Pipelines Utility Functions

The **Utility Functions** are built-in shell functions that can be used in steps to interact with the runtime environment.

Most utility functions are available in both Bash (Linux) and Powershell (Windows) OS runtimes.

Page Contents

- bump_semver
- replace_envs
- retry_command
- get_uuid
- save_artifact_info
- validate_artifact
- configure_jfrog_cli
- cleanup_jfrog_cli

Source Control
- compare_git
- update_commit_status

Test Reports
- save_tests

Encryption
- encrypt_string
- decrypt_string
- encrypt_file
- decrypt_file

Notifications
- send_notification

JSON
- set_payload
- read_json

Resources
- replicate_resource
- write_output

Caching
- add_cache_files
- restore_cache_files

Run State Management
- add_run_variables
- export_run_variables
- add_run_files
- restore_run_files

Pipeline State Management
- add_pipeline_variables
- export_pipeline_variables
- add_pipeline_files
- restore_pipeline_files

Step Properties
- find_resource_variable
- get_integration_name
- get_resource_name
- get_resource_names
- find_step_configuration_value

bump_semver

Description

Increments the provided semver version with the given action.

Usage

```
bump_semver <semver string> <action>
```

- `semver string` is the semver version to be incremented
- `action` is the type of increment to be applied

The valid actions are:

- major: Increment the major version. The minor and patch versions are reset to 0.
- minor: Increment the minor version. The patch version is reset to 0.
- patch: Increment the patch version.
- alpha: Increment or add an `alpha` pre-release tag. For example, `v1.1.1` becomes `v1.1.1-alpha` and `v1.1.1-alpha` becomes `v1.1.1-alpha.1`. Any other pre-release tags will be removed.
- beta: Increment or add a `beta` pre-release tag. For example, `v1.1.1` becomes `v1.1.1-beta` and `v1.1.1-beta` becomes `v1.1.1-beta.1`. Any other pre-release tags will be removed.
- rc: Increment or add an `rc` pre-release tag. For example, `v1.1.1` becomes `v1.1.1-rc` and `v1.1.1-rc`becomes `v1.1.1-rc.1`. Any other pre-release tags will be removed.
- final: Remove any pre-release tags, leaving `major.minor.patch`.

Examples

```
ubuntu:~$ bump_semver v1.0.0 major
v2.0.0

ubuntu:~$ bump_semver v1.0.0 minor
v1.1.0

ubuntu:~$ bump_semver v1.0.0 patch
v1.0.1

ubuntu:~$ bump_semver v1.0.0 rc
v1.0.0-rc

ubuntu:~$ bump_semver v1.0.0-rc rc
v1.0.0-rc.1

ubuntu:~$ bump_semver v1.0.0 alpha
v1.0.0-alpha

ubuntu:~$ bump_semver v1.0.0 beta
v1.0.0-beta

ubuntu:~$ bump_semver v1.0.0-rc.12 final
v1.0.0

ubuntu:~$ bump_semver v1. rc
error: Invalid semantic version given in the argument.

ubuntu:~$ bump_semver v1.0.0 badaction
error: Invalid action given in the argument.
```

replace_envs

Description

Replaces variables in a file with values based on your current shell `env`. This is useful to create config files from templates, for example.

If the file contains placeholders that are not defined in the environment, they will become empty strings (“”). The original file is overwritten with the modified file.

Usage

```
replace_envs <filename1> <filename2> <filenameN>
```

where your files have placeholders in the format `$ENVIRONMENT_VARIABLE_NAME` or `${ENVIRONMENT_VARIABLE_NAME}`.

Examples

```
replace_envs properties.json deploy.json run.json
```

retry_command

Description

Execute any command up to three times if it returns a non-zero error code. This is useful when you need to execute a command that can be flaky as a result of network hiccups, for example.

Usage

Bash	`retry_command <shell command>`
PowerShell	`retry_command <shell command>`

- `shell command` is the command to be retried

Bash Example

```
retry_command docker push mydirectory/myImage
```

PowerShell Example

```
retry_command docker push mydirectory/myImage
```

get_uuid

Description

Puts a uuid to stdout. Uses `/proc/sys/kernel/random/uuid` if available and falls back to `uuidgen` if not. The function calls `exit 1` if neither of these are available.

Usage

Bash	`get_uuid`

Bash Example

```
my_uuid=$(get_uuid)
```

save_artifact_info

Description

Saves metadata about an artifact. When saved, this metadata is used to enable signed pipelines for the artifacts.

Usage

Bash	`save_artifact_info <artifact type> <file path> [--build-name <build name> --build-number <build number> --project-key <project key>]`
PowerShell	`save_artifact_info <artifact type> <file path> [-build-name <build name> -build-number <build number> -release-bundle-name <name> -release-bundle-version <version> -project-key <project key>]`

- **`artifact type`**: This is the type of artifact. Either `file`, `buildInfo`, or `releaseBundle`.
- **`file path`**: This is the path to the metadata file to be saved.

Bash	Powershell	Description
--build-name	**-build-name**	This is name of the build. Required when artifact type is `buildInfo`.
--build-number	**-build-number**	This is number of the build. Required when artifact type is `buildInfo`.
--release-bundle-name	**-release-bundle-name**	This is name of the Release Bundle. Required when artifact type is `releaseBundle`.
--release-bundle-version	**-release-bundle-version**	This is version of the Release Bundle. Required when artifact type is `releaseBundle`.
--`project-key`	**-`project-key`**	(optional) Defaults to the environment's `project_key`. Can be specified to save info for a different project.

