote server. This parameter is optional and defaults to localhost.

- **--port** <port>
  Where <port> is the port of the remote server you are trying to deploy that application or resource. This parameter is optional and defaults to port 1099.

- **--driver** <driver_path>
  Where <driver_path> is the path to the driver JAR if you want to use this tool with a server other than Geronimo. Currently, manifest Class-Path entries in that JAR are ignored.

- **--user** <username>
  Where <username> is a user name authorized to be an administrator on the server. If the command requires authorization, you must use this option.

- **--password** <password>
  Where <password> is a the password required to authenticate the user name. If this flag is not specified, the deployer will attempt to perform the command with no password, but if that fails, it will prompt you to enter a password.

- **--secure**
  Use secure channel to communicate with JMX server, see #Security for details. *This parameter is only available in Geronimo 2.1.2 or greater.*

- **--syserr** <select>
  Where <select> can be either true or false. If this flag is unspecified. false is assumed. Specify true when you want errors to be logged to the syserr device.

- **--verbose** <select>
  Where <select> can be either true or false. If this flag is unspecified. false is assumed. Specify true when you need more messages to determine the cause of an error.

Back to top

Security

Starting with Geronimo 2.1.2, the deployer tool can use a secure channel (SSL/TLS) to communicate with the JMX server to perform the given actions. To enable secure communication just add the **--secure** option. Depending on your configuration you might also need to specify some Java security properties to configure the JVM to use the right keystores and passwords. For example, on a default Geronimo installation you might need to set the following (all in one line):

```
export JAVA_OPTS="-Djavax.net.ssl.keyStore=<geronimo_home>/var/security/keystores/geronimo-default -Djavax.net.
ssl.keyStorePassword=secret \
  -Djavax.net.ssl.trustStore=<geronimo_home>/var/security/keystores/geronimo-default -Djavax.net.ssl.
trustStorePassword=secret"
```

Once that property is set, you can execute the following command (just as an example):

```
<geronimo_home>/bin/deploy -u system -p manager --secure list-modules
```

The secure JMX server might not be running by default. Please see Configure secure JMX server for more information.

Back to top

Commands

The available commands for the Geronimo deployer tool are listed below:

- Common commands
  - #Deploy
  - #Login
  - #Redeploy

    - #Start
    - #Stop
    - #Undeploy
- Other commands
    - #Distribute
    - #Encrypt
    - Install-library
    - List-modules
    - List-targets
    - #UnlockKeystore
- Geronimo Plugins
    - Install-plugin
    - Search-plugins

Additionally, you can type **help** for further details on a given command, the syntax is as follows:

```
<geronimo_home>/bin/deploy help <command>
```

Back to top

Deploy

Use the **deploy** command to add and start a new module. The deploy command has the following syntax:

```
<geronimo_home>/bin/deploy <general_options> deploy <module> <deployment_plan>
```

The <module> specifies the application file name and location. The <deployment_plan> specifies the file name and location of the XML with the deployment plan. Sometimes the application module already has included in the package a deployment plan or the application is so simple that does not require any deployment plan, in these cases this parameter can be omited.

A module file can be one of the following:

- J2EE Enterprise Application Archive (EAR) file
- J2EE Web Application Archive (WAR) file
- J2EE Enterprise JavaBean Archive (JAR) file
- J2EE Java Resource Archive (RAR) file

If the server is not currently running at the time of deploying the application, the module will be marked to start next time the server is started.

The most common <general_options> would be --user and --password. The **--inPlace** option allows you point to and deploy an application directly from a directory external to Geornimo without the need for even packaging the application. In other words, you can have an application **running** in Geronimo but that application may be anywhere else on the file system.

Please note that the **--inPlace** option cannot be used when deploying an application to a remote server.

To use this option you should type:

```
<geronimo_home>/bin/deploy <general_options> deploy --inPlace <app_home>
```

Where <app_home> indicates the home directory where you have your application (exploded).

You can also deploy applications if Geronimo is not running by using the **--offline** option, the syntax for this command would be:

```
<geronimo_home>/bin/deploy <general_options> --offline deploy <module>
```

Off course, you can also combine --offline and --inPlace

```
<geronimo_home>/bin/deploy <general_options> --offline deploy --inPlace <app_home>
```

Back to top

Login

Use the **login** command to save the username and password for the current connection to the file `.geronimo-deployer` in the current user's home directory. Future connections to the same server will try to use this saved authentication information instead of prompting where possible.

