This PDF contains the contents of folders and files from the directory 'PDF-Generator' and its subdirectories.

config (PDF-Generator\.git\config):

```
[core]
 repositoryformatversion = 0
 filemode = false
 bare = false
 logallrefupdates = true
 symlinks = false
 ignorecase = true
[submodule]
 active = .
[remote "origin"]
 url = https://github.com/playfuldreamz/PDF-Generator.git
 fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
 remote = origin
 merge = refs/heads/main
[lfs]
 repositoryformatversion = 0
```

description (PDF-Generator\.git\description):

Unnamed repository; edit this file 'description' to name the repository.

FETCH_HEAD (PDF-Generator\.git\FETCH_HEAD):

c3be2c4dc504c7ea5d455a8d9f9a5c38e5f2e2a4 branch 'main' of https://github.com/playfuldreamz/PDF-Generator

HEAD (PDF-Generator\.git\HEAD):

ref: refs/heads/main

applypatch-msg.sample (PDF-Generator\.git\hooks\applypatch-msg.sample):

```
#!/bin/sh
#
# An example hook script to check the commit log message taken by
# applypatch from an e-mail message.
#
# The hook should exit with non-zero status after issuing an
# appropriate message if it wants to stop the commit.  The hook is
# allowed to edit the commit message file.
#
# To enable this hook, rename this file to "applypatch-msg".

. git-sh-setup
commitmsg="$(git rev-parse --git-path hooks/commit-msg)"
test -x "$commitmsg" && exec "$commitmsg" ${1+"$@"}
:
```

commit-msg.sample (PDF-Generator\.git\hooks\commit-msg.sample):

```
#!/bin/sh
#
# An example hook script to check the commit log message.
# Called by "git commit" with one argument, the name of the file
# that has the commit message.  The hook should exit with non-zero
```

```
# status after issuing an appropriate message if it wants to stop the

# commit.  The hook is allowed to edit the commit message file.

#

# To enable this hook, rename this file to "commit-msg".



# Uncomment the below to add a Signed-off-by line to the message.

# Doing this in a hook is a bad idea in general, but the prepare-commit-msg

# hook is more suited to it.

#

# SOB=$(git var GIT_AUTHOR_IDENT | sed -n 's/^\(.*>\).*$/Signed-off-by: \1/p')

# grep -qs "^$SOB" "$1" || echo "$SOB" >> "$1"



# This example catches duplicate Signed-off-by lines.



test "" = "$(grep '^Signed-off-by: ' "$1" |

  sort | uniq -c | sed -e '/^[  ]*1[  ]/d')" || {

 echo >&2 Duplicate Signed-off-by lines.

 exit 1

}
```