Bash Example for buildInfo

```
save_artifact_info 'buildInfo' './myBuildDetailedSummary.json' --build-name myBuild --build-number 42
```

Bash Example for file

```
save_artifact_info 'file' './myFileDetailedSummary.json'
```

PowerShell Example for buildInfo

```
save_artifact_info buildInfo myBuildDetailedSummary.json -build-name myBuild -build-number 42
```

PowerShell Example for file

```
save_artifact_info file myFileDetailedSummary.json
```

PowerShell Example for releaseBundle

```
save_artifact_info releaseBundle myFileDetailedSummary.json -release-bundle-name myBundle -release-bundle-version 1
```

validate_artifact

Description

Validates the signature of an artifact. Requires signed pipelines to be enabled.

Usage

Bash	`validate_artifact <artifact type> <file path> [--build-name <build name> --build-number <build number> --project-key <project key>]`
PowerShell	`validate_artifact <artifact type> <file path> [-build-name <build name> -build-number <build number> -project-key <project key>]`

- **`artifact type`**: This is the type of artifact. Either `file` or `buildinfo`.
- **`file path`**: This is the path to the metadata file to be validated.

Bash	Powershell	Description
--build-name	**-build-name**	This is name of the build. Required when artifact type is `buildInfo`.
--build-number	**-build-number**	This is number of the build. Required when artifact type is `buildInfo.`

--project-key	**-project-key**	(optional) Defaults to the environment's `project_key`. Can be specified to validate info for a different project.

Bash Example for buildInfo

```
validate_artifact 'buildInfo' './myBuildDetailedSummary.json' --build-name myBuild --build-number 42
```

Bash Example for file

```
validate_artifact 'file' './myFileDetailedSummary.json'
```

PowerShell Example for buildInfo

```
validate_artifact buildInfo myBuildDetailedSummary.json -build-name myBuild -build-number 42
```

PowerShell Example for file

```
validate_artifact file myFileDetailedSummary.json
```

configure_jfrog_cli

Description

Configures the JFrog CLI (v1) with the provided credentials, handling the different formats for different minor versions. Artifactory integrations listed in the integrations section of the step will be automatically configured, but this may be useful for resources or if the credentials are provided to the step in another way.

Usage

Bash	`configure_jfrog_cli --artifactory-url <url> [--user USER --apikey <key> --access-token <token> --server-name <name>]`
PowerShell	`configure_jfrog_cli -artifactory-url <url> [-user USER -apikey <key> -access-token <token> -server-name <name>]`

- **`artifactory-url`**: Required. The Artifactory URL.
- **user**: The user. Required when an API key is provided.
- **`apikey`**: An API key. Requires --user and may not be used with --access-token.
- **`access-token`**: An Access token. May not be used with --access-token or --user.
- **`server-name`**: Defaults to `default`. Can be specified to configure the CLI with that name.

Bash Example

```
configure_jfrog_cli --artifactory-url https://my-artifactory.local/artifactory --user me --
apikey AKCp8jQTdMTFdbkX14MJ8whtyV5Su5eawFhi2granysttMuyRPRA9FgxpfKzi1vFwDJXqYw9R --server-name
myAPIKeyArtifactory

configure_jfrog_cli --artifactory-url https://my-artifactory.local/artifactory --access-
token eyJ2ZXIiOiIyIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYiLCJraWQiOiJTMmVLcmRLdElWR2JnSTJWbVM2T0NQRDFHYW1ZbVp
wSXYtWS1meFN6TFJJIn0.
eyJzdWIiOiJqZmZlQDAwMFwvdXNlcnNcL2FkbWluIiwic2NwIjoiYXBwbGllZC1wZXJtaXNzaW9uc1wvYWRtaW4gYXBpOioiLCJhdWQiO
iJqZnJ0QCoiLCJpc3MiOiJqZmZlQDAwMCIsImV4cCI6MTYzMDY0NTM1OCwiaWF0IjoxNjMwNjQxNzU4LCJqdGkiOiI5YzcyMjMxOC01Zj
BmLTQ1MTEtYTY2Yi1lMzVhMDkyYmFlMWMifQ.
YOP6a2GmooC9nQcYROxGuN_1ZS7wt5YaTyqUs8ZIrUvkXxeMzLO0GI5q6KxFNIagVwetb5RP2tmLKm29HM7qp4HoK_EW5QnGQPZOY-
kOPm8Z1IzXKHaAKBpPttFkCKu1unj8bByRJL8mn63nG1gMORS2Eh2fvpzej3yyuRDnD65273AC5Qs4uNxp_4DqaqhgB_Xu0b2xzHAjqLK
BqWelICKBLcRbZQc4oSvphw2lj10wQHFqedOFej0akNaz8WGUo28l4GlJR7uPzQjAO0VWarKC_U3JySGSM_SAxH1z72AVHAErHCzM7tr5
gDbRsfRCuAHok66PeLiDAQxfXmsYZg --server-name myAccessTokenArtifactory
```