This information will be saved separately per connection URL, so you can specify --url or --host and/or --port on the command line to save a login to a different server.

The **login** command has the following syntax:

```
<geronimo_home>/bin/deploy --user <user_name> --password <password> login
```

So, next time you run a different command that originally required user name and password, you can run the command directly, for example:

```
<geronimo_home>/bin/deploy list-modules
```

Even when the login information is not saved in clear text, it is not secure either. If you want to save the authentication securely, you should change the `.geronimo-deployer` file in your home directory so that nobody else can read or write it.

Back to top

Redeploy

Use the **redeploy** command to stop, replace and restart a module that has been deployed before. The redeploy command has the following syntax:

```
<geronimo_home>/bin/deploy <general_options> redeploy <--targets target> <module> <deployment_plan>
```

Just like the deploy command, the redeploy command accepts the following modules file types:

- J2EE Enterprise Application Archive (EAR) file
- J2EE Web Application Archive (WAR) file
- J2EE Enterprise JavaBean Archive (JAR) file
- J2EE Java Resource Archive (RAR) file

Typically, both a module and a plan are specified. If the module contains a plan or if a default plan can be used, the plan can be omitted. However, if a plan is specified in this case, it overrides the other plans. If the plan references a server component already deployed in the server's environment, the module is omitted. you can use **--targets** option only for clustering redeployment. For clustering redeployment you can find the target with **list-targets** command. Copy the one with the name as MasterConfigurationStore and use it as a target variable.

Back to top

Start

Use the **start** command to start a previously deployed module. The start command has the following syntax:

```
<geronimo_home>/bin/deploy <general_options> start <moduleIDs>
```

Where <moduleIDs> is a list of one or more modules (configID) separated by blank space. The module identification (or ConfigID) is defined at deployment time in the respective deployment plan for each module previously deployed.

Back to top

Stop

Use the **stop** command to stop a running module. The stop command has the following syntax:

```
<geronimo_home>/bin/deploy <general_options> stop <moduleIDs>
```

Where <moduleIDs> is a list of one or more modules (configID) separated by blank space. The module identification (or ConfigID) is defined at deployment time in the respective deployment plan for each module previously deployed.

Back to top

Undeploy

Use the **undeploy** command to stop and remove a module (running or not) and its deployment information from the server. The undeploy command has the following syntax:

```
<geronimo_home>/bin/deploy <general_options> undeploy <moduleIDs>
```

Where <moduleIDs> is a list of one or more modules (configID) separated by blank space. The module identification (or ConfigID) is defined at deployment time in the respective deployment plan for each module previously deployed.

This command has the same ability as with **deploy** to uninstall applications when the server is not running, this command has the following syntax:

```
<geronimo_home>/bin/deploy <general_options> --offline undeploy <moduleID>
```

Back to top

Distribute

Use the **distribute** command to add a new module to the server. This command does not start the module nor mark it to be started in the future. The distribute command has the following syntax:

```
<geronimo_home>/bin/deploy <general_options> distribute <module> <deployment_plan>
```

Just like with the deploy command, <module> specifies the application file name and location. The <deployment_plan> specifies the file name and location of the XML with the deployment plan. Sometimes the application module already has included in the package a deployment plan or the application is so simple that does not require any deployment plan, in these cases this parameter can be omitted.

A module file can be one of the following:

- J2EE Enterprise Application Archive (EAR) file
- J2EE Web Application Archive (WAR) file
- J2EE Enterprise JavaBean Archive (JAR) file
- J2EE Java Resource Archive (RAR) file

Back to top

Encrypt

Use the **encrypt** command to generate an encrypted string. This command takes use of org.apache.geronimo.util.EncryptionManager and has the following syntax:

`<geronimo_home>/bin/deploy <general_options> encrypt <string>`

Where <general_options> are common options that apply to all commands, <string> specifies a string to get encrypted.

Currently password strings are plain text in deployment plans, such as datasource or JMS deployment plans within an EAR. It might pose a security problem to store password strings as plain text even though the deployment plans are only used during the deployment process, and not at runtime. Starting from Geronimo 2.1.5, users can encrypt passwords using the encrypt command and paste the encrypted strings into deployment plans as password.

