Putting git hooks into repository

Is it considered to be a bad practice - to put .git/hooks into the projects repository (using symlinks, for example). If yes, what is the best way to deliver same hooks to different git users?

git hook githooks

No, putting them into the repository is fine, I'd even suggest doing so (if they are useful for others as well). The user has to explicitly enable them (as you said, for example by symlinking), which is on one hand a bit of a pain, but protects users on the other hand from running arbitrary code without their consent.

9 what if it is a company policy thing, then the code is not "arbitrary" this is required code, so this would be considered a limitation in GIT, for not having another (pre-defined) directory, which is tracked, which also gets executed along with the regular hooks – Tobias Hagenbeek Dec 24 '14 at 15:56

9 Automatically delivering hooks is a security issue, I'm glad that Git doesn't do it directly - to enforce team/company policy, use hooks on the server side or let users manually decide to enable them as @scy describes :) – Mark K Cowan Apr 1 '15 at 7:46

3 "protects users [...] from running arbitrary code without their consent". If a developer would do like you suggest (symlinking) then the hook could be changed by someone else, and run "arbitrary code without their consent" – MiniGod Sep 25 '15 at 15:59

20 MiniGod: Of course. If you're sufficiently paranoid, you could copy the hooks instead of symlinking them, then audit them, and only then enable them. However, most (citation needed) Git repositories will contain source code which is to be run on the user's machine, so you're likely to run constantly changing, unaudited code anyway. But yes, you've got a point. ;) – scy Sep 30 '15 at 16:30

I generally agree with Scytale, with a couple additional suggestions, enough that it's worth a separate answer.

First, you should write a script which creates the appropriate symlinks, especially if these hooks are about enforcing policy or creating useful notifications. People will be much more likely to use the hooks if they can just type `bin/create-hook-symlinks` than if they have to do it themselves.

Second, directly symlinking hooks prevents users from adding in their own personal hooks. For example, I rather like the sample pre-commit hook which makes sure I don't have any whitespace errors. A great way around this is to drop in a hook wrapper script in your repo, and symlink *all* of the hooks to it. The wrapper can then examine `$0` (assuming it's a bash script; an equivalent like `argv[0]` otherwise) to figure out which hook it was invoked as, then invoke the appropriate hook within your repo, as well as the appropriate user's hook, which will have to be renamed, passing all the arguments to each. Quick example from memory:

```
#!/bin/bash
if [ -x $0.local ]; then
    $0.local "$@" || exit $?
fi
if [ -x tracked_hooks/$(basename $0) ]; then
    tracked_hooks/$(basename $0) "$@" || exit $?
fi
```

The installation script would move all pre-existing hooks to the side (append `.local` to their names), and symlink all known hook names to the above script:

```
#!/bin/bash
HOOK_NAMES="applypatch-msg pre-applypatch post-applypatch pre-commit prepare-commit-msg
commit-msg post-commit pre-rebase post-checkout post-merge pre-receive update post-
receive post-update pre-auto-gc"
# assuming the script is in a bin directory, one level into the repo
HOOK_DIR=$(git rev-parse --show-toplevel)/.git/hooks

for hook in $HOOK_NAMES; do
    # If the hook already exists, is executable, and is not a symlink
    if [ ! -h $HOOK_DIR/$hook -a -x $HOOK_DIR/$hook ]; then
        mv $HOOK_DIR/$hook $HOOK_DIR/$hook.local
    fi
    # create the symlink, overwriting the file if it exists
    # probably the only way this would happen is if you're using an old version of git
    # -- back when the sample hooks were not executable, instead of being named
____.sample
    ln -s -f ../../bin/hooks-wrapper $HOOK_DIR/$hook
done
```

6 I added `chmod +x .git/hooks/*` to your `bin/create-hook-symlinks` to work it. – guneysus Jan 5 '14 at 22:51

6 @guneysus You shouldn't need that, because the hooks should already be executable (they should be checked in that way) and the links don't need any special permissions, just the files they link to. – Cascabel Jan 6 '14 at 5:03

13 A better way to get the hook dir is `HOOK_DIR=$(git rev-parse --show-toplevel)/.git/hooks` . – Jasny - Arnold Daniels Jan 10 '14 at 20:14

2 I've put together a simple system based on this to manage the hooks in my project: ell.io/tt$Paws.js/blob/Master/Scripts/install-git-hooks.sh – ELLIOTTCABLE May 11 '14 at 23:24

6 I took just the essentials and put it in a repo github.com/sjungwirth/githooks – Scott Jungwirth Jul 15 '15 at 23:19

From http://git-scm.com/docs/git-init#_template_directory, you could use one of these mechanisms to update the .git/hooks dir of each newly created git repo:

> The template directory contains files and directories that will be copied to the $GIT_DIR after it is created.
>
> The template directory will be one of the following (in order):
>
> - the argument given with the --template option;
> - the contents of the $GIT_TEMPLATE_DIR environment variable;
> - the init.templateDir configuration variable; or
> - the default template directory: /usr/share/git-core/templates.