PowerShell Example

```
configure_jfrog_cli -artifactory-url https://my-artifactory.local/artifactory -user me -
apikey AKCp8jQTdMTFdbkX14MJ8whtyV5Su5eawFhi2granysttMuyRPRA9FgxpfKzi1vFwDJXqYw9R -server-name
myAPIKeyArtifactory

configure_jfrog_cli -artifactory-url https://my-artifactory.local/artifactory -access-
token eyJ2ZXIiOiIyIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYiLCJraWQiOiJTMmVLcmRLdElWR2JnSTJWbVM2T0NQRDFHYW1ZbVp
wSXYtWS1meFN6TFJJIn0.
eyJzdWIiOiJqZmZlQDAwMFwvdXNlcnNcL2FkbWluIiwic2NwIjoiYXBwbGllZC1wZXJtaXNzaW9uc1wvYWRtaW4gYXBpOioiLCJhdWQiO
iJqZnJ0QCoiLCJpc3MiOiJqZmZlQDAwMCIsImV4cCI6MTYzMDY0NTM1OCwiaWF0IjoxNjMwNjQxNzU4LCJqdGkiOiI5YzcyMjMxOC01Zj
BmLTQ1MTEtYTY2Yi1lMzVhMDkyYmFlMWMifQ.
YOP6a2GmooC9nQcYROxGuN_1ZS7wt5YaTyqUs8ZIrUvkXxeMzLO0GI5q6KxFNIagVwetb5RP2tmLKm29HM7qp4HoK_EW5QnGQPZOY-
kOPm8Z1IzXKHaAKBpPttFkCKu1unj8bByRJL8mn63nG1gMORS2Eh2fvpzej3yyuRDnD65273AC5Qs4uNxp_4DqaqhgB_Xu0b2xzHAjqLK
BqWelICKBLcRbZQc4oSvphw2lj10wQHFqedOFej0akNaz8WGUo28l4GlJR7uPzQjAO0VWarKC_U3JySGSM_SAxH1z72AVHAErHCzM7tr5
gDbRsfRCuAHok66PeLiDAQxfXmsYZg -server-name myAccessTokenArtifactory
```

`cleanup_jfrog_cli`

Description

Removes configuration for the JFrog CLI (v1), handling the different formats for different minor versions. Artifactory integrations listed in the integrations section of the step will be automatically removed at the end of the step, but this may be useful to remove the credentials earlier or when using `configure_jfrog_cli`.

Usage

Bash	`cleanup_jfrog_cli [--server-name <name>]`
PowerShell	`cleanup_jfrog_cli [-server-name <name>]`

- **`server-name`**: Defaults to `default`. Can be specified to remove that configuration.

Bash Example

```
cleanup_jfrog_cli --server-name myArtifactory
```

PowerShell Example

```
configure_jfrog_cli -artifactory-url https://my-artifactory.local/artifactory -user me -
apikey AKCp8jQTdMTFdbkX14MJ8whtyV5Su5eawFhi2granysttMuyRPRA9FgxpfKzi1vFwDJXqYw9R -server-name
myAPIKeyArtifactory

configure_jfrog_cli -artifactory-url https://my-artifactory.local/artifactory -access-
token eyJ2ZXIiOiIyIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYiLCJraWQiOiJTMmVLcmRLdElWR2JnSTJWbVM2T0NQRDFHYW1ZbVp
wSXYtWS1meFN6TFJJIn0.
eyJzdWIiOiJqZmZlQDAwMFwvdXNlcnNcL2FkbWluIiwic2NwIjoiYXBwbGllZC1wZXJtaXNzaW9uc1wvYWRtaW4gYXBpOioiLCJhdWQiO
iJqZnJ0QCoiLCJpc3MiOiJqZmZlQDAwMCIsImV4cCI6MTYzMDY0NTM1OCwiaWF0IjoxNjMwNjQxNzU4LCJqdGkiOiI5YzcyMjMxOC01Zj
BmLTQ1MTEtYTY2Yi1lMzVhMDkyYmFlMWMifQ.
YOP6a2GmooC9nQcYROxGuN_1ZS7wt5YaTyqUs8ZIrUvkXxeMzLO0GI5q6KxFNIagVwetb5RP2tmLKm29HM7qp4HoK_EW5QnGQPZOY-
kOPm8Z1IzXKHaAKBpPttFkCKu1unj8bByRJL8mn63nG1gMORS2Eh2fvpzej3yyuRDnD65273AC5Qs4uNxp_4DqaqhgB_Xu0b2xzHAjqLK
BqWelICKBLcRbZQc4oSvphw2lj10wQHFqedOFej0akNaz8WGUo28l4GlJR7uPzQjAO0VWarKC_U3JySGSM_SAxH1z72AVHAErHCzM7tr5
gDbRsfRCuAHok66PeLiDAQxfXmsYZg -server-name myAccessTokenArtifactory
```

Source Control

compare_git

Lists the files/directories containing changes within a commit range. This function is useful when building a monorepo (monolithic repository) to determine which services have changes.

Bash	`compare_git [--path | --resource] [options]`
PowerShell	`compare_git [-path | -resource] [options]`

- `path` is the file system path to a git repository.
- `resource` is the name of the gitRepo resource.
- `commit-range` option specifies the range of commits to look for changes (Example: `HEAD~1..HEAD`).
- `directories-only` option lists only the directories containing changes.
- `depth` option returns file/folder at certain depth. Root directory has depth value 1.

Bash Examples

```
ubuntu:~$ compare_git --path ./microservices --commit-range HEAD~2..HEAD
api/common/utilities/leftpad.js
api/main.js
notifier/main.js

ubuntu:~$ compare_git --path ./microservices --commit-range HEAD~2..HEAD --directories-only
api/common/utilities/
api/
notifier/

ubuntu:~$ compare_git --path ./microservices --commit-range HEAD~2..HEAD --directories-only --depth 1
api
notifier

ubuntu:~$ compare_git --path ./microserv --commit-range HEAD~2..HEAD --directories-only --depth 1

git repository not found at path: ./microserv
```

PowerShell Examples

```
PS C:\Users\windowsuser> compare_git -path microservices -commit-range HEAD~2..HEAD
api/common/utilities/leftpad.js
api/main.js
notifier/main.js

PS C:\Users\windowsuser> compare_git -path microservices -commit-range HEAD~2..HEAD -directories-only
api/common/utilities/
api/
notifier/

PS C:\Users\windowsuser> compare_git -path microservices -commit-range HEAD~2..HEAD -directories-only -
depth 1
api
notifier

PS C:\Users\windowsuser> compare_git -path microserv -commit-range HEAD~2..HEAD -directories-only -depth
1

git repository not found at path: ./microserv
```

update_commit_status

Description

Updates the statusof the commit on the source provider. Status options are *processing*, *success*, and *failure*.