fsmonitor-watchman.sample (PDF-Generator\.git\hooks\fsmonitor-watchman.sample):

```
#!/usr/bin/perl



use strict;

use warnings;

use IPC::Open2;
```

```
# An example hook script to integrate Watchman
# (https://facebook.github.io/watchman/) with git to speed up detecting
# new and modified files.
#
# The hook is passed a version (currently 2) and last update token
# formatted as a string and outputs to stdout a new update token and
# all files that have been modified since the update token. Paths must
# be relative to the root of the working tree and separated by a single NUL.
#
# To enable this hook, rename this file to "query-watchman" and set
# 'git config core.fsmonitor .git/hooks/query-watchman'
#
my ($version, $last_update_token) = @ARGV;

# Uncomment for debugging
# print STDERR "$0 $version $last_update_token\n";

# Check the hook interface version
if ($version ne 2) {
 die "Unsupported query-fsmonitor hook version '$version'.\n" .
     "Falling back to scanning...\n";
}

my $git_work_tree = get_working_dir();
```

```
my $retry = 1;

my $json_pkg;
eval {
 require JSON::XS;
 $json_pkg = "JSON::XS";
 1;
} or do {
 require JSON::PP;
 $json_pkg = "JSON::PP";
};

launch_watchman();

sub launch_watchman {
 my $o = watchman_query();
 if (is_work_tree_watched($o)) {
  output_result($o->{clock}, @{$o->{files}});
 }
}

sub output_result {
 my ($clockid, @files) = @_;

 # Uncomment for debugging watchman output
 # open (my $fh, ">", ".git/watchman-output.out");
```

```
 # binmode $fh, ":utf8";
 # print $fh "$clockid\n@files\n";
 # close $fh;

 binmode STDOUT, ":utf8";
 print $clockid;
 print "\0";
 local $, = "\0";
 print @files;
}

sub watchman_clock {
 my $response = qx/watchman clock "$git_work_tree"/;
 die "Failed to get clock id on '$git_work_tree'.\n" .
  "Falling back to scanning...\n" if $? != 0;

 return $json_pkg->new->utf8->decode($response);
}

sub watchman_query {
 my $pid = open2(\*CHLD_OUT, \*CHLD_IN, 'watchman -j --no-pretty')
 or die "open2() failed: $!\n" .
 "Falling back to scanning...\n";

 # In the query expression below we're asking for names of files that
 # changed since $last_update_token but not from the .git folder.
```

```
#
# To accomplish this, we're using the "since" generator to use the
# recency index to select candidate nodes and "fields" to limit the
# output to file names only. Then we're using the "expression" term to
# further constrain the results.
my $last_update_line = "";
if (substr($last_update_token, 0, 1) eq "c") {
 $last_update_token = "\"$last_update_token\"";
 $last_update_line = qq[\n"since": $last_update_token,];
}
my $query = <<" END";
 ["query", "$git_work_tree", {$last_update_line
  "fields": ["name"],
  "expression": ["not", ["dirname", ".git"]]
 }]
END

# Uncomment for debugging the watchman query
# open (my $fh, ">", ".git/watchman-query.json");
# print $fh $query;
# close $fh;

print CHLD_IN $query;
close CHLD_IN;
my $response = do {local $/; <CHLD_OUT>};
```

```
 # Uncomment for debugging the watch response
 # open ($fh, ">", ".git/watchman-response.json");
 # print $fh $response;
 # close $fh;

 die "Watchman: command returned no output.\n" .
 "Falling back to scanning...\n" if $response eq "";
 die "Watchman: command returned invalid output: $response\n" .
 "Falling back to scanning...\n" unless $response =~ /^\{/;

 return $json_pkg->new->utf8->decode($response);
}

sub is_work_tree_watched {
 my ($output) = @_;
 my $error = $output->{error};
 if ($retry > 0 and $error and $error =~ m/unable to resolve root .* directory (.*) is not watched/) {
  $retry--;
  my $response = qx/watchman watch "$git_work_tree"/;
  die "Failed to make watchman watch '$git_work_tree'.\n" .
      "Falling back to scanning...\n" if $? != 0;
  $output = $json_pkg->new->utf8->decode($response);
  $error = $output->{error};
  die "Watchman: $error.\n" .
  "Falling back to scanning...\n" if $error;
```

```
  # Uncomment for debugging watchman output
  # open (my $fh, ">", ".git/watchman-output.out");
  # close $fh;

  # Watchman will always return all files on the first query so
  # return the fast "everything is dirty" flag to git and do the
  # Watchman query just to get it over with now so we won't pay
  # the cost in git to look up each individual file.
  my $o = watchman_clock();
  $error = $output->{error};

  die "Watchman: $error.\n" .
  "Falling back to scanning...\n" if $error;

  output_result($o->{clock}, ("/"));
  $last_update_token = $o->{clock};

  eval { launch_watchman() };
  return 0;
 }

 die "Watchman: $error.\n" .
 "Falling back to scanning...\n" if $error;

 return 1;
}
```

```
sub get_working_dir {
 my $working_dir;
 if ($^O =~ 'msys' || $^O =~ 'cygwin') {
  $working_dir = Win32::GetCwd();
  $working_dir =~ tr/\\/\//;
 } else {
  require Cwd;
  $working_dir = Cwd::cwd();
 }

 return $working_dir;
}
```

