Jenkins

> User Documentation Home

User Handbook

- User Handbook Overview
- Installing Jenkins
- Platform Information
- Using Jenkins
- Pipeline
- Blue Ocean
- Managing Jenkins
- Securing Jenkins
- System Administration
- Scaling Jenkins
- Troubleshooting Jenkins
- Glossary

Tutorials

- Guided Tour
- Jenkins Pipeline
- Using Build Tools

Resources

- Pipeline Syntax reference
- Pipeline Steps reference
- LTS Upgrade guides

The following plugin provides functionality available through Pipeline-compatible steps. Read more about how to integrate steps into your Pipeline in the Steps section of the Pipeline Syntax page.

For a list of other such plugins, see the Pipeline Steps Reference page.

Table of Contents

Pipeline Utility Steps

View this plugin on the Plugins site

`compareVersions`: Compare two version number strings

Compare two version numbers with each other. See VersionNumber.java for how version strings are handled.

The return value is an Integer;

- `-1` if `v1 < v2`
- `0` if `v1 == v2`
- `1` if `v1 > v2`

- `v1 : String`

  The version number string that will be compared to `v2`.

- `v2 : String`

  The version number string that `v1` will be compared to.

- `failIfEmpty : boolean` (optional)

  Fail the build if `v1` or `v2` is empty or `null`.

  By default the empty string or `null` is treated as the lowest version and will not fail the build. I.e.:

  - `null` compared to `null == 0`
  - empty compared to empty `== 0`
  - `null` compared to empty `== 0`
  - `null` or empty compared to anything `== -1`
  - anything compared to `null` or empty `== 1`

`findFiles`: Find files in the workspace

Find files in the current working directory. The step returns an array of file info objects who's properties you can see in the below example.
Ex: `def files = findFiles(glob: '**/TEST-*.xml') echo """${files[0].name} ${files[0].path} ${files[0].directory} ${files[0].length} ${files[0].lastModified}"""`

- `excludes : String` (optional)
- `glob : String` (optional)

  Ant style pattern of file paths that should match. If this property is set all descendants of the current working directory will be searched for a match and returned, if it is omitted only the direct descendants of the directory will be returned.

`nodesByLabel`: List of nodes by Label, by default excludes offline nodes.

Returns an array of `node` names with the given label.

- `label : String`
- `offline : boolean` (optional)

`prependToFile`: Create a file (if not already exist) in the workspace, and prepend given content to that file.

Creates a file if it does not already exist, and prepends given content to it.

- `file : String`

  The path to the file that will be prepended.

- `content : String`

  The content to prepend.

`readCSV`: Read content from a CSV file in the workspace.

Reads a file in the current working directory or a String as a plain text. A List of CSVRecord instances is returned

Example:

```
def records = readCSV file: 'dir/input.csv'
assert records[0][0] == 'key'
assert records[1][1] == 'b'

def records = readCSV text: 'key,value\na,b'
assert records[0][0] == 'key'
assert records[1][1] == 'b'
```

Advanced Example:

```
def excelFormat = CSVFormat.EXCEL
def records = readCSV file: 'dir/input.csv', format: excelFormat
assert records[0][0] == 'key'
assert records[1][1] == 'b'

def records = readCSV text: 'key,value\na,b', format: CSVFormat.DEFAULT.withHeader()
assert records[0].get('key') == 'a'
assert records[0].get('value') == 'b'
```

- `file : String` (optional)

  Path to a file in the workspace from which to read the CSV data. *Data is accessed as a List of String Arrays.*

  You can only specify `file` **or** `text`, not both in the same invocation.

- `format` (optional)
  - **Type:** `class org.apache.commons.csv.CSVFormat`
- `text : String` (optional)

  A string containing the CSV formatted data. *Data is accessed as a List of String Arrays.*

  You can only specify `file` **or** `text`, not both in the same invocation.

`readJSON`: Read JSON from files in the workspace.

Reads a file in the current working directory or a String as a plain text JSON file. The returned object is a normal Map with String keys or a List of primitives or Map.

Example:

```
def props = readJSON file: 'dir/input.json'
assert props['attr1'] == 'One'
assert props.attr1 == 'One'

def props = readJSON text: '{ "key": "value" }'
assert props['key'] == 'value'
assert props.key == 'value'

def props = readJSON text: '[ "a", "b"]'
assert props[0] == 'a'
assert props[1] == 'b'

def props = readJSON text: '{ "key": null, "a": "b" }', returnPojo: true
assert props['key'] == null
props.each { key, value ->
    echo "Walked through key $key and value $value"
}
```

- `file : String` (optional)

  Path to a file in the workspace from which to read the JSON data. *Data could be access as an array or a map.*

  You can only specify `file` **or** `text`, not both in the same invocation.