Usage

Bash	`update_commit_status <gitRepo resource name> --status <status> --message <message> --context <context>`
PowerShell	`update_commit_status <gitRepo resource name> -status <status> -message <message> -context <context>`

update_commit_status <gitRepo resource name> --status <status> --message <message> --context <context>

- `gitRepo resource name` is the name of the gitRepo resource.
- `status` is the status to be set on the source provider: *processing*, *success*, or *failure*.
  If no status is specified:
  - *processing* will be assumed in `onStart` and `onExecute`
  - *success* will be assumed in `onSuccess`
  - *failure* will be assumed in `onFailure`
- `message` is the message (description) string to send with the status.
  If no message is specified, the default message will be "Step <status> in pipeline $pipeline_name"
- `context` is the context (key) for the `status`. The source provider will retain only the latest status received for that context.
  If no context is specified, the default is "$pipeline_name_$step_name"

Bash Examples

update_commit_status myGitRepoResource

update_commit_status myGitRepoResource --status processing --message “my description” --context $step_name

PowerShell Examples

```
update_commit_status myGitRepoResource

update_commit_status myGitRepoResource -status processing -message “my description” -context $step_name
```

Test Reports

save_tests

Description

Copies test reports given as input to later be parsed and uploaded (if file storage is available).

Usage

`save_tests <file or directory>`

- `file` or `directory` specifies either a filename for the test report file, or a directory name for a directory of test report files

Examples

```
save_tests testreport.xml
```

Encryption

encrypt_string

Description

Uses the provided public key to encrypt the specified string.

Usage

Bash	`encrypt_string --key <path> <source string>`
PowerShell	`encrypt_string -key <path> <source string>`

- `key` is the fully qualified path of the public key file
- `source string` is the string to be encrypted

Bash Example

```
ubuntu:~$ encrypt_string --key pub.pem "admin:passw0rd"
LbuD69yxEC6wi1M2B+/06ZY0vS+VcahcWeHovWi8GnUC04zBFrFXBmkAbG4TWRRvwbROgSsj2fo+
06SsaSPnb8fZDKuFP6Z89yJnLsh8UCq3gUbvFcGtgTQHUzIHQ0PVfrKrk9IyIvJex6+0ZPkVqa6t
dVK0X/5pCTWBKk5nIvw=
```

PowerShell Example

```
PS C:\Users\windowsuser> encrypt_string "admin:passw0rd" -key pub.pem
LbuD69yxEC6wi1M2B+/06ZY0vS+VcahcWeHovWi8GnUC04zBFrFXBmkAbG4TWRRvwbROgSsj2fo+
06SsaSPnb8fZDKuFP6Z89yJnLsh8UCq3gUbvFcGtgTQHUzIHQ0PVfrKrk9IyIvJex6+0ZPkVqa6t
dVK0X/5pCTWBKk5nIvw=
```

decrypt_string

Description

Uses the provided private key to decrypt the specified string.

This is typically used to decrypt information that was encrypted using `encrypt_string` with the corresponding public key. It helps you avoid building your own encrypt-decrypt system.

Usage

Bash	`decrypt_string --key <path> <encrypted string>`
PowerShell	`decrypt_string -key <path> <encrypted string>`

- `key` is the fully qualified path of the private key file
- `encrypted string` is the string to be decrypted

Bash Examples

```
ubuntu:~$ decrypt_string --key key.pem LbuD69yxEC6wi1M2B+
/06ZY0vS+VcahcWeHovWi8GnUC04zBFrFXBmkAbG4TWRRvwbROgSsj2fo+06SsaSPnb8fZDKuFP6Z89yJnLsh8UCq3gUbvFcGtgTQHUzIH
Q0PVfrKrk9IyIvJex6+0ZPkVqa6tdVK0X/5pCTWBKk5nIvw=

admin:passw0rd



ubuntu:~$ decrypt_string LbuD69yxEC6wi1M2B+
/06ZY0vS+VcahcWeHovWi8GnUC04zBFrFXBmkAbG4TWRRvwbROgSsj2fo+06SsaSPnb8fZDKuFP6Z89yJnLsh8UCq3gUbvFcGtgTQHUzI
HQ0PVfrKrk9IyIvJex6+0ZPkVqa6tdVK0X/5pCTWBKk5nIvw=
decrypt_string: ERROR - Key file not found
```

PowerShell Examples

```
PS C:\Users\windowsuser> decrypt_string -key key.pem LbuD69yxEC6wi1M2B+
/06ZY0vS+VcahcWeHovWi8GnUC04zBFrFXBmkAbG4TWRRvwbROgSsj2fo+06SsaSPnb8fZDKuFP6Z89yJnLsh8UCq3gUbvFcGtgTQHUzI
HQ0PVfrKrk9IyIvJex6+0ZPkVqa6tdVK0X/5pCTWBKk5nIvw=

admin:passw0rd



PS C:\Users\windowsuser> decrypt_string LbuD69yxEC6wi1M2B+
/06ZY0vS+VcahcWeHovWi8GnUC04zBFrFXBmkAbG4TWRRvwbROgSsj2fo+06SsaSPnb8fZDKuFP6Z89yJnLsh8UCq3gUbvFcGtgTQHUzI
HQ0PVfrKrk9IyIvJex6+0ZPkVqa6tdVK0X/5pCTWBKk5nIvw=
key is mandatory, please provide a value.
```

encrypt_file

Description

Uses the provided public key to encrypt the specified file to a new file.