post-update.sample (PDF-Generator\.git\hooks\post-update.sample):

```
#!/bin/sh
#
# An example hook script to prepare a packed repository for use over
# dumb transports.
#
# To enable this hook, rename this file to "post-update".

exec git update-server-info
```

pre-applypatch.sample (PDF-Generator\.git\hooks\pre-applypatch.sample):

```
#!/bin/sh
```

```
#
# An example hook script to verify what is about to be committed
# by applypatch from an e-mail message.
#
# The hook should exit with non-zero status after issuing an
# appropriate message if it wants to stop the commit.
#
# To enable this hook, rename this file to "pre-applypatch".

. git-sh-setup
precommit="$(git rev-parse --git-path hooks/pre-commit)"
test -x "$precommit" && exec "$precommit" ${1+"$@"}
:
```

pre-commit.sample (PDF-Generator\.git\hooks\pre-commit.sample):

```
#!/bin/sh
#
# An example hook script to verify what is about to be committed.
# Called by "git commit" with no arguments.  The hook should
# exit with non-zero status after issuing an appropriate message if
# it wants to stop the commit.
#
# To enable this hook, rename this file to "pre-commit".

if git rev-parse --verify HEAD >/dev/null 2>&1
then
```

```
 against=HEAD
else
 # Initial commit: diff against an empty tree object
 against=$(git hash-object -t tree /dev/null)
fi

# If you want to allow non-ASCII filenames set this variable to true.
allownonascii=$(git config --type=bool hooks.allownonascii)

# Redirect output to stderr.
exec 1>&2

# Cross platform projects tend to avoid non-ASCII filenames; prevent
# them from being added to the repository. We exploit the fact that the
# printable range starts at the space character and ends with tilde.
if [ "$allownonascii" != "true" ] &&
 # Note that the use of brackets around a tr range is ok here, (it's
 # even required, for portability to Solaris 10's /usr/bin/tr), since
 # the square bracket bytes happen to fall in the designated range.
 test $(git diff --cached --name-only --diff-filter=A -z $against |
   LC_ALL=C tr -d '[ -~]\0' | wc -c) != 0
then
 cat <<\EOF
Error: Attempt to add a non-ASCII file name.

This can cause problems if you want to work with people on other platforms.
```

To be portable it is advisable to rename the file.

If you know what you are doing you can disable this check using:

```
  git config hooks.allownonascii true
EOF
 exit 1
fi

# If there are whitespace errors, print the offending file names and fail.
exec git diff-index --check --cached $against --
```

pre-merge-commit.sample (PDF-Generator\.git\hooks\pre-merge-commit.sample):

```
#!/bin/sh
#
# An example hook script to verify what is about to be committed.
# Called by "git merge" with no arguments.  The hook should
# exit with non-zero status after issuing an appropriate message to
# stderr if it wants to stop the merge commit.
#
# To enable this hook, rename this file to "pre-merge-commit".

. git-sh-setup
test -x "$GIT_DIR/hooks/pre-commit" &&
        exec "$GIT_DIR/hooks/pre-commit"
```

:

pre-push.sample (PDF-Generator\.git\hooks\pre-push.sample):

```
#!/bin/sh

# An example hook script to verify what is about to be pushed.  Called by "git
# push" after it has checked the remote status, but before anything has been
# pushed.  If this script exits with a non-zero status nothing will be pushed.
#
# This hook is called with the following parameters:
#
# $1 -- Name of the remote to which the push is being done
# $2 -- URL to which the push is being done
#
# If pushing without using a named remote those arguments will be equal.
#
# Information about the commits which are being pushed is supplied as lines to
# the standard input in the form:
#
#   <local ref> <local oid> <remote ref> <remote oid>
#
# This sample shows how to prevent push of commits where the log message starts
# with "WIP" (work in progress).

remote="$1"
url="$2"
```

```
zero=$(git hash-object --stdin </dev/null | tr '[0-9a-f]' '0')

while read local_ref local_oid remote_ref remote_oid
do
 if test "$local_oid" = "$zero"
 then
  # Handle delete
  :
 else
  if test "$remote_oid" = "$zero"
  then
   # New branch, examine all commits
   range="$local_oid"
  else
   # Update to existing branch, examine new commits
   range="$remote_oid..$local_oid"
  fi

  # Check for WIP commit
  commit=$(git rev-list -n 1 --grep '^WIP' "$range")
  if test -n "$commit"
  then
   echo >&2 "Found WIP commit in $local_ref, not pushing"
   exit 1
  fi
```

```
 fi

done


exit 0
```

pre-rebase.sample (PDF-Generator\.git\hooks\pre-rebase.sample):

```
#!/bin/sh

#

# Copyright (c) 2006, 2008 Junio C Hamano

#

# The "pre-rebase" hook is run just before "git rebase" starts doing

# its job, and can prevent the command from running by exiting with

# non-zero status.

#

# The hook is called with the following parameters:

#

# $1 -- the upstream the series was forked from.

# $2 -- the branch being rebased (or empty when rebasing the current branch).

#

# This sample shows how to prevent topic branches that are already

# merged to 'next' branch from getting rebased, because allowing it

# would result in rebasing already published history.


publish=next

basebranch="$1"

if test "$#" = 2
```

```
then
 topic="refs/heads/$2"
else
 topic=`git symbolic-ref HEAD` ||
 exit 0 ;# we do not interrupt rebasing detached HEAD
fi

case "$topic" in
refs/heads/??/*)
 ;;
*)
 exit 0 ;# we do not interrupt others.
 ;;
esac

# Now we are dealing with a topic branch being rebased
# on top of master.  Is it OK to rebase it?

# Does the topic really exist?
git show-ref -q "$topic" || {
 echo >&2 "No such branch $topic"
 exit 1
}

# Is topic fully merged to master?
not_in_master=`git rev-list --pretty=oneline ^master "$topic"`
```

```
if test -z "$not_in_master"
then
 echo >&2 "$topic is fully merged to master; better remove it."
 exit 1 ;# we could allow it, but there is no point.
fi

# Is topic ever merged to next?  If so you should not be rebasing it.
only_next_1=`git rev-list ^master "^$topic" ${publish} | sort`
only_next_2=`git rev-list ^master           ${publish} | sort`
if test "$only_next_1" = "$only_next_2"
then
 not_in_topic=`git rev-list "^$topic" master`
 if test -z "$not_in_topic"
 then
  echo >&2 "$topic is already up to date with master"
  exit 1 ;# we could allow it, but there is no point.
 else
  exit 0
 fi
else
 not_in_next=`git rev-list --pretty=oneline ^${publish} "$topic"`
 /usr/bin/perl -e '
  my $topic = $ARGV[0];
  my $msg = "* $topic has commits already merged to public branch:\n";
  my (%not_in_next) = map {
   /^([0-9a-f]+) /;
```

```
   ($1 => 1);
  } split(/\n/, $ARGV[1]);
  for my $elem (map {
    /^([0-9a-f]+) (.*)$/;
    [$1 => $2];
   } split(/\n/, $ARGV[2])) {
   if (!exists $not_in_next{$elem->[0]}) {
    if ($msg) {
     print STDERR $msg;
     undef $msg;
    }
    print STDERR " $elem->[1]\n";
   }
  }
 ' "$topic" "$not_in_next" "$not_in_master"
 exit 1
fi

<<\DOC_END

This sample hook safeguards topic branches that have been
published from being rewound.

The workflow assumed here is:

 * Once a topic branch forks from "master", "master" is never
```

merged into it again (either directly or indirectly).

