GIT Hooks

What are Git Hooks?

Git hooks are scripts that run automatically every time a particular event occurs in a Git repository. They let you customize Git's internal behavior and trigger customizable actions at key points in the development life cycle.

Types of Git Hooks :

1. Client-side hooks
2. Server side hooks

- Common use cases for Git hooks include :
  - Encouraging a commit policy
  - Altering the project environment depending on the state of the repository
  - Implementing continuous integration workflows

- You can write these programs in any programming language (Ex : Bash, Python,Ruby etc.) as long as we put the shebang line at the beginning of each program

commit-msg.py

```
#!/usr/bin/python

import sys
import re
```

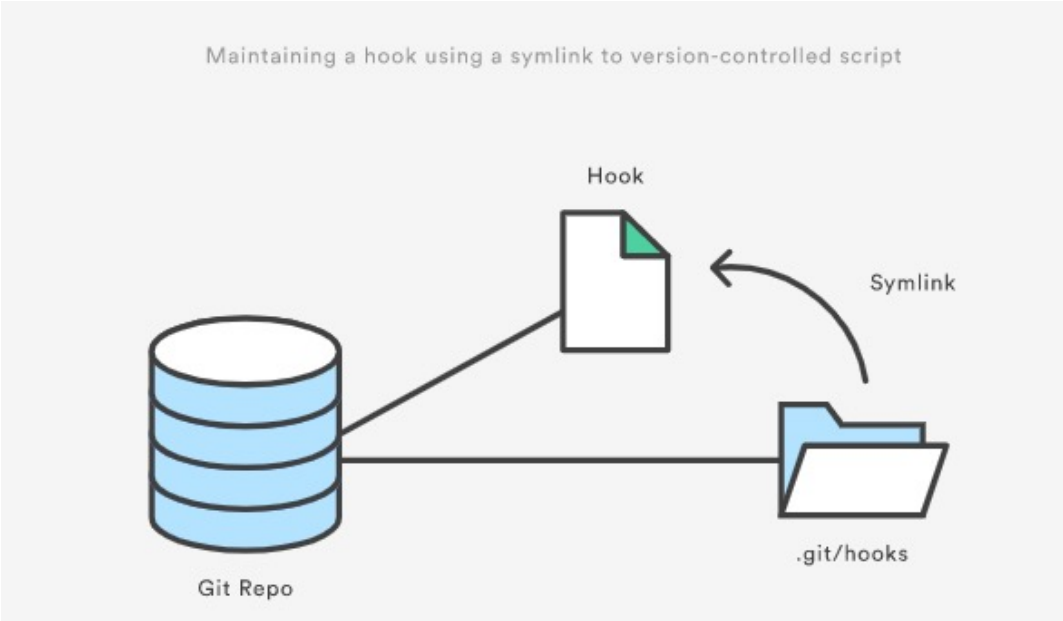
```
#!/bin/sh

echo "# Please include a useful commit message!"
```

Installing Git Hooks

```
bash-4.2$ pwd
/home/sunda29/code_repo_initiative/sample_hooks
bash-4.2$
bash-4.2$ ls -la
total 42
drwxr-xr-x 1 sunda29 users     8 Oct  7 22:59 .
drwxr-xr-x 1 sunda29 users     2 Oct  4 01:02 ..
drwxr-xr-x 1 sunda29 users    13 Oct  7 23:00 .git
drwxr-xr-x 1 sunda29 users     4 Oct  7 22:57 hooks
-rw-r--r-- 1 sunda29 users   627 Oct  4 01:09 install_hooks.bat
-rw-r--r-- 1 sunda29 users   914 Oct  5 01:04 install_hooks.sh
drwxr-xr-x 1 sunda29 users     4 Oct  7 22:59 .ipynb_checkpoints
-rw-r--r-- 1 sunda29 users  1193 Oct  7 22:37 README.md
-rw-r--r-- 1 sunda29 users 39268 Oct  7 22:35 sample.py
drwxr-xr-x 1 sunda29 users     3 Oct  7 22:59 src
bash-4.2$
bash-4.2$ cd .git/hooks/
bash-4.2$
bash-4.2$ ls -la
total 19
drwxr-xr-x 1 sunda29 users   13 Oct  7 22:39 .
drwxr-xr-x 1 sunda29 users   13 Oct  7 23:00 ..
-rwxr-xr-x 1 sunda29 users  452 Oct  4 01:02 applypatch-msg.sample
lrwxrwxrwx 1 sunda29 users   25 Oct  7 22:39 commit-msg -> ../../hooks/commit-msg.py
-rwxr-xr-x 1 sunda29 users  896 Oct  4 01:02 commit-msg.sample
-rwxr-xr-x 1 sunda29 users  189 Oct  4 01:02 post-update.sample
-rwxr-xr-x 1 sunda29 users  398 Oct  4 01:02 pre-applypatch.sample
lrwxrwxrwx 1 sunda29 users   25 Oct  7 22:39 pre-commit -> ../../hooks/pre-commit.py
-rwxr-xr-x 1 sunda29 users 1704 Oct  4 01:02 pre-commit.sample
lrwxrwxrwx 1 sunda29 users   33 Oct  5 00:21 prepare-commit-msg -> ../../hooks/prepare-commit-msg.py
-rwxr-xr-x 1 sunda29 users 1239 Oct  4 01:02 prepare-commit-msg.sample
-rw-r--r-- 1 sunda29 users 1348 Oct  4 01:02 pre-push.sample
lrwxrwxrwx 1 sunda29 users   25 Oct  7 22:39 pre-rebase -> ../../hooks/pre-rebase.py
-rwxr-xr-x 1 sunda29 users 4951 Oct  4 01:02 pre-rebase.sample
-rwxr-xr-x 1 sunda29 users 3611 Oct  4 01:02 update.sample
bash-4.2$
```