- `returnPojo : boolean` (optional)

  Transforms the output into a POJO type (`LinkedHashMap` or `ArrayList`) before returning it.

  By default deactivated (`false`), and a JSON object (`JSONObject` or `JSONArray` from json-lib) is returned.

- `text : String` (optional)

  A string containing the JSON formatted data. *Data could be access as an array or a map.*

  You can only specify `file` **or** `text`, not both in the same invocation.

`readManifest`: Read a Jar Manifest

Reads a Jar Manifest file or text and parses it into a set of Maps. The returned data structure has two properties: `main` for the main attributes, and `entries` containing each individual section (except for main).

Example:

```
        def man = readManifest file: 'target/my.jar'
        assert man.main['Version'] == '6.15.8'
        assert man.main['Application-Name'] == 'My App'
        assert man.entries['Section1']['Key1'] == 'value1-1'
        assert man.entries['Section2']['Key2'] == 'value2-2'
```

- `file : String` (optional)

  Optional path to a file to read. It could be a plain text, `.jar`, `.war` or `.ear`. In the latter cases the manifest will be extracted from the archive and then read.

  You can only specify `file` **or** `text`, not both in the same invocation.

- `text : String` (optional)

  Optional text containing the manifest data.

  You can only specify `file` **or** `text`, not both in the same invocation.

`readMavenPom`: Read a maven project file.

Reads a Maven project file. The returned object is a Model .

Avoid using this step and `writeMavenPom`. It is better to use the `sh` step to run `mvn` goals. For example:

```
def version = sh script: 'mvn help:evaluate -Dexpression=project.version -q -DforceStdout', returnStdout: true
```

- `file : String` (optional)

  Optional path to the file to read. *If left empty the step will try to read `pom.xml` in the current working directory.*

`readProperties`: Read properties from files in the workspace or text.

Reads a file in the current working directory or a String as a plain text Java Properties file. The returned object is a normal Map with String keys. The map can also be pre loaded with default values before reading/parsing the data.

Fields:

- `file`: Optional path to a file in the workspace to read the properties from. *These are added to the resulting map after the defaults and so will overwrite any key/value pairs already present.*
- `text`: An Optional String containing properties formatted data. *These are added to the resulting map after `file` and so will overwrite any key/value pairs already present.*
- `defaults`: An Optional Map containing default key/values. *These are added to the resulting map first.*
- `interpolate`: Flag to indicate if the properties should be interpolated or not.
  Prefix interpolations allowed by default are: `urlDecoder`, `urlEncoder`, `date`, `base64Decoder`, `base64Encoder`. Default prefix interpolations can be overridden by setting the system property:
  `org.jenkinsci.plugins.pipeline.utility.steps.conf.ReadPropertiesStepExecution.CUSTOM_PREFIX_INTERPOLATOR_LOOKUPS`
  Note that overriding default prefix interpolations can be insecure depending on which ones you enable. In case of error or cyclic dependencies, the original properties will be returned.

Example:

```
def d = [test: 'Default', something: 'Default', other: 'Default']
def props = readProperties defaults: d, file: 'dir/my.properties', text: 'other=Override'
assert props['test'] == 'One'
assert props['something'] == 'Default'
assert props.something == 'Default'
assert props.other == 'Override'
```

Example with interpolation:

```
def props = readProperties interpolate: true, file: 'test.properties'
assert props.url = 'http://localhost'
assert props.resource = 'README.txt'
// if fullUrl is defined to ${url}/${resource} then it should evaluate to http://localhost/README.txt
assert props.fullUrl = 'http://localhost/README.txt'
```

- `defaults` (optional)
  - **Type:** `java.util.Map<java.lang.Object, java.lang.Object>`
- `file : String` (optional)
- `interpolate : boolean` (optional)
- `text : String` (optional)

`readYaml`: Read yaml from files in the workspace or text.

- `codePointLimit : int` (optional)
- `file : String` (optional)
- `maxAliasesForCollections : int` (optional)
- `text : String` (optional)

`sha1`: Compute the SHA1 of a given file

Computes the SHA1 of a given file.

- `file : String`

  The path to the file to hash.

`sha256`: Compute the SHA256 of a given file

Computes the SHA256 of a given file.

- `file : String`

The path to the file to hash.

`tar`: Create Tar file

Create a tar/tar.gz file of content in the workspace.

- `file : String` (optional)

  The name/path of the tar file to create.

- `archive : boolean` (optional)

  If the tar file should be archived as an artifact of the current build. The file will still be kept in the workspace after archiving.

- `compress : boolean` (optional)

  The created tar file shall be compressed as gz.

- `defaultExcludes : boolean` (optional)
- `dir : String` (optional)

  The path of the base directory to create the tar from. Leave empty to create from the current working directory.

- `exclude : String` (optional)

  Ant style pattern of files to exclude from the tar.