Usage

Bash	`encrypt_file --key <path> [--output <filename>] <source filename>`
PowerShell	`encrypt_file -key <path> [-output <filename>] <source filename>`

- `key` is the fully qualified path of the public key file
- `output` is the name of the resulting encrypted file. Defaults to “encrypted”
- `source filename` is the file to be decrypted

Bash Examples

```
ubuntu:~$ cat secrets.json
{"TOP_SECRET_CONFIDENTIAL_FORMULA": "uuddlrlrbas"}
ubuntu:~$ encrypt_file secrets.json --key pub.pem
encrypt_file: Encrypting secrets.json using key pub.pem
encrypt_file: Encrypted secrets.json to encrypted
ubuntu:~$ cat encrypted
dGpLiI2IORfkRpSEhhuN9A8U/dQLlyHD6EKermRM5bnoIBcM7TWpLU3Y53f4zsAKTSmmQKHucPxJ
YFOVc6F1AWItYIgTSp2dEY4ugIZ7uTn/IIa0qU7EGUREtPyrdu9N5phS2UybTn0u80CSP7Bf/HF+
5xHBCvlenuFanIFLDbo=
```

PowerShell Examples

```
PS C:\Users\windowsuser> type secrets.json
{"TOP_SECRET_CONFIDENTIAL_FORMULA": "uuddlrlrbas"}
PS C:\Users\windowsuser> encrypt_file secrets.json -key pub.pem
encrypt_file: Encrypting secrets.json using key pub.pem
encrypt_file: Encrypted secrets.json to encrypted
PS C:\Users\windowsuser> type encrypted
dGpLiI2IORfkRpSEhhuN9A8U/dQLlyHD6EKermRM5bnoIBcM7TWpLU3Y53f4zsAKTSmmQKHucPxJ
YFOVc6F1AWItYIgTSp2dEY4ugIZ7uTn/IIa0qU7EGUREtPyrdu9N5phS2UybTn0u80CSP7Bf/HF+
5xHBCvlenuFanIFLDbo=
```

decrypt_file

Description

Uses the provided private key to decrypt the specified file to a new file.

This is typically used to decrypt information that was encrypted using `encrypt_file` with the corresponding public key. It helps you avoid building your own encrypt-decrypt system.

Usage

Bash	`decrypt_file --key <path> [--output <filename>] <source filename>`
PowerShell	`decrypt_file -key <path> [-output <filename>] <source filename>`

- `key` is the fully qualified path of the private key file
- `output` is the name of the resulting decrypted file. Defaults to “decrypted”
- `source filename` is the file to be decrypted

Bash Examples

```
ubuntu:~$ cat encrypted
dGpLiI2IORfkRpSEhhuN9A8U/dQLlyHD6EKermRM5bnoIBcM7TWpLU3Y53f4zsAKTSmmQKHucPxJ
YFOVc6F1AWItYIgTSp2dEY4ugIZ7uTn/IIa0qU7EGUREtPyrdu9N5phS2UybTn0u80CSP7Bf/HF+
5xHBCvlenuFanIFLDbo=


ubuntu:~$ decrypt_file encrypted --key key.pem
decrypt_file: Decrypting encrypted using key key.pem
decrypt_file: Decrypted encrypted to decrypted


ubuntu:~$ cat decrypted
{"TOP_SECRET_CONFIDENTIAL_FORMULA": "uuddlrlrbas"}



ubuntu:~$ decrypt_file encrypted --key key.pem --output secrets.json
decrypt_file: Decrypting encrypted using key key.pem
decrypt_file: Decrypted encrypted to secrets.json
```

PowerShell Examples

```
PS C:\Users\windowsuser> type encrypted
dGpLiI2IORfkRpSEhhuN9A8U/dQLlyHD6EKermRM5bnoIBcM7TWpLU3Y53f4zsAKTSmmQKHucPxJ
YFOVc6F1AWItYIgTSp2dEY4ugIZ7uTn/IIa0qU7EGUREtPyrdu9N5phS2UybTn0u80CSP7Bf/HF+
5xHBCvlenuFanIFLDbo=


PS C:\Users\windowsuser> decrypt_file encrypted -key key.pem
decrypt_file: Decrypting encrypted using key key.pem
decrypt_file: Decrypted encrypted to decrypted


PS C:\Users\windowsuser> type decrypted
{"TOP_SECRET_CONFIDENTIAL_FORMULA": "uuddlrlrbas"}


PS C:\Users\windowsuser> decrypt_file encrypted -key key.pem -output secrets.json
decrypt_file: Decrypting encrypted using key key.pem
decrypt_file: Decrypted encrypted to secrets.json
```

Notifications

send_notification

Description

Utilizes notification integration to send custom messages at any time during the build to any recipient.

Usage

```
send_notification <integration> [options]
```

The options can be specified as part of the command, or defined as environment variables before the command is issued.

The command line arguments take priority over the environment variables.

Command line options

AirBrake

Creates an AirBrake deployment through an Airbrake Integration. Not supported in PowerShell.

Bash	Option Description
--project-id	the project ID to send the notification for
--environment	the environment value to use when posting the deployment
--username	used when posting an AirBrake deployment
--email	the email to be used when posting the AirBrake deployment
--repository	the repository to use when posting the AirBrake deployment
--revision	the deployment revision
--version	the version to use when posting the AirBrake deployment
--type	currently only type “deploy” is supported
--description	description of the deployment
--payload	path to a valid JSON file that contains a payload to use to POST the AirBrake deployment

Jira

Creates a Jira issue (also known as a ticket).