* Once a topic branch is fully cooked and merged into "master",
  it is deleted. If you need to build on top of it to correct
  earlier mistakes, a new topic branch is created by forking at
  the tip of the "master". This is not strictly necessary, but
  it makes it easier to keep your history simple.

* Whenever you need to test or publish your changes to topic
  branches, merge them into "next" branch.

The script, being an example, hardcodes the publish branch name
to be "next", but it is trivial to make it configurable via
$GIT_DIR/config mechanism.

With this workflow, you would want to know:

(1) ... if a topic branch has ever been merged to "next". Young
    topic branches can have stupid mistakes you would rather
    clean up before publishing, and things that have not been
    merged into other branches can be easily rebased without
    affecting other people. But once it is published, you would
    not want to rewind it.

(2) ... if a topic branch has been fully merged to "master".
    Then you can delete it. More importantly, you should not

build on top of it -- other people may already want to

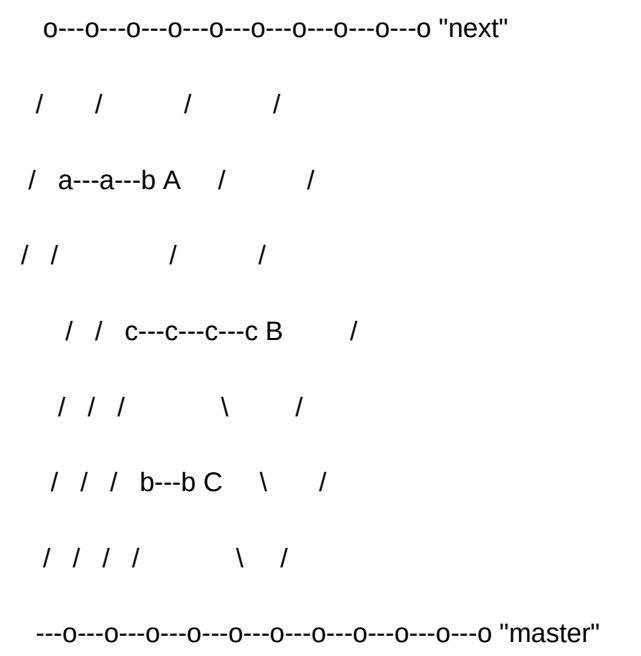
change things related to the topic as patches against your

"master", so if you need further changes, it is better to

fork the topic (perhaps with the same name) afresh from the

tip of "master".

Let's look at this example:



A, B and C are topic branches.

* A has one fix since it was merged up to "next".

* B has finished. It has been fully merged up to "master" and "next",

and is ready to be deleted.

 * C has not merged to "next" at all.

We would want to allow C to be rebased, refuse A, and encourage

B to be deleted.

To compute (1):

 git rev-list ^master ^topic next

 git rev-list ^master   next

 if these match, topic has not merged in next at all.

To compute (2):

 git rev-list master..topic

 if this is empty, it is fully merged to "master".

DOC_END

pre-receive.sample (PDF-Generator\.git\hooks\pre-receive.sample):

#!/bin/sh

#

An example hook script to make use of push options.

The example simply echoes all push options that start with 'echoback='

and rejects all pushes when the "reject" push option is used.

```
#
# To enable this hook, rename this file to "pre-receive".

if test -n "$GIT_PUSH_OPTION_COUNT"
then
 i=0
 while test "$i" -lt "$GIT_PUSH_OPTION_COUNT"
 do
  eval "value=\$GIT_PUSH_OPTION_$i"
  case "$value" in
  echoback=*)
   echo "echo from the pre-receive-hook: ${value#*=}" >&2
   ;;
  reject)
   exit 1
  esac
  i=$((i + 1))
 done
fi
```