- `glob : String` (optional)

  Ant style pattern of files to include in the tar. Leave empty to include all files and directories.

- `overwrite : boolean` (optional)

  If the tar file should be overwritten in case of already existing a file with the same name.

`tee`: Tee output to file

- `file : String`

`touch`: Create a file (if not already exist) in the workspace, and set the timestamp

Creates a file if it does not already exist, and updates the timestamp.

- `file : String`

  The path to the file to touch.

- `timestamp : long` (optional)

  The timestamp to set (number of ms since the epoc), leave empty for current system time.

`untar`: Extract Tar file

Extract a tar/tar.gz file in the workspace.

- `file : String` (optional)

  The name/path of the tar/tar.gz file to extract.

- `dir : String` (optional)

  The path of the base directory to extract the tar to. Leave empty to extract in the current working directory.

- `glob : String` (optional)

  Ant style pattern of files to extract from the tar. Leave empty to include all files and directories.

- `keepPermissions : boolean` (optional)

  Extract file permissions. *E.g.* `untar file: 'example.tgz', keepPermissions: true`

- `quiet : boolean` (optional)

  Suppress the verbose output that logs every single file that is dealt with. *E.g.* `untar file: 'example.tgz', quiet: true`

- `test : boolean` (optional)

  Test the integrity of the archive instead of extracting it. When this parameter is enabled, all other parameters *(except for file)* will be ignored. The step will return `true` or `false` depending on the result instead of throwing an exception.

`unzip`: Extract Zip file

Extract a zip file in the workspace.

- `zipFile : String`

  The name/path of the zip file to extract.

- `charset : String` (optional)

  Specify which Charset you wish to use eg. UTF-8

- `dir : String` (optional)

  The path of the base directory to extract the zip to. Leave empty to extract in the current working directory.

- `file : String` (optional)
- `glob : String` (optional)

  Ant style pattern of files to extract from the zip. Leave empty to include all files and directories.

- `quiet : boolean` (optional)

  Suppress the verbose output that logs every single file that is dealt with. *E.g.* `unzip zipFile: 'example.zip', quiet: true`

- `read : boolean` (optional)

  Read the content of the files into a Map instead of writing them to the workspace. The keys of the map will be the path of the files read. *E.g.* `def v = unzip zipFile: 'example.zip', glob: '*.txt', read: true String version = v['version.txt']`

- `test : boolean` (optional)

  Test the integrity of the archive instead of extracting it. When this parameter is enabled, all other parameters *(except for zipFile)* will be ignored. The step will return `true` or `false` depending on the result instead of throwing an exception.

`verifySha1`: Verify the SHA1 of a given file

Verifies the SHA1 of a given file.

- `file : String`

  The path to the file to hash.

- `hash : String`

  The expected hash.

`verifySha256`: Verify the SHA256 of a given file

Verifies the SHA256 of a given file.

- `file : String`

  The path to the file to hash.

- `hash : String`

  The expected hash.

`writeCSV`: Write content to a CSV file in the workspace.

Write a CSV file in the current working directory. That for example was previously read by `readCSV`. See CSVPrinter for details.

Fields:

- `records`: The list of CSVRecord instances to write.
- `file`: Path to a file in the workspace to write to.
- `format`: See CSVFormat for details.

Example:

```
def records = [['key', 'value'], ['a', 'b']]
writeCSV file: 'output.csv', records: records, format: CSVFormat.EXCEL
```

- `file : String`
- `records`
  - **Type:** `java.lang.Iterable<?>`
- `format` (optional)
  - **Type:** `class org.apache.commons.csv.CSVFormat`

`writeJSON`: Write JSON to a file in the workspace.

Write JSON to a file in the current working directory, or to a String.

Fields:

- `json`: The object to write. Can either be a JSON instance or another Map/List implementation. Both are supported.
- `file` *(optional)*: Optional path to a file in the workspace to write to. If provided, then `returnText` must be `false` or omitted. It is required that either `file` is provided, or `returnText` is `true`.
- `pretty` *(optional)*: Prettify the output with this number of spaces added to each level of indentation.
- `returnText` *(optional)*: Return the JSON as a string instead of writing it to a file. Defaults to `false`. If `true`, then `file` must not be provided. It is required that either `file` is provided, or `returnText` is `true`.

Example:
Writing to a file:

```
        def amap = ['something': 'my datas',
                    'size': 3,
                    'isEmpty': false]

        writeJSON file: 'data.json', json: amap
        def read = readJSON file: 'data.json'

        assert read.something == 'my datas'
        assert read.size == 3
        assert read.isEmpty == false
```

Writing to a string:

```
        def amap = ['something': 'my datas',
                    'size': 3,
                'isEmpty': false]

        String json = writeJSON returnText: true, json: amap
        def read = readJSON text: json

        assert read.something == 'my datas'
        assert read.size == 3
        assert read.isEmpty == false
```

- `json : Object`
- `file : String` (optional)
- `pretty : int` (optional)
- `returnText : boolean` (optional)

`writeMavenPom`: Write a maven project file.

