Number 09 in 2039

Gary B. Genett Author 1 Author 2 Author 3

2039-01-01

2020 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2021 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2022 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2023 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2024 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2025 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2026 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2027 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2028 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2029 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2030 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2031 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2032 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2033 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2034 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2035 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2036 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2037 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2038 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***

2039 Lorem Ipsum

Directory Tree

The ideal workflow is to put [Composer] in a top-level .Composer for each directory tree you want to manage, creating a structure similar to this:

```
.../.Composer
.../
.../tld/
.../tld/sub/
```

To save on disk space, using a central [Composer] install for multiple directory trees, the [init] target can be used to create a linked .Composer directory:

```
make -f .../Makefile init
```

The directory tree can then be converted to a [Composer] documentation archive ([Quick Start] example):

```
make -f .Composer/Makefile install-all
make all-all
```

Customization

If specific settings need to be used, either globally or per-directory, .composer.mk and .composer.yml files can be created (see [Configuration Settings], [Quick Start] example):

```
make template >.composer.mk && $EDITOR .composer.mk
make template-yml >.composer.yml && $EDITOR .composer.yml
```

Custom targets can also be defined, using standard [GNU Make] syntax (see [Custom Targets]).

Important Notes

[GNU Make] does not support file and directory names with spaces in them, and neither does [Composer]. Documentation archives which have such files or directories will produce unexpected results.

It is fully supported for input files to be symbolic links to files that reside outside the documentation archive:

```
cd .../tld
ln -rs .../README.md ./
make README.html
```

Similarly to source code, [GNU Make] is meant to only run one instance within the directory at a time, and [Composer] shares this requirement. It should be run as a single user, to avoid duplication and conflicts. Concurrent runs will produce unexpected results. It is highly recommended to set [MAKEJOBS] to a value greater than the default, to speed up processing.

It is best practice to [install-force] after every [Composer] upgrade, in case there are any changes to the Makefile template (see [Templates]). Everything in [Composer] sources from the main Makefile, so that is the only file which requires review to see what changes have been made between versions.

Next Steps

The archive is ready, and each directory is both a part of the collective and its own individual instance. Targets can be run per-file, per-directory, or recursively through an entire directory tree. The most commonly used targets are in [Primary Targets].

Welcome to [Composer]. ***Happy Making!***