prepare-commit-msg.sample (PDF-Generator\.git\hooks\prepare-commit-msg.sample):

```
#!/bin/sh
#
# An example hook script to prepare the commit log message.
# Called by "git commit" with the name of the file that has the
# commit message, followed by the description of the commit
```

```
# message's source.  The hook's purpose is to edit the commit
# message file.  If the hook fails with a non-zero status,
# the commit is aborted.
#
# To enable this hook, rename this file to "prepare-commit-msg".

# This hook includes three examples. The first one removes the
# "# Please enter the commit message..." help message.
#
# The second includes the output of "git diff --name-status -r"
# into the message, just before the "git status" output.  It is
# commented because it doesn't cope with --amend or with squashed
# commits.
#
# The third example adds a Signed-off-by line to the message, that can
# still be edited.  This is rarely a good idea.

COMMIT_MSG_FILE=$1
COMMIT_SOURCE=$2
SHA1=$3

/usr/bin/perl -i.bak -ne 'print unless(m/^. Please enter the commit message/..m/^#$/)' "$COMMIT_MSG_FILE"

# case "$COMMIT_SOURCE,$SHA1" in
#  ,|template,)
#    /usr/bin/perl -i.bak -pe '
```

```
#       print "\n" . `git diff --cached --name-status -r`
#   if /^#/ && $first++ == 0' "$COMMIT_MSG_FILE" ;;
#  *) ;;
# esac

# SOB=$(git var GIT_COMMITTER_IDENT | sed -n 's/^\(.*>\).*$/Signed-off-by: \1/p')
# git interpret-trailers --in-place --trailer "$SOB" "$COMMIT_MSG_FILE"
# if test -z "$COMMIT_SOURCE"
# then
#   /usr/bin/perl -i.bak -pe 'print "\n" if !$first_line++' "$COMMIT_MSG_FILE"
# fi
```

push-to-checkout.sample (PDF-Generator\.git\hooks\push-to-checkout.sample):

```
#!/bin/sh

# An example hook script to update a checked-out tree on a git push.
#
# This hook is invoked by git-receive-pack(1) when it reacts to git
# push and updates reference(s) in its repository, and when the push
# tries to update the branch that is currently checked out and the
# receive.denyCurrentBranch configuration variable is set to
# updateInstead.
#
# By default, such a push is refused if the working tree and the index
# of the remote repository has any difference from the currently
# checked out commit; when both the working tree and the index match
```

```
# the current commit, they are updated to match the newly pushed tip
# of the branch. This hook is to be used to override the default
# behaviour; however the code below reimplements the default behaviour
# as a starting point for convenient modification.
#
# The hook receives the commit with which the tip of the current
# branch is going to be updated:
commit=$1

# It can exit with a non-zero status to refuse the push (when it does
# so, it must not modify the index or the working tree).
die () {
 echo >&2 "$*"
 exit 1
}

# Or it can make any necessary changes to the working tree and to the
# index to bring them to the desired state when the tip of the current
# branch is updated to the new commit, and exit with a zero status.
#
# For example, the hook can simply run git read-tree -u -m HEAD "$1"
# in order to emulate git fetch that is run in the reverse direction
# with git push, as the two-tree form of git read-tree -u -m is
# essentially the same as git switch or git checkout that switches
# branches while keeping the local changes in the working tree that do
# not interfere with the difference between the branches.
```

```
# The below is a more-or-less exact translation to shell of the C code
# for the default behaviour for git's push-to-checkout hook defined in
# the push_to_deploy() function in builtin/receive-pack.c.
#
# Note that the hook will be executed from the repository directory,
# not from the working tree, so if you want to perform operations on
# the working tree, you will have to adapt your code accordingly, e.g.
# by adding "cd .." or using relative paths.

if ! git update-index -q --ignore-submodules --refresh
then
 die "Up-to-date check failed"
fi

if ! git diff-files --quiet --ignore-submodules --
then
 die "Working directory has unstaged changes"
fi

# This is a rough translation of:
#
#   head_has_history() ? "HEAD" : EMPTY_TREE_SHA1_HEX
if git cat-file -e HEAD 2>/dev/null
then
 head=HEAD
```

```
else
 head=$(git hash-object -t tree --stdin </dev/null)
fi

if ! git diff-index --quiet --cached --ignore-submodules $head --
then
 die "Working directory has staged changes"
fi

if ! git read-tree -u -m "$commit"
then
 die "Could not update working tree to new HEAD"
fi
```