Examples:

Use this syntax to encrypt string `passw0rd` on an active server so that the encryption settings of that server will be used

```
deploy --user myadmin --password mypassword encrypt passw0rd
```

Online encryption result:

```
......
String to encrypt: passw0rd
Online encryption result:
{Simple}rO0ABXNyABlqYXZheC5jcnlwdG8uU2VhbGVkT2JqZWN0PjY9psO3VHACAARbAA1lbmNv
ZGVkUGFyYW1zdAACW0JbABBlbmNyeXB0ZWRDb250ZW50cQB+AAFMAAlwYXJhbXNBbGd0ABJMamF2YS9s
YW5nL1N0cmluZztMAAdzZWFsQWxncQB+AAJ4cHB1cgACW0Ks8xf4BghU4AIAAHhwAAAAEG2NoqXONCcU
GqfK0reVCpVwdAADQUVT
```

Use this syntax to encrypt string passw0rd offline

```
deploy --offline encrypt passw0rd
```

Offline encryption result:

```
......
String to encrypt: passw0rd
Offline encryption result:
{Simple}rO0ABXNyABlqYXZheC5jcnlwdG8uU2VhbGVkT2JqZWN0PjY9psO3VHACAARbAA1lbmNv
ZGVkUGFyYW1zdAACW0JbABBlbmNyeXB0ZWRDb250ZW50cQB+AAFMAAlwYXJhbXNBbGd0ABJMamF2YS9s
YW5nL1N0cmluZztMAAdzZWFsQWxncQB+AAJ4cHB1cgACW0Ks8xf4BghU4AIAAHhwAAAAEG2NoqXONCcU
GqfK0reVCpVwdAADQUVT
```

Note: Online encryption needs a running server to connect to and will use the encryption settings of that server, such as an encryption key, to do the encryption. As a result, the encrypted password usually can only be used for that particular server. Offline encryption uses the default encryption settings, and the encrypted password can be used by all servers. Offline encryption is thus less secure than online encryption.

Back to top

Install-library

Use the **install-library** command to install a library into server's repository. The install-library command has the following syntax:

`<geronimo_home>/bin/deploy <general_options> install-library [--groupId groupName] <libraryFile>`

Use the **--groupId** option to specify a non-default group id for the library. Otherwise, the library file will be installed with the group id named **default**.

Examples:

```
<geronimo_home>/bin/deploy -u system -p manager install-library mylib-1.0.jar
```

That command will install the mylib-1.0.jar at **`<geronimo_home>/repository/default/mylib/1.0/mylib-1.0.jar`**

```
<geronimo_home>/bin/deploy -u system -p manager install-library --groupId mygroup mylib-1.0.jar
```

That command will install the mylib-1.0.jar at **`<geronimo_home>/repository/mygroup/mylib/1.0/mylib-1.0.jar`**

Back to top

List-modules

Use the **list-modules** command to list all available modules on the server, note that for running this command the server must be running. The list-modules command has the following syntax:

`<geronimo_home>/bin/deploy <general_options> list-modules [-`~~**`all|`**~~~~started|~~-stopped]

- --all : is used by default when no other option is specified. It will list all the available modules.
- --started : this option will list only the modules that are running.
- --stopped : this option will list only the modules that are not running.

Back to top

List-targets

Use the **list-targets** command to lists the targets known to the server you have connected to. The list-targets command has the following syntax:

`<geronimo_home>/bin/deploy <general_options> list-targets`

In the case of Geronimo, each configuration store is a separate target. Geronimo does not yet support clusters as targets.

Back to top

UnlockKeystore

Use the **unlockKeystore** command to unlock a keystore and private keys. The unlockKeystore command has the following syntax:

`<geronimo_home>/bin/deploy <general_options> unlockKeystore <keyStoreName> <keyAlias1> <keyAlias2>`

Where <keyStoreName> specifies a locked keystore to get unlocked, <keyAlias1> and <keyAlias2> are optionally used to specify one or more locked private keys in the keystore to get unlocked.

Note that before you can use the unlockKeystore command, you need to ensure that the following lines are added to <geronimo_home>/var/config/config-substitutions.properties:

```
<keyStoreName>=<keyStoreEncryptedPassword>
<keyAlias1>=<keyAlias1EncryptedPassword>
<keyAlias2>=<keyAlias1EncryptedPassword>
...
```

Where

- <keyStoreName> is the name of the keystore.
- <keyStoreEncryptedPassword> is the encrypted password for the keystore, which can be generated by using the **encrypt** command. When you copy and paste the generated encrypted password to <geronimo_home>/var/config/config-substitutions.properties, there should be no space in the encrypted password string.
- <keyAlias1>, <keyAlias2> are the names of the private keys in the keystore.
- <keyAlias1EncryptedPassword>, <keyAlias2EncryptedPassword> are the encrypted passwords for the private keys, which can also be generated by using the encrypt command.