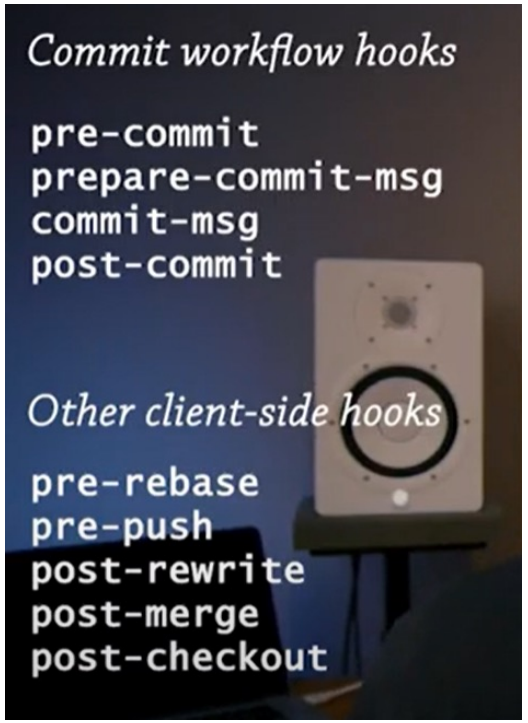
- Hooks reside in the .git/hooks directory of every Git repository
- Need to remove .sample at the end and provide EXECUTE permissions (*chmod +x filename*) for Git hooks to be installed and executable



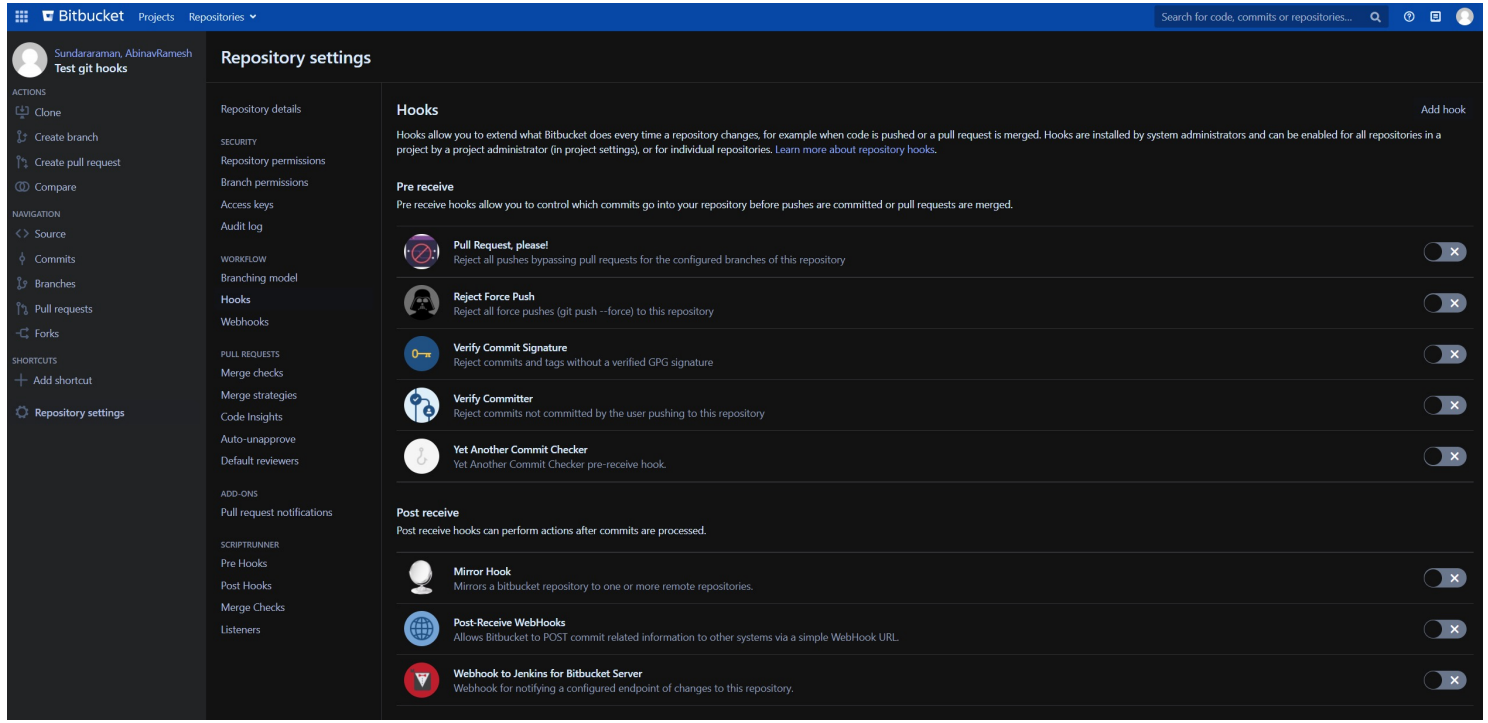
- Place Hook programs inside the git repo
- Create Symbolic links between programs outside git repo to those within .git/hooks directory (View the pink boxes on the first pic)