Bash	Powershell	Option Description
--project-id	**-project-id**	the Project Key of the project to associate the new issue with. The project key is the short string that begins all issue numbers for the project (e.g., "EXAMPLE-1234")
--type	**-type**	the issue type for the new issue (e.g., "Bug", "Task", etc.). This string must be one of the recognized Jira issue types
--summary	**-summary**	a string for the new issue's Summary field (it's title)
--description	**--description**	(optional) a string for the new issue's Description field
--attach-file	**--attach-file**	(optional) a path to a file that you'd like to attach to the issue

NewRelic

Creates a NewRelic deployment through a NewRelic Integration. Not supported in PowerShell.

Bash	Option Description
--type	the type of object to be posted. At the moment, only "deployment" is supported
--description	description of the deployment
--username	the user recording the deployment. Defaults to "JFrog Pipelines"
--changelog	the changelog value to use in the deployment
--revision	the deployment revision (required)
--appId	the ID of the app being deployed. If not provided, --appName must be present
--appName	the name of the app being deployed. If not provided, --appId must be present
--payload	path to a valid JSON file that contains a payload to use to POST the NewRelic deployment

PagerDuty Events

Sends an event through a PagerDuty Events Integration.

Bash	Powershell	Option Description
--text	**-text**	The main text to display in the event on PagerDuty.

Slack

Sends a message on Slack through a Slack Integration.

Bash	Powershell	Option Description
--payload	**-payload**	(optional) A path to a valid json file to act as the payload of the message. If a payload is provided, all other parameters are ignored. This payload is directly sent to Slack, so please view the Slack API documentation for information on how the payload should be formatted.
--username	**-username**	(optional) shows in the heading of the Slack message
--pretext	**-pretext**	(optional) a string that becomes the first part of the Slack message. Defaults to current date/time
--text	**-text**	(optional) the main text to display in the message.
--color	**-color**	(optional) hex string that changes the color of the status bar to the left of the Slack message.

--recipient	**-recipient**	(optional) the target of the message. Should start with “@” or “#” for user or channel, respectively.
--icon-url	**-icon-url**	(optional) the url of the icon to show next to the message

smtpCreds (email)

Sends an email through an SMTP Credentials Integration.

Bash	Powershell	Option Description
--recipients	**-recipients**	one or more email addresses
--subject	**-subject**	(optional) add a message to the subject. Does not replace the default subject
--body	**-body**	(optional) specify some text to add to the body of the email. Does not replace the existing body information
--status	**-status**	(optional) can be set to a valid status string. By default it will be set based on the section of scrip the command is executed in.
--attachments	**-attachments**	(optional) a list of files to attach to the email. Combined total of all files cannot exceed 5MB
--attach-logs	**-attach-logs**	(optional) ‘true’ or ‘false’. Defaults to false. All available logs for the step will be attached to the email. Note that it can only attach logs that have already been created, so using this option in the onStart section, for example, would not have very detailed logs.
--show-failing-commands	**-show-failing-commands**	(optional) ‘true’ or ‘false’. Defaults to false. The existing logs for the step will be parsed. Any failed command that is detected will be added to the body of the email, along with up to 100 preceding lines (if printed from the same command)

Environment Options

All of the above options can also be included as environment variables instead of arguments. The command line argument will have priority over the environment. Here is the full list of ENVs:

- `NOTIFY_USERNAME` (--username/-username)
- `NOTIFY_PASSWORD` (--password/-password)
- `NOTIFY_RECIPIENT` (--recipient/-recipient)
- `NOTIFY_PRETEXT` (--pretext/-pretext)
- `NOTIFY_TEXT` (--text/-text)
- `NOTIFY_COLOR` (--color/-color)
- `NOTIFY_ICON_URL` (--icon-url/-icon-url)
- `NOTIFY_PAYLOAD` (--payload/-payload)
- `NOTIFY_TYPE` (--type/-type)
- `NOTIFY_PROJECT_ID` (--project-id/-project-id)
- `NOTIFY_ENVIRONMENT` (--environment/-environment)
- `NOTIFY_REVISION` (–revision/-revision)
- `NOTIFY_SUMMARY` (--summary/-summary)
- `NOTIFY_ATTACH_FILE` (--attach-file/-attach-file)
- `NOTIFY_REPOSITORY` (--repository/-repository)
- `NOTIFY_EMAIL` (–email--email/-email)
- `NOTIFY_STATUS` (--status/-status)
- `NOTIFY_VERSION` (--version/-version)
- `NOTIFY_CHANGELOG` (--changelog/-changelog)
- `NOTIFY_DESCRIPTION` (–description--description/-description)
- `NOTIFY_ATTACHMENTS` (–attachments--attachments/-attachments)
- `NOTIFY_ATTACH_LOGS` (--attach-logs/-attach-logs)
- `NOTIFY_SHOW_FAILING_COMMANDS` (--show-failing-commands/-show-failing-commands)
- `NOTIFY_SUBJECT` (--subject/-subject)
- `NOTIFY_BODY` (–body)

JSON

set_payload

Description

Sets an optional JSON payload (string or file) for an OutgoingWebhook resource. When the OutgoingWebhook is specified in a step's `outputresources` the payload is sent when the step is complete.

Usage

Bash	`set_payload <resource> <payload> [--file]`
PowerShell	`set_payload <resource> <payload> [-file]`

- `resource` is the name of an OutgoingWebhook resource.
- `payload` is a JSON string or file to attach to the resource that will be sent as part of the outgoing webhook. A file can be specified as a path relative to the current directory, absolute path, or path relative to the step workspace directory.
- `file` option specifies that the `payload` parameter is a file. If not specified, `payload` will be processed as a string.