Alternatively, you can create a key file to contain the passwords of the keystore and its private keys. Use `org.apache.geronimo.keyStoreTrustStorePasswordFile` property to specify the key file. See Configuring SSL client authentication for more detailed instructions.

Examples:

Use this syntax to unlock the keystore whose name is mykeystore

```
deploy --user myadmin --password mypassword unlockKeystore mykeystore
```

Use this syntax to unlock the keystore whose name is mykeystore and the private key whose alias is key1

```
deploy --user myadmin --password mypassword unlockKeystore mykeystore key1
```

Back to top

Install-plugin

Use the **install-plugin** command to install a Geronimo plugin previously exported from a Geronimo server or downloaded from a repository. A Geronimo plugin can be an application, a configuration such data sources and drivers or a combination. The install-plugin command has the following syntax:

`<geronimo_home>/bin/deploy install-plugin <plugin_file>`

Back to top

Search-plugins

Use the **search-plugins** command to list all the Geronimo plugins available in a Maven repository. The search-plugins command has the following syntax:

`<geronimo_home>/bin/deploy search-plugins <maven_repository_URL>`

Back to top

client

The **client** command launches the client application container. This command has the following syntax:

`<geronimo_home>/bin/client config-name [app arg] [app arg] ...`

The first argument identifies the Geronimo configuration that contains the application client you want to run. The rest of the arguments will be passed as arguments to the client application when it is started.

The **client** command can also be started by using the **java -jar** command:

`java -Djava.endorsed.dirs=lib/endorsed -jar bin/client.jar config-name [app arg] [app arg] ...`

jaxws-tools

The **jaxws-tools** command can be used to generate portable artifacts used in JAX-WS web services. For example, portable artifacts such as Service Endpoint Interface (SEI) class, Service class, JAXB generated value types, etc.

This command has the following syntax:

`<geronimo_home>/bin/jaxws-tools toolName [toolOptions] ...`

Where `toolName` can be either:

- **#wsgen** - generate portable artifacts from Java
- **#wsimport** - generate portable artifacts from WSDL

The **jaxws-tools** command can also be started by using the **java -jar** command:

`java -Djava.endorsed.dirs=lib/endorsed -jar bin/jaxws-tools.jar toolName [toolOptions] ...`

### wsgen

The **jaxws-tools** uses the wsgen tool provided by Sun to generate the portable artifacts from Java class. The **wsgen** tools has the following syntax:

`<geronimo_home>/bin/jaxws-tools wsgen [options] <SEI>`

- **-classpath** <path>
  Specifies where to find input class files.

- **-d** <directory>
  Specifies where to place generated output files.

- **-keep**
  Keep generated files.

- **-r** <directory>
  Specifies where to place resource files such as WSDLs.

- **-s** <directory>
  Specifies where to place generated source files.

- **-wsdl**
  Generate a WSDL file.

- **-servicename** <name>
  Specifies the Service name to use in the generated WSDL (used in conjunction with the -wsdl option).

- **-portname** <name>
  Specifies the Port name to use in the generated WSD (used in conjunction with the -wsdl option).

Example:

```
<geronimo_home>/bin/jaxws-tools wsgen -d output -keep -wsdl -classpath . foo.BarService
```

wsimport

The **jaxws-tools** uses the **wsimport** tool provided by Sun to generate the portable artifacts from WSDL. The **wsimport** has the following syntax:

`<geronimo_home>/bin/jaxws-tools wsimport [options] <WSDL_URI>`

- **-d** <directory>
  Specifies where to place generated output files.

- **-keep**
  Keep generated files.

- **-p** <pkg>
  Specifies the target package.

- **-s** <directory>
  Specifies where to place generated source files.

- **-wsdllocation** <location>
  Specified @WebService.wsdlLocation and @WebServiceClient.wsdlLocation value.

Example:

```
<geronimo_home>/bin/jaxws-tools wsimport -d output -keep http://localhost:8080/foo/Bar?wsdl
```