update.sample (PDF-Generator\.git\hooks\update.sample):

```
#!/bin/sh
#
# An example hook script to block unannotated tags from entering.
# Called by "git receive-pack" with arguments: refname sha1-old sha1-new
#
# To enable this hook, rename this file to "update".
#
# Config
# ------
# hooks.allowunannotated
#   This boolean sets whether unannotated tags will be allowed into the
```

```
#   repository.  By default they won't be.
# hooks.allowdeletetag
#   This boolean sets whether deleting tags will be allowed in the
#   repository.  By default they won't be.
# hooks.allowmodifytag
#   This boolean sets whether a tag may be modified after creation. By default
#   it won't be.
# hooks.allowdeletebranch
#   This boolean sets whether deleting branches will be allowed in the
#   repository.  By default they won't be.
# hooks.denycreatebranch
#   This boolean sets whether remotely creating branches will be denied
#   in the repository.  By default this is allowed.
#

# --- Command line
refname="$1"
oldrev="$2"
newrev="$3"

# --- Safety check
if [ -z "$GIT_DIR" ]; then
 echo "Don't run this script from the command line." >&2
 echo " (if you want, you could supply GIT_DIR then run" >&2
 echo "  $0 <ref> <oldrev> <newrev>)" >&2
 exit 1
```

```
fi

if [ -z "$refname" -o -z "$oldrev" -o -z "$newrev" ]; then
 echo "usage: $0 <ref> <oldrev> <newrev>" >&2
 exit 1
fi

# --- Config
allowunannotated=$(git config --type=bool hooks.allowunannotated)
allowdeletebranch=$(git config --type=bool hooks.allowdeletebranch)
denycreatebranch=$(git config --type=bool hooks.denycreatebranch)
allowdeletetag=$(git config --type=bool hooks.allowdeletetag)
allowmodifytag=$(git config --type=bool hooks.allowmodifytag)

# check for no description
projectdesc=$(sed -e '1q' "$GIT_DIR/description")
case "$projectdesc" in
"Unnamed repository"* | "")
 echo "*** Project description file hasn't been set" >&2
 exit 1
 ;;
esac

# --- Check types
# if $newrev is 0000...0000, it's a commit to delete a ref.
zero=$(git hash-object --stdin </dev/null | tr '[0-9a-f]' '0')
```

```
if [ "$newrev" = "$zero" ]; then
 newrev_type=delete
else
 newrev_type=$(git cat-file -t $newrev)
fi


case "$refname","$newrev_type" in
 refs/tags/*,commit)
  # un-annotated tag
  short_refname=${refname##refs/tags/}
  if [ "$allowunannotated" != "true" ]; then
   echo "*** The un-annotated tag, $short_refname, is not allowed in this repository" >&2
   echo "*** Use 'git tag [ -a | -s ]' for tags you want to propagate." >&2
   exit 1
  fi
  ;;
 refs/tags/*,delete)
  # delete tag
  if [ "$allowdeletetag" != "true" ]; then
   echo "*** Deleting a tag is not allowed in this repository" >&2
   exit 1
  fi
  ;;
 refs/tags/*,tag)
  # annotated tag
  if [ "$allowmodifytag" != "true" ] && git rev-parse $refname > /dev/null 2>&1
```

```
 then
  echo "*** Tag '$refname' already exists." >&2
  echo "*** Modifying a tag is not allowed in this repository." >&2
  exit 1
 fi
 ;;
refs/heads/*,commit)
 # branch
 if [ "$oldrev" = "$zero" -a "$denycreatebranch" = "true" ]; then
  echo "*** Creating a branch is not allowed in this repository" >&2
  exit 1
 fi
 ;;
refs/heads/*,delete)
 # delete branch
 if [ "$allowdeletebranch" != "true" ]; then
  echo "*** Deleting a branch is not allowed in this repository" >&2
  exit 1
 fi
 ;;
refs/remotes/*,commit)
 # tracking branch
 ;;
refs/remotes/*,delete)
 # delete tracking branch
 if [ "$allowdeletebranch" != "true" ]; then
```

```
    echo "*** Deleting a tracking branch is not allowed in this repository" >&2
    exit 1
  fi
  ;;
 *)
  # Anything else (is there anything else?)
  echo "*** Update hook: unknown type of update to ref $refname of type $newrev_type" >&2
  exit 1
  ;;
esac

# --- Finished
exit 0
```

