NAME

yadm − Yet Another Dotfiles Manager

SYNOPSIS

yadm *command* [*options*]

yadm *git-command-or-alias* [*options*]

yadm init [**-f**] [**-w** *dir*]

yadm clone *url* [**-f**] [**-w** *dir*] [**-b** *branch*] [**--bootstrap**] [**--no-bootstrap**]

yadm config *name* [*value*]

yadm config [**-e**]

yadm list [**-a**]

yadm bootstrap

yadm encrypt

yadm decrypt [**-l**]

yadm alt

yadm perms

yadm enter [command]

yadm git-crypt [options]

yadm transcrypt [options]

yadm upgrade [**-f**]

yadm introspect *category*

DESCRIPTION

yadm is a tool for managing a collection of files across multiple computers, using a shared Git repository. In addition, yadm provides a feature to select alternate versions of files for particular systems. Lastly, yadm supplies the ability to manage a subset of secure files, which are encrypted before they are included in the repository.

COMMANDS

git-command or *git-alias*

Any command not internally handled by yadm is passed through to **git**(1). Git commands or aliases are invoked with the yadm managed repository. The working directory for Git commands will be the configured *work-tree* (usually *$HOME*).

Dotfiles are managed by using standard **git** commands; *add*, *commit*, *push*, *pull*, etc.

The *config* command is not passed directly through. Instead use the *gitconfig* command (see

below).

alt Create symbolic links and process templates for any managed files matching the naming rules described in the ALTERNATES and TEMPLATES sections. It is usually unnecessary to run this command, as yadm automatically processes alternates by default. This automatic behavior can be disabled by setting the configuration *yadm.auto-alt* to "false".

bootstrap
Execute *$HOME/.config/yadm/bootstrap* if it exists.

clone *url*
Clone a remote repository for tracking dotfiles. After the contents of the remote repository have been fetched, a "check out" of the remote HEAD branch is attempted. If there are conflicting files already present in the *work-tree*, the local version will be left unmodified and you'll have to review and resolve the difference.

The repository is stored in *$HOME/.local/share/yadm/repo.git*. By default, *$HOME* will be used as the *work-tree*, but this can be overridden with the **-w** option. yadm can be forced to overwrite an existing repository by providing the **-f** option. If you want to use a branch other than the remote HEAD branch you can specify it using the **-b** option. By default yadm will ask the user if the bootstrap program should be run (if it exists). The options **--bootstrap** or **--no-bootstrap** will either force the bootstrap to be run, or prevent it from being run, without prompting the user.

config This command manages configurations for yadm. This command works exactly the way **git-config**(1) does. See the CONFIGURATION section for more details.

decrypt
Decrypt all files stored in *$HOME/.local/share/yadm/archive*. Files decrypted will be relative to the configured *work-tree* (usually *$HOME*). Using the**-l** option will list the files stored without extracting them.

encrypt
Encrypt all files matching the patterns found in *$HOME/.config/yadm/encrypt*. See the ENCRYPTION section for more details.

enter Run a sub-shell with all Git variables set. Exit the sub-shell the same way you leave your normal shell (usually with the "exit" command). This sub-shell can be used to easily interact with your yadm repository using "git" commands. This could be useful if you are using a tool which uses Git directly, such as tig, vim-fugitive, git-cola, etc.

Optionally, you can provide a command after "enter", and instead of invoking your shell, that command will be run with all of the Git variables exposed to the command's environment.

Emacs Tramp and Magit can manage files by using this configuration:

```
(add-to-list 'tramp-methods
    '("yadm"
      (tramp-login-program "yadm")
      (tramp-login-args (("enter")))
      (tramp-login-env (("SHELL") ("/bin/sh")))
      (tramp-remote-shell "/bin/sh")
      (tramp-remote-shell-args ("-c"))))
```

With this config, use (magit-status "/yadm::").

git-crypt *options*
If git-crypt is installed, this command allows you to pass options directly to git-crypt, with the environment configured to use the yadm repository.

git-crypt enables transparent encryption and decryption of files in a git repository. You can read https://github.com/AGWA/git-crypt for details.