Bash Examples

```
set_payload myExternalHookResource "{\"test\":\"payload\"}"

echo "{\"test\":\"payload\"}" > testpayload.json
set_payload myExternalHookResource testpayload.json --file
```

PowerShell Examples

```
set_payload myExternalHookResource "{`"test`":`"payload`"}"

Set-Content -Path testpayload.json -Value "{`"test`":`"payload`"}"
set_payload myExternalHookResource testpayload.json -file
```

read_json

Description

Extracts the json property value from the specified file.

This simplifies handling of a JSON file to read specific property values that are required for your workflow.

> ⚠ **Not supported in PowerShell**
>
> In PowerShell, `ConvertTo-Json` is suggested as an alternative.

Usage

```
read_json <path to file> <field name>
```

- `path to file` is the fully qualified path of the JSON file
- field name is the field for which you want to read the value. Use dot notation and [n] for arrays.

Examples

```
ubuntu:~$ cat secrets.json
{
 "TOP_SECRET_CONFIDENTIAL_FORMULA": "uuddlrlrbas",
 "LESSER_SECRET_FORMULA": "dyddy",
 "TOP_5_PASSWORDS_LIST": [
   "admin",
   "passw0rd",
   "testing123",
   "correcthorsebatterystaple"
 ],
   “nesting”: {
      “is”: {
      “fun”: “yay”
      }
   }
}

ubuntu:~$ read_json secrets.json "LESSER_SECRET_FORMULA"
dyddy

ubuntu:~$ read_json secrets.json "TOP_5_PASSWORDS_LIST[3]"
correcthorsebatterystaple

ubuntu:~$ read_json secrets.json "nesting.is.fun"
yay
```

Resources

replicate_resource

Description

This command takes an input resource and creates an exact copy. This helps you to transfer metadata from one step to the next.

Usage

Bash	`replicate_resource <from_resource> <to_resource> [--options]`
PowerShell	`replicate_resource <from_resource> <to_resource> [-options]`

- `from_resource` is the name of the `inputResources` resource that you're copying from.
- `to_resource` is the name of the `outputResources` resource that will receive the replicated data from the `from_resource`. Any pre-existing files or key-value pairs in the `to_resource` will be replaced.
- `match-settings` option should be set when you want the replication to adhere to any branch/tag settings in the `to_resource`. For example, If your `from_resource` gitRepo can trigger on both commits and pull requests, but you only want to update your `to_resource` on commits, you can replicate with --match-settings, and the `to_resource` will only be updated when the `from_resource` had a commit.

Bash Examples

```
replicate_resource myRepo1 myRepo2 --match-settings
replicate_resource myTestImage myStageImage
```

PowerShell Examples

```
replicate_resource myRepo1 myRepo2 -match-settings
replicate_resource myTestImage myStageImage
```

write_output

Description

Adds data to an output resource in the form of key/value pairs that will become properties of the resource.

Usage

`write_output <resource> <key value pair>...`

- `resource` is the resource to update
- `key value pair` is a single string with a key and a value, separated by an "=". Multiple of these strings can be supplied as input. A value with spaces should be surrounded by quotes.

Examples

```
write_output myImage imageTag=master sha=$commitSha description=\"hello world\"
```

The newly attached properties can be accessed as environment variables of the form `res_{Resource Name}_{Key Name}`.

For example, the above created properties can be accessed as these environment variables:

```
$ printenv res_myImage_master
master
$ printenv res_myImage_sha
d6cd1e2bd19e03a81132a23b2025920577f84e37
$ printenv res_myImage_description
"hello world"
```

Caching

Caching helps you speed up execution of your steps by preserving and restoring packages and dependencies between runs of a step. In this way, you can reduce build times by avoiding repeating the installation or loading of large dependencies.

add_cache_files

Description

Copies files given as input to later be uploaded if file storage is available.

Usage

`add_cache_files <file or directory> <name>`

- `file or directory` is a file or directory to store in the cache
- `name` is a name to give the stored file or directory (without spaces)

Examples

```
add_cache_files cachefile.txt my_file

add_cache_files directory/subdirectory my_directory
```

restore_cache_files

Description

Copies stored cache (if file storage is available) to the specified location. No error will occur if nothing is available for <name> in the cache.

Usage

`restore_cache_files <name> <path>`

- `name` is the name the file or driectory to be restored was given when cached.
- `path` is a path at which to place the file or directory.

Examples

```
restore_cache_files my_file cachefile.txt

restore_cache_files my_directory directory/subdirectory
```

Run State Management

add_run_variables

Description

Allows you to add environment variables that will be available in the following steps of the run.

If the following variables are set, they will be used:

- `JFROG_CLI_BUILD_NAME`: If set, the pipeline uses this value instead of the default pipeline name for the build info collected.
- `JFROG_CLI_BUILD_NUMBER`: If set, the pipeline uses this value instead of the default run number for the build info collected.
- `USE_LOCAL_JFROG_CLI`: If set as `true`, the local JFrog CLI on the host or in the image (depending on `runtime` configuration) is used instead of the version packaged with JFrog Pipelines. This is not recommended and native steps may not be able to run with the local JFrog CLI version.

Usage

```
add_run_variables <key value pair>...
```

- `key value pair` is a single string with a key and a value, separated by an "=". Multiple of these strings can be supplied as input. Each value will be exported as an environment variable at the time this command is used and automatically in any later steps within the run.

Examples

```
add_run_variables imageTag="master"
```

export_run_variables

Description

Sources the file containing the run variables. This will be done automatically, but may also be used to "reset" the environment variables in the current step.