Types of Git Hooks

Client side hooks



Server side hooks



You can view Server side hooks as an admin of the project

Types of Git Hooks - Explanation

Hook	Command	When	Exit 1	Parameters
applypatch-msg	git am	before patch applied	stops patch application	message file name
pre-applypatch	git am	after applied, before commit	stops commit	n/a
post-applypatch	git am	after commit	n/a	n/a CLIENT SIDE
pre-commit	git commit	before message	stops commit	n/a
prepare-commit-msg	git commit	after default message generated, before invoking editor	stops commit	message file name, message source type, commit hash
commit-msg	git commit	inspect, edit, and format the message	stops commit	message file name
post-commit	git commit	after commit finished	n/a	n/a
pre-rebase	git rebase	before rebase	stops rebase	n/a
post-checkout	git checkout git clone	after worktree is changed	n/a	previous HEAD ref, new HEAD ref, is branch
post-merge	git merge git pull	after successful merge completed	n/a	is squash
pre-receive	git push	before the first ref is updated	stops update	n/a (but STDIN gets refs)
update	git push	before each ref is updated	stops present ref from being updated	ref name, old object name, new object name
post-receive	git push	after the last ref is updated	n/a	n/a (but STDIN gets refs) SERVER
post-update	git push	after the last ref is updated	n/a	ref names that were updated SIDE
pre-auto-gc	git gc --auto	before garbage collection begins	stops garbage collection	n/a

Use cases of Git Hooks – Commit Workflow hooks

Commit workflow hooks :

1. pre-commit
   - Linting/Static code analysis
   - Spell checking
   - Codestyle (Enforce PEP-8 standards)

2. Prepare-commit-msg
   - To alter the commit message/ give a completely new commit message
   - We can include ticket id, branch name, style checklist, rules for commits

3. Commit-msg
   - to check that your commit message is conformant to a required pattern.

4. Post-commit
   - We can use it for notifications to team members via slack, teams

Use cases of Git Hooks – Other Client side hooks

Other client side hooks :

1. pre-rebase
   - You can use this hook to disallow rebasing any commits that have already been pushed.
2. Pre-push
   - You can use it to validate a set of ref updates before a push occurs (a non-zero exit code will abort the push).
3. Post-rewrite : This hook has many of the same uses as the post-checkout and post-merge hooks.
4. Post-merge
   - You can use it to restore data in the working tree that Git can't track, such as permissions data.
   - This hook can likewise validate the presence of files external to Git control that you may want copied in when the working tree changes.
5. Post-checkout
   - You can use it to set up your working directory properly for your project environment. This may mean moving in large binary files that you don't want source controlled, auto-generating documentation, or something along those lines.

E-mail workflow hooks (Only if you use *git –am command*)

1. applypatch-msg
2. pre-applypatch
3. post-applypatch

Use cases of Git Hooks – Server side hooks

Server side hooks :

1. pre-receive
   - make sure none of the updated references are non-fast-forwards, or to do access control for all the refs and files they're modifying with the push.

2. update : The update script is very similar to the pre-receive script, except that it's run once for each branch the pusher is trying to
3. Post-receive
   - can be used to update other services or notify users
   - emailing a list, notifying a continuous integration server, or updating a ticket-tracking system – you can even parse the commit messages to see if any tickets need to be opened, modified, or closed.

Example

Sample GIT project implementing hooks

Thank you!