gitconfig
Pass options to the **git config** command. Since yadm already uses the *config* command to manage its own configurations, this command is provided as a way to change configurations of the repository managed by yadm. One useful case might be to configure the repository so untracked files are shown in status commands. yadm initially configures its repository so that untracked files are not shown. If you wish use the default Git behavior (to show untracked files and directories), you can remove this configuration.

yadm gitconfig --unset status.showUntrackedFiles

help Print a summary of yadm commands.

init Initialize a new, empty repository for tracking dotfiles. The repository is stored in *$HOME/.local/share/yadm/repo.git*. By default, *$HOME* will be used as the *work-tree*, but this can be overridden with the **-w** option. yadm can be forced to overwrite an existing repository by providing the **-f** option.

list Print a list of files managed by yadm. The **-a** option will cause all managed files to be listed. Otherwise, the list will only include files from the current directory or below.

introspect *category*
Report internal yadm data. Supported categories are *commands*, *configs*, *repo*, and *switches*. The purpose of introspection is to support command line completion.

perms Update permissions as described in the PERMISSIONS section. It is usually unnecessary to run this command, as yadm automatically processes permissions by default. This automatic behavior can be disabled by setting the configuration *yadm.auto-perms* to "false".

transcrypt *options*
If transcrypt is installed, this command allows you to pass options directly to transcrypt, with the environment configured to use the yadm repository.

transcrypt enables transparent encryption and decryption of files in a git repository. You can read https://github.com/elasticdog/transcrypt for details.

upgrade
Version 3 of yadm uses a different directory for storing data. When you start to use version 3 for the first time, you may see warnings about moving your data to this new directory. The easiest way to accomplish this is by running "yadm upgrade". This command will start by moving your yadm repo to the new path. Next it will move any archive data. If the archive is tracked within your yadm repo, this command will "stage" the renaming of that file in the repo's index.

Upgrading will attempt to de-initialize and re-initialize your submodules. If your submodules cannot be de-initialized, the upgrade will fail. The most common reason submodules will fail to de-initialize is because they have local modifications. If you are willing to lose the local modifications to those submodules, you can use the **-f** option with the "upgrade" command to force the de-initialization.

After running "yadm upgrade", you should run "yadm status" to review changes which have been staged, and commit them to your repository.

You can read https://yadm.io/docs/upgrade_from_2 for more information.

version
Print the version of yadm.

OPTIONS

yadm supports a set of universal options that alter the paths it uses. The default paths are documented in the FILES section. Any path specified by these options must be fully qualified. If you always want to override one or more of these paths, it may be useful to create an alias for the yadm command. For example, the following alias could be used to override the repository directory.

```
alias yadm='yadm --yadm-repo /alternate/path/to/repo'
```

The following is the full list of universal options. Each option should be followed by a path.

-Y,--yadm-dir
Override the yadm directory. yadm stores its configurations relative to this directory.

--yadm-data
Override the yadm data directory. yadm stores its data relative to this directory.

--yadm-repo
Override the location of the yadm repository.

--yadm-config
Override the location of the yadm configuration file.

--yadm-encrypt
Override the location of the yadm encryption configuration.

--yadm-archive
Override the location of the yadm encrypted files archive.

--yadm-bootstrap
Override the location of the yadm bootstrap program.

CONFIGURATION

yadm uses a configuration file named *$HOME/.config/yadm/config*. This file uses the same format as**git-config**(1). Also, you can control the contents of the configuration file via the**yadm config** command (which works exactly like **git-config**). For example, to disable alternates you can run the command:

```
yadm config yadm.auto-alt false
```

The following is the full list of supported configurations:

yadm.alt-copy
If set to "true", alternate files will be copies instead of symbolic links. This might be desirable, because some systems may not properly support symlinks.

yadm.auto-alt
Disable the automatic linking described in the section ALTERNATES. If disabled, you may still run "yadm alt" manually to create the alternate links. This feature is enabled by default.