Usage

```
export_run_variables
```

Examples

```
export_run_variables
```

add_run_files

Description

Copies files given as input into the run state for use in later steps in the run, if file storage is available.

Usage

```
add_run_files <file or directory> <name>
```

`file or directory` is a file or directory to store in the run state

`name` is a name to give the stored file or directory (without spaces). This cannot be run.env.

Examples

```
add_run_files cachefile.txt my_file

add_run_files directory/subdirectory my_directory

add_run_files directory/*/subdirectory my_directory
```

restore_run_files

Description

Copies files stored in the run state (if file storage is available) to the specified location. No error will occur if nothing is available for <name> in the run state.

Usage

```
restore_run_files <name> <path>
```

- `path` is the name the files to be restored were given when added to the run state.
- `file or directory` is a path at which to place the file or files.

Examples

```
restore_run_files my_file cachefile.txt

restore_run_files my_directory directory/subdirectory
```

Pipeline State Management

add_pipeline_variables

Description

Allows you to add environment variables that will be available in the following steps of the run and in future runs. These variables may be overridden by another variable with the same key added to the current run.

If the following variables are set, they will be used:

- `JFROG_CLI_BUILD_NAME`: If set, the pipeline uses this value instead of the default pipeline name for the build info collected.
- `JFROG_CLI_BUILD_NUMBER`: If set, the pipeline uses this value instead of the default run number for the build info collected.
- `USE_LOCAL_JFROG_CLI`: If set as `true`, the local JFrog CLI on the host or in the image (depending on `runtime` configuration) is used instead of the version packaged with JFrog Pipelines. This is not recommended and native steps may not be able to run with the local JFrog CLI version.

Usage

```
add_pipeline_variables <key value pair>...
```

- `key value pair` is a single string with a key and a value, separated by an "=". Multiple of these strings can be supplied as input. Each value will be exported as an environment variable at the time this command is used and automatically in any steps that start after this run is complete.

Examples

```
add_pipeline_variables imageTag="master"

add_pipeline_variables imageName="myimage" imageTag="master"
```

export_pipeline_variables

Description

Sources the file containing the pipeline variables. This will be done automatically, but may also be used to "reset" the environment variables in the current step.

Usage

`export_pipeline_variables`

Examples

```
export_pipeline_variables
```

add_pipeline_files

Description

Copies files given as input into the pipeline state for use in later steps in the run and future runs, if file storage is available.

Usage

`add_pipeline_files <file or directory> <name>`

- `file or directory` is a file or directory to store in the pipeline state.
- `name` is a name to give the stored file or directory (without spaces). This cannot be pipeline.env.

Examples

```
add_pipeline_files cachefile.txt my_file

add_pipeline_files directory/subdirectory my_directory
```

restore_pipeline_files

Description

Copies files stored in the pipeline state (if file storage is available) to the specified location. No error will occur if nothing is available for <name> in the run state.

Usage

`restore_pipeline_files <name> <path>`

- `name` is the name the file to be restored was given when added to the pipeline state.
- `path` is a path at which to place the file or files.

Examples

```
restore_pipeline_files my_file cachefile.txt

restore_pipeline_files my_directory directory/subdirectory
```

Step Properties

find_resource_variable

Description

Retrieves the value of the named property of a resource.

Usage

`find_resource_variable <resourceName> <propertyName>`

- `resourceName` is the name of the resource.
- `propertyName` is the name of the resource property whose value to retrieve.

Examples

```
find_resource_variable myGitHub commitSha

find_resource_variable myImage imageTag
```

get_integration_name

Description

Retrieves the name of the first integration found of the type specified. Available to extension steps to get the name of the first input integration of a particular type.

Usage

`get_integration_name --type <integration type>`

- `integration type` is the name of an Pipelines Integration type

Examples

```
get_integration_name --type Slack

get_integration_name --type "Docker Registry"
```

get_resource_name

Description

Retrieves the name of the first resource found of the type specified in `inputResources` or `outputResources`. Available to extension steps to get the name of the first input or output resource of a particular type.

Usage

get_resource_name --type <resource type> --operation <IN | OUT> --syntax-version <semver>

- `resource type` is the name of a Pipelines Resource type
- `IN | OUT` selects whether the resource is named in `inputResources` or `outputResources`
- `semver` is the semantic version number of the resource's syntax version

Examples

```
get_resource_name --type GitRepo --operation IN

get_resource_name --type BuildInfo --operation OUT

get_resource_name --type GitRepo --operation IN --syntax-version 1.5.0
```

get_resource_names

Description

Retrieves an array of names of the type specified in `inputResources` or `outputResources`. Available to extension steps to get the names of input or output resource of a particular type.

Usage

get_resource_names --type <resource type> --operation <IN | OUT> --syntax-version <semver>

- `resource type` is the name of a Pipelines Resource type
- `IN | OUT` selects whether the resource is named in `inputResources` or `outputResources`
- `semver` is the semantic version number of the resource's syntax version
- In PowerShell, a native PowerShell array is returned. In Bash, a JSON array is returned that can be handled with jq.

Examples

```
get_resource_names --type GitRepo --operation IN

get_resource_names --type BuildInfo --operation OUT

get_resource_names --type GitRepo --operation IN --syntax-version 1.5.0
```

find_step_configuration_value

Description

Retrieves the value of the `configuration` property for the currently executing step. If the property is a collection, the first value will be returned. Available to extension steps to get the value of a configuration.

Usage

```
find_step_configuration_value <propertyName>
```

- `propertyName` is the name of the step's configuration property whose value to retrieve

Examples
`find_step_configuration_value forceXrayScan`