Writes a Maven project file. That for example was previously read by `readMavenPom`.

Fields:

- `model`: The Model object to write.
- `file`: Optional path to a file in the workspace to write to. *If left empty the step will write to `pom.xml` in the current working directory.*

Example:

```
        def pom = readMavenPom file: 'pom.xml'
        //Do some manipulation
        writeMavenPom model: pom
```

Avoid using this step and `readMavenPom`. It is better to use the `sh` step to run `mvn` goals: For example:

```
sh 'mvn versions:set-property -Dproperty=some-key -DnewVersion=some-value -DgenerateBackupPoms=false'
```

- `model`
  - **Type:** `class org.apache.maven.model.Model`
- `file : String` (optional)

`writeYaml`: Write a yaml from an object or objects.

Writes yaml to a file in the current working directory or a String from an Object or a String. It uses SnakeYAML as YAML processor. The call will fail if the file already exists.

Fields:

- `file` *(optional)*: Optional path to a file in the workspace to write the YAML datas to. If provided, then `returnText` must be `false` or omitted. It is required that either `file` is provided, or `returnText` is `true`.
- `data` *(optional)*: An Optional Object containing the data to be serialized. You must specify `data` **or** `datas`, but not both in the same invocation.
- `datas` *(optional)*: An Optional Collection containing the datas to be serialized as several YAML documents. You must specify `data` **or** `datas`, but not both in the same invocation.
- `charset` *(optional)*: Optionally specify the charset to use when writing the file. Defaults to `UTF-8` if nothing else is specified. What charsets that are available depends on your Jenkins master system. The java specification tells us though that at least the following should be available:
  - US-ASCII
  - ISO-8859-1
  - UTF-8
  - UTF-16BE
  - UTF-16LE
  - UTF-16
- `overwrite` *(optional)*: Allow existing files to be overwritten. Defaults to `false`.

- `returnText` *(optional)*: Return the YAML as a string instead of writing it to a file. Defaults to `false`. If `true`, then `file`, `charset`, and `overwrite` must not be provided. It is required that either `file` is provided, or `returnText` is `true`.

Examples:
Writing to a file:

```
def amap = ['something': 'my datas',
            'size': 3,
            'isEmpty': false]

writeYaml file: 'datas.yaml', data: amap
def read = readYaml file: 'datas.yaml'

assert read.something == 'my datas'
assert read.size == 3
assert read.isEmpty == false
```

Writing to a string:

```
def amap = ['something': 'my datas',
            'size': 3,
            'isEmpty': false]

String yml = writeYaml returnText: true, data: amap
def read = readYaml text: yml

assert read.something == 'my datas'
assert read.size == 3
assert read.isEmpty == false
```

- `charset : String` (optional)
- `data : Object` (optional)
- `datas` (optional)
  Nested Choice of Objects
  - + `$class: 'RegistrationConfigCollection'`
  - - `$class: 'RegistrationConfigCollection'`
    - `data` (optional)
      - **Type:** `T`
- `file : String` (optional)
- `overwrite : boolean` (optional)
- `returnText : boolean` (optional)

`zip`: Create Zip file

Create a zip file of content in the workspace.

- `zipFile : String`

  The name/path of the zip file to create.

- `archive : boolean` (optional)

  If the zip file should be archived as an artifact of the current build. The file will still be kept in the workspace after archiving.

- `defaultExcludes : boolean` (optional)
- `dir : String` (optional)

  The path of the base directory to create the zip from. Leave empty to create from the current working directory.

- `exclude : String` (optional)

  Ant style pattern of files to exclude from the zip.

- `file : String` (optional)

- `glob : String` (optional)

  Ant style pattern of files to include in the zip. Leave empty to include all files and directories.

- `overwrite : boolean` (optional)

  If the zip file should be overwritten in case of already existing a file with the same name.

Was this page helpful?

Please submit your feedback about this page through this quick form.

Alternatively, if you don't wish to complete the quick form, you can simply indicate if you found this page helpful?

○ Yes ○ No

Type the answer to 5 plus 5 before clicking "Submit" below.

Submit

See existing feedback here.

Report page issue

Resources

Downloads
Blog
Documentation
Plugins
Security
Contributing

Project

Structure and governance
Issue tracker
Roadmap
GitHub
Jenkins on Jenkins

Community

Forum

Events
Mailing lists
Chats
Special Interest Groups
𝕏 (formerly Twitter)
Reddit

Other

Code of Conduct
Press information
Merchandise
Artwork
Awards