yadm.auto-exclude
Disable the automatic exclusion of patterns defined in *$HOME/.config/yadm/encrypt*. This feature is enabled by default.

yadm.auto-perms
Disable the automatic permission changes described in the section PERMISSIONS. If disabled, you may still run **yadm perms** manually to update permissions. This feature is enabled by default.

yadm.auto-private-dirs
Disable the automatic creating of private directories described in the section PERMISSIONS.

yadm.cipher
Configure which encryption system is used by the encrypt/decrypt commands. Valid options are "gpg" and "openssl". The default is "gpg". Detailed information can be found in the section ENCRYPTION.

yadm.git-program
Specify an alternate program to use instead of "git". By default, the first "git" found in $PATH is used.

yadm.gpg-perms
Disable the permission changes to *$HOME/.gnupg/**. This feature is enabled by default.

yadm.gpg-program
Specify an alternate program to use instead of "gpg". By default, the first "gpg" found in $PATH is used.

yadm.gpg-recipient
Asymmetrically encrypt files with a gpg public/private key pair. Provide a "key ID" to specify which public key to encrypt with. The key must exist in your public keyrings. Multiple recipients can be specified (separated by space). If left blank or not provided, symmetric encryption is used instead. If set to "ASK", gpg will interactively ask for recipients. See the ENCRYPTION section for more details. This feature is disabled by default.

yadm.openssl-ciphername
Specify which cipher should be used by openssl. "aes-256-cbc" is used by default.

yadm.openssl-old
Newer versions of openssl support the pbkdf2 key derivation function. This is used by default. If this configuration is set to "true", openssl operations will use options compatible with older versions of openssl. If you change this option, you will need to recreate your encrypted archive.

yadm.openssl-program
Specify an alternate program to use instead of "openssl". By default, the first "openssl" found in $PATH is used.

yadm.ssh-perms
Disable the permission changes to *$HOME/.ssh/**. This feature is enabled by default.

The following four "local" configurations are not stored in the *$HOME/.config/yadm/config*, they are stored in the local repository.

local.class
Specify a class for the purpose of symlinking alternate files. By default, no class will be matched.

local.hostname
Override the hostname for the purpose of symlinking alternate files.

local.os
Override the OS for the purpose of symlinking alternate files.

local.user
Override the user for the purpose of symlinking alternate files.

ALTERNATES

When managing a set of files across different systems, it can be useful to have an automated way of choosing an alternate version of a file for a different operating system, host, user, etc.

yadm will automatically create a symbolic link to the appropriate version of a file, when a valid suffix is appended to the filename. The suffix contains the conditions that must be met for that file to be used.

The suffix begins with "##", followed by any number of conditions separated by commas.

##<condition>[,<condition>,...]

Each condition is an attribute/value pair, separated by a period. Some conditions do not require a "value", and in that case, the period and value can be omitted. Most attributes can be abbreviated as a single letter.

<attribute>[.<value>]

These are the supported attributes, in the order of the weighted precedence:

template, **t**
Valid when the value matches a supported template processor. See the TEMPLATES section for more details.

user, **u** Valid if the value matches the current user. Current user is calculated by running**id -u -n**.

distro, **d**
Valid if the value matches the distro. Distro is calculated by running **lsb_release -si** or by inspecting the ID from **/etc/os-release**.

os, **o** Valid if the value matches the OS. OS is calculated by running **uname -s**.

class, **c** Valid if the value matches the **local.class** configuration. Class must be manually set using **yadm config local.class <class>**. See the CONFIGURATION section for more details about setting **local.class**.

hostname, **h**
Valid if the value matches the short hostname. Hostname is calculated by running **uname -n**, and trimming off any domain.

default Valid when no other alternate is valid.

extension, **e**
A special "condition" that doesn't affect the selection process. Its purpose is instead to allow the alternate file to end with a certain extension to e.g. make editors highlight the content properly.

NOTE: The OS for "Windows Subsystem for Linux" is reported as "WSL", even though uname identifies as "Linux".