exclude (PDF-Generator\.git\info\exclude):

```
# git ls-files --others --exclude-from=.git/info/exclude
# Lines that start with '#' are comments.
# For a project mostly in C, the following would be a good set of
# exclude patterns (uncomment them if you want to use them):
# *.[oa]
# *~
```

packed-refs (PDF-Generator\.git\packed-refs):

```
# pack-refs with: peeled fully-peeled sorted
c3be2c4dc504c7ea5d455a8d9f9a5c38e5f2e2a4 refs/remotes/origin/main
```

main (PDF-Generator\.git\refs\heads\main):

c3be2c4dc504c7ea5d455a8d9f9a5c38e5f2e2a4

HEAD (PDF-Generator\.git\refs\remotes\origin\HEAD):

ref: refs/remotes/origin/main

config.json (PDF-Generator\config.json):

```
{
    "font_family": "Arial",
    "font_size": 10,
    "line_spacing": 10
}
```

LICENSE (PDF-Generator\LICENSE):

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,

and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by

the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without

modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and

do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this

License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

README.md (PDF-Generator\README.md):

PDF Generator: Convert Directory Contents to PDFs

This Python application helps you effortlessly convert the contents of files within a directory and its subdirectories into a well-formatted PDF document. It also generates a text file outlining the directory structure for easy reference.

Features

Easy to use: Simply provide the directory path and let the application do its magic!

Customizable: Choose which file types to include and exclude specific folders.

Parallel processing: Generates the PDF and directory structure text file simultaneously for faster execution.

GUI or command-line interface: Choose the interface that suits your workflow.

Cross-platform: Works seamlessly on Windows, macOS, and Linux.

Open-source: Freely use, modify, and contribute to the project.

Getting Started

Prerequisites:

Python 3.6 or later

Tkinter library (usually included with Python)

Installation:

Clone or download the repository.

Open a terminal in the project directory.

Install the required dependencies:

pip install -r requirements.txt

Usage:

Basic usage

python main.py <directory_path>

Example: Generate PDF from files in "my_project" directory

python main.py my_project

Include hidden files

python main.py my_project -i

Specify file types to process (e.g., .txt and .py)

python main.py my_project -t .txt .py

Exclude specific folders (e.g., "logs" and "venv")

python main.py my_project -e logs venv

Combine options

python main.py my_project -i -t .txt .py -e logs venv

GUI:

Run python main.py.

Use the GUI to select the directory, specify file types, and choose options.

Click "Generate PDF".

Contributing

We welcome contributions to make this project even better! Here's how you can get involved:

Report bugs: If you encounter any issues, please open an issue on GitHub.

Suggest features: Have an idea for a cool new feature? Share it with us!

Submit pull requests: Fix bugs, implement new features, or improve the documentation.

Development Setup:

Fork the repository.

Create a virtual environment:

python -m venv venv

Activate the virtual environment:

Windows: venv\Scripts\activate

macOS/Linux: source venv/bin/activate

Install development dependencies:

pip install -r requirements-dev.txt

Run tests:

pytest