The https://www.npmjs.com/package/pre-commit npm package handles this elegantly allowing you to specify pre-commit hooks in your package.json.

Store in the project and install in the build

As Scy states in his answer, If your hooks are specific for your particular projects I would include them in the project itself, managed by git. I would take this even further and say that, given that it is good practice to have your project build using a single script or command, your hooks should be installed during the build.

Java & Maven

Full disclaimer; I wrote the Maven plugin described below.

If you are handling build management with Maven for your Java projects, the following Maven plugin handles installing hooks from a location in your project.

https://github.com/rudikershaw/git-build-hook

```
<build>
  <plugins>
    <plugin>
      <groupId>com.rudikershaw.gitbuildhook</groupId>
      <artifactId>git-build-hook-maven-plugin</artifactId>
      <version>2.0.1</version>
      <configuration>
        <!-- The locations of a variety of different hooks to install in the local project. -->
        <preCommit>path/to/hook.sh</preCommit>
      </configuration>
      <executions>
        <execution>
          <goals>
            <!-- Inititalise a Git repository if one does not already exist. -->
            <goal>initialize</goal>
            <!-- Install Git hooks. -->
            <goal>install</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  <!-- ... etc ... -->
  </plugins>
</build>
```

JavaScript & NPM

For NPM there is a dependency called Husky which allows you to install hooks including ones written in JavaScript.

```
// package.json
{
  "husky": {
    "hooks": {
      "pre-commit": "npm test",
      "pre-push": "npm test",
      "...": "..."
    }
  }
}
```

Here's a script, add-git-hook.sh, which you can ship as a regular file in the repository and can be executed to append the git hook to the script file. Adjust which hook to use (pre-commit, post-commit, pre-push, etc.) and the definition of the hook in the cat heredoc.

```
#!/usr/bin/bash
# Adds the git-hook described below. Appends to the hook file
# if it already exists or creates the file if it does not.

HOOK_DIR=$(git rev-parse --show-toplevel)/.git/hooks
HOOK_FILE="$HOOK_DIR"/post-commit

# Create script file if doesn't exist
if [ ! -e "$HOOK_FILE" ] ; then
        echo '#!/usr/bin/bash' >> "$HOOK_FILE"
        chmod 700 "$HOOK_FILE"
fi

# Append hook code into script
cat >> "$HOOK_FILE" <<EOF

########################################
# ... post-commit hook script here ... #
########################################

EOF
```

This script might make sense to have executable permissions or the user can run it directly. I used this to automatically git-pull on other machines after I committed.

EDIT-- I answered the easier question which wasn't what was asked and wasn't what the OP was looking for. I opined on the use-cases and arguments for shipping hook scripts in the repo versus managing them externally in the comments below. Hope that was more what you're looking for.

I appreciate your effort and do believe there's a valuable information here however - it does not answer the question stated. – shabunc 2 days ago

In my opinion, if the hooks are specific to a particular repository or are integral components of the workflows used then they belong in the repository as files. It's hard to put them anywhere else without creating more problems than it solves. You could store general hooks in a repository of it's own or on a shared drive which could keep the project repo squeaky clean but at the cost of being much less practical. I agree with the other users in saying that the hooks must be easy to add. Symbolic links may create unnecessary dependence on a particular system or file structure. – mathewguest 2 days ago

Additionally, symbolic links break the users ability to add their own hooks. The .git/hooks directory isn't tracked so the source should start in the repository and make it's way into the hooks script, not the other way around. I think the counter-argument would be that the git hooks are more related to the workflow or team rather than the project and thus don't belong in the repository. Depending on your specific use-case, are you more okay with potentially polluting the git repository with less-related hooks or would you rather forego a bunch of complexity in putting them somewhere else? – mathewguest 2 days ago