You may use any number of conditions, in any order. An alternate will only be used if ALL conditions are valid. For all files managed by yadm's repository or listed in *$HOME/.config/yadm/encrypt*, if they match this naming convention, symbolic links will be created for the most appropriate version.

The "most appropriate" version is determined by calculating a score for each version of a file. A template is always scored higher than any symlink condition. The number of conditions is the next largest factor in scoring. Files with more conditions will always be favored. Any invalid condition will disqualify that file completely.

If you don't care to have all versions of alternates stored in the same directory as the generated symlink, you can place them in the *$HOME/.config/yadm/alt* directory. The generated symlink or processed template will be created using the same relative path.

Alternate linking may best be demonstrated by example. Assume the following files are managed by yadm's repository:

- $HOME/path/example.txt##default
- $HOME/path/example.txt##class.Work
- $HOME/path/example.txt##os.Darwin

- $HOME/path/example.txt##os.Darwin,hostname.host1
- $HOME/path/example.txt##os.Darwin,hostname.host2
- $HOME/path/example.txt##os.Linux
- $HOME/path/example.txt##os.Linux,hostname.host1
- $HOME/path/example.txt##os.Linux,hostname.host2

If running on a Macbook named "host2", yadm will create a symbolic link which looks like this:

$HOME/path/example.txt -> $HOME/path/example.txt##os.Darwin,hostname.host2

However, on another Mackbook named "host3", yadm will create a symbolic link which looks like this:

$HOME/path/example.txt -> $HOME/path/example.txt##os.Darwin

Since the hostname doesn't match any of the managed files, the more generic version is chosen.

If running on a Linux server named "host4", the link will be:

$HOME/path/example.txt -> $HOME/path/example.txt##os.Linux

If running on a Solaris server, the link will use the default version:

$HOME/path/example.txt -> $HOME/path/example.txt##default

If running on a system, with class set to "Work", the link will be:

$HOME/path/example.txt -> $HOME/path/example.txt##class.Work

If no "##default" version exists and no files have valid conditions, then no link will be created.

Links are also created for directories named this way, as long as they have at least one yadm managed file within them.

yadm will automatically create these links by default. This can be disabled using the *yadm.auto-alt* configuration. Even if disabled, links can be manually created by running **yadm alt**.

Class is a special value which is stored locally on each host (inside the local repository). To use alternate symlinks using class, you must set the value of class using the configuration **local.class**. This is set like any other yadm configuration with the **yadm config** command. The following sets the class to be "Work".

```
yadm config local.class Work
```

Similarly, the values of os, hostname, and user can be manually overridden using the configuration options **local.os**, **local.hostname**, and **local.user**.

TEMPLATES

If a template condition is defined in an alternate file's "##" suffix, and the necessary dependencies for the template are available, then the file will be processed to create or overwrite files.

Supported template processors:

default This is yadm's built-in template processor. This processor is very basic, with a Jinja-like syntax. The advantage of this processor is that it only depends upon **awk**, which is available on most *nix systems. To use this processor, specify the value of "default" or just leave the value off (e.g. "##template").

ESH ESH is a template processor written in POSIX compliant shell. It allows executing shell commands within templates. This can be used to reference your own configurations within templates, for example:

```
<% yadm config mysection.myconfig %>
```

To use the ESH template processor, specify the value of "esh"

j2cli To use the j2cli Jinja template processor, specify the value of "j2" or "j2cli".

envtpl To use the envtpl Jinja template processor, specify the value of "j2" or "envtpl".

NOTE: Specifying "j2" as the processor will attempt to use j2cli or envtpl, whichever is available.

If the template processor specified is available, templates will be processed to create or overwrite files.

During processing, the following variables are available in the template:

Default	Jinja or ESH	Description
yadm.class	YADM_CLASS	Locally defined yadm class
yadm.distro	YADM_DISTRO	lsb_release -si
yadm.hostname	YADM_HOSTNAME	uname -n (without domain)
yadm.os	YADM_OS	uname -s
yadm.user	YADM_USER	id -u -n
yadm.source	YADM_SOURCE	Template filename

NOTE: The OS for "Windows Subsystem for Linux" is reported as "WSL", even though uname identifies as "Linux".

NOTE: If lsb_release is not available, DISTRO will be the ID specified in /etc/os-release.

Examples:

whatever##template with the following content

```
{% if yadm.user == "harvey" %}
config={{yadm.class}}-{{yadm.os}}
{% else %}
config=dev-whatever
{% include "whatever.extra" %}
{% endif %}
```

would output a file named *whatever* with the following content if the user is "harvey":

```
config=work-Linux
```

and the following otherwise (if *whatever.extra* contains admin=false):

```
config=dev-whatever
admin=false
```

An equivalent Jinja template named *whatever##template.j2* would look like:

```
{% if YADM_USER == 'harvey' -%}
config={{YADM_CLASS}}-{{YADM_OS}}
```

```
{% else -%}
config=dev-whatever
{% include 'whatever.extra' %}
{% endif -%}
```

An equivalent ESH templated named *whatever##template.esh* would look like:

```
<% if [ "$YADM_USER" = "harvey" ]; then -%>
config=<%= $YADM_CLASS %>-<%= $YADM_OS %>
<% else -%>
config=dev-whatever
<%+ whatever.extra %>
<% fi -%>
```

ENCRYPTION

It can be useful to manage confidential files, like SSH or GPG keys, across multiple systems. However, doing so would put plain text data into a Git repository, which often resides on a public system. yadm can make it easy to encrypt and decrypt a set of files so the encrypted version can be maintained in the Git repository. This feature will only work if a supported tool is available. Both**gpg**(1) and **openssl**(1) are supported. gpg is used by default, but openssl can be configured with the *yadm.cypher* configuration.

To use this feature, a list of patterns must be created and saved as *$HOME/.config/yadm/encrypt*. This list of patterns should be relative to the configured *work-tree* (usually *$HOME*). For example:

```
.ssh/*.key
.gnupg/*.gpg
```

Standard filename expansions (*, ?, []) are supported. If you have Bash version 4, you may use "**" to match all subdirectories. Other shell expansions like brace and tilde are not supported. Spaces in paths are supported, and should not be quoted. If a directory is specified, its contents will be included, but not recursively. Paths beginning with a "!" will be excluded.

The **yadm encrypt** command will find all files matching the patterns, and prompt for a password. Once a password has confirmed, the matching files will be encrypted and saved as *$HOME/.local/share/yadm/archive*. The "encrypt" and "archive" files should be added to the yadm repository so they are available across multiple systems.

To decrypt these files later, or on another system run **yadm decrypt** and provide the correct password. After files are decrypted, permissions are automatically updated as described in the PERMISSIONS section.

Symmetric encryption is used by default, but asymmetric encryption may be enabled using the *yadm.gpg-recipient* configuration.

NOTE: It is recommended that you use a private repository when keeping confidential files, even though they are encrypted.

Patterns found in *$HOME/.config/yadm/encrypt* are automatically added to the repository's *info/exclude* file every time **yadm encrypt** is run. This is to prevent accidentally committing sensitive data to the repository. This can be disabled using the *yadm.auto-exclude* configuration.

Using transcrypt or git-crypt

A completely separate option for encrypting data is to install and use transcrypt or git-crypt. Once installed, you can use these tools by running **yadm transcrypt** or **yadm git-crypt**. These tools enables

transparent encryption and decryption of files in a git repository. See the following web sites for more information:

- https://github.com/elasticdog/transcrypt

- https://github.com/AGWA/git-crypt

PERMISSIONS

When files are checked out of a Git repository, their initial permissions are dependent upon the user's umask. Because of this, yadm will automatically update the permissions of some file paths. The "group" and "others" permissions will be removed from the following files:

- *$HOME/.local/share/yadm/archive*

- All files matching patterns in *$HOME/.config/yadm/encrypt*

- The SSH directory and files, *.ssh/**

- The GPG directory and files, *.gnupg/**

yadm will automatically update permissions by default. This can be disabled using the *yadm.auto-perms* configuration. Even if disabled, permissions can be manually updated by running **yadm perms**. The*.ssh* directory processing can be disabled using the *yadm.ssh-perms* configuration. The *.gnupg* directory processing can be disabled using the *yadm.gpg-perms* configuration.

When cloning a repo which includes data in a *.ssh* or *.gnupg* directory, if those directories do not exist at the time of cloning, yadm will create the directories with mask 0700 prior to merging the fetched data into the work-tree.

When running a Git command and *.ssh* or *.gnupg* directories do not exist, yadm will create those directories with mask 0700 prior to running the Git command. This can be disabled using the *yadm.auto-private-dirs* configuration.

HOOKS

For every command yadm supports, a program can be provided to run before or after that command. These are referred to as "hooks". yadm looks for hooks in the directory *$HOME/.config/yadm/hooks*. Each hook is named using a prefix of *pre_* or *post_*, followed by the command which should trigger the hook. For example, to create a hook which is run after every *yadm pull* command, create a hook named *post_pull*. Hooks must have the executable file permission set.

If a *pre_* hook is defined, and the hook terminates with a non-zero exit status, yadm will refuse to run the yadm command. For example, if a *pre_commit* hook is defined, but that command ends with a non-zero exit status, the *yadm commit* will never be run. This allows one to "short-circuit" any operation using a *pre_* hook.

Hooks have the following environment variables available to them at runtime:

YADM_HOOK_COMMAND
: The command which triggered the hook

YADM_HOOK_EXIT
: The exit status of the yadm command

YADM_HOOK_FULL_COMMAND
: The yadm command with all command line arguments (parameters are space delimited, and any space, tab or backslash will be escaped with a backslash)

YADM_HOOK_REPO
The path to the yadm repository

YADM_HOOK_WORK
The path to the work-tree

FILES

All of yadm's configurations are relative to the "yadm directory". yadm uses the "XDG Base Directory Specification" to determine this directory. If the environment variable **$XDG_CONFIG_HOME** is defined as a fully qualified path, this directory will be *$XDG_CONFIG_HOME/yadm*. Otherwise it will be *$HOME/.config/yadm*.

Similarly, yadm's data files are relative to the "yadm data directory". yadm uses the "XDG Base Directory Specification" to determine this directory. If the environment variable **$XDG_DATA_HOME** is defined as a fully qualified path, this directory will be *$XDG_DATA_HOME/yadm*. Otherwise it will be*$HOME/.local/share/yadm*.

The following are the default paths yadm uses for its own data. Most of these paths can be altered using universal options. See the OPTIONS section for details.

$HOME/.config/yadm
The yadm directory. By default, all configs yadm stores is relative to this directory.

$HOME/.local/share/yadm
The yadm data directory. By default, all data yadm stores is relative to this directory.

$YADM_DIR/config
Configuration file for yadm.

$YADM_DIR/alt
This is a directory to keep "alternate files" without having them side-by-side with the resulting symlink or processed template. Alternate files placed in this directory will be created relative to $HOME instead.

$YADM_DATA/repo.git
Git repository used by yadm.

$YADM_DIR/encrypt
List of globs used for encrypt/decrypt

$YADM_DATA/archive
All files encrypted with **yadm encrypt** are stored in this file.

EXAMPLES

yadm init
Create an empty repo for managing files

yadm add .bash_profile ; yadm commit
Add *.bash_profile* to the Git index and create a new commit

yadm remote add origin <url>
Add a remote origin to an existing repository

yadm push -u origin master
Initial push of master to origin

echo .ssh/*.key >> $HOME/.config/yadm/encrypt
Add a new pattern to the list of encrypted files

yadm encrypt ; yadm add ~/.local/share/yadm/archive ; yadm commit
Commit a new set of encrypted files

REPORTING BUGS

Report issues or create pull requests at GitHub:

https://github.com/TheLocehiliosan/yadm/issues

AUTHOR

Tim Byrne <sultan@locehilios.com>

SEE ALSO

git(1), **gpg**(1) **openssl**(1) **transcrypt**(1) **git-crypt**(1)

https://yadm.io/