github

kevinluo

Contents

contents

1 參考鏈接

git-scm 官方 doc 網址

Git push 常見用法

git add -A 和 git add . 的區別

git 命令之 git remote 的用法

git 跟踪遠程分支，查看本地分支追踪和遠程分支的關系

git 跟踪遠程分支，查看本地分支追踪和遠程分支的關系簡書

Git 本地分支與遠程分支的追踪關系

2 工具

git 恢復保存 REPO 文件時間信息的工具--

stackoverflow.com/ What's the equivalent of use-commit-times for git? <https://stackoverflow.com/questions/1964470/w
the-equivalent-of-use-commit-times-for-git/13284229#13284229 >__

2.1 metastore 額外加上文件 meta 信息

Making git usable for backing up file attributes too

metastore is a tool to store the metadata of files/directories/links in a file tree to a separate file and to later compare and apply the stored metadata to said file tree.

2.2 git-tools: git-restore-mtime 恢復文件時間等功能 use-commit-times

MestreLion/git-tools

git-restore-mtime Restore original modification time of files based on the date of the most recent commit that modified them

Probably the most popular and useful tool, and the reason this repository was packaged into Debian.

Git, unlike other version control systems, does not preserve the original timestamp of committed files. Whenever repositories are cloned, or branches/files are checked out, file timestamps are reset to the current date. While this behavior has its justifications (notably when using make to compile software), sometimes it is desirable to restore the original modification date of a file (for example, when generating release tarballs). As git does not provide any way to do that, git-restore-mtime tries to workaround this limitation.

For more information and background, see stackoverflow.com/ whats-the-equivalent-of-use-commit-times

For TravisCI users, simply add a config to .travis.yml so it clones the full repository history:

git: depth: false

2.3 參考鏈接 1:What’s the equivalent of use-commit-times for git?

stackoverflow.com/ What’s the equivalent of use-commit-times for git? <https://stackoverflow.com/questions/1964470/w
the-equivalent-of-use-commit-times-for-git/13284229#13284229 >__

IMHO, not storing timestamps (and other metadata like permissions and ownership) is a big limitation of git.

Linus’ rationale of timestamps being harmful just because it ”confuses make” is lame:

make clean is enough to fix any problems.

Applies only to projects that use make, mostly C/C++. It is completely moot for scripts like Python, Perl, or documentation in general.

There is only harm if you apply the timestamps. There would be no harm in storing them in repo. Applying them could be a simple --with-timestamps option for git checkout and friends (clone, pull etc), at the user’s discretion.

Your arguments are valid. I’d hope somebody with some clout would make an enhancement request for git to have your suggested --with-timestamps option. – weberjn Nov 2 ’17 at 12:40

git 克隆更改文件修改時間 voidcn.com/article/p-pbfzuvro-bty.html

stackoverflow.com/questions/21735435/git-clone-changes-file-modification-time

2.4 kareltucek/git-mtime-extension

github.com/kareltucek/git-mtime-extension

2.5 shell solution optimized 1

Here is an optimized version of the above shell solutions, with minor fixes:

```
#!/bin/sh

if [ "$(uname)" = 'Darwin' ] ||
   [ "$(uname)" = 'FreeBSD' ]; then
   gittouch() {
     touch -ch -t "$(date -r "$(git log -1 --format=%ct "$1")"   '+%Y%m%d%H%M.%S')" "$1"
   }
else
   gittouch() {
      touch -ch -d "$(git log -1 --format=%ci "$1")" "$1"
   }
fi

git ls-files |
   while IFS= read -r file; do
      gittouch "$file"
   done
```

2.6 shell solution optimized 1

The following script incorporates the -n 1 and HEAD suggestions, works in most non-Linux environments (like Cygwin), and can be run on a checkout after the fact:

```
#!/bin/bash -e

OS=${OS:-`uname`}

get_file_rev() {
    git rev-list -n 1 HEAD "$1"
}

if [ "$OS" = 'FreeBSD' ]
then
    update_file_timestamp() {
     file_time=`date -r "$(git show --pretty=format:%at  --abbrev-commit "$(get_file_rev "$1")
        touch -h -t "$file_time" "$1"
    }
else
    update_file_timestamp() {
     file_time=`git show --pretty=format:%ai --abbrev-commit  "$(get_file_rev "$1")" | head -n
        touch -d "$file_time" "$1"
    }
fi

OLD_IFS=$IFS
IFS=$'\n'

for file in `git ls-files`
do
```

```
    update_file_timestamp "$file"
done

IFS=$OLD_IFS

git update-index --refresh
```

3 經驗點滴

3.0.1 命令

3.1 git clone

```
# 注意：如果直接顯式指明clone目標目錄，則一定要把repo名字寫上，不然不會自動加上； 如果省略,則
# <directory>The name of a new directory to clone into. The "humanish"  part of the source repos

  - git clone -b gh-pages https://$GH_TOKEN_FULL@github.com/kevinluolog/gp-memo.git
#  進入到clone 創建的repo目錄
  - cd gp-memo
```

3.2 git add

- git add -A 提交所有變化
- git add -u 提交被修改 (modified) 和被刪除 (deleted) 文件，不包括新文件 (new)
- git add . 提交新文件 (new) 和被修改 (modified) 文件，不包括被刪除 (deleted) 文件

```
git add .: 他會監控工作區的狀態樹，使用它會把工作時的所有變化提交到暫存區，包括文件內容修改(
git add -u ：他僅監控已經被add的文件（即tracked  file），他會將被修改的文件提交到暫存區。add
git add -A ：是上面兩個功能的合集（git add --all的縮寫）
```

3.3 git commit

```
# git commit -m ‘message’
# -m 參數表示可以直接輸入後面的“message”，如果不加 -m參數，那麼是不能直接輸入mess  age的，
# message即是我們用來簡要説明這次提交的語句。
# git commit -am ‘message’ -am等同于-a -m
# -a參數可以將所有已跟踪文件中的執行修改或刪除操作的文件都提交到本地倉庫，即使它們  沒有經過
#  注意：新加的文件（即沒有被git系統管理的文件）是不能被提交到本地倉庫的。
# --allow-empty
# Usually recording a commit that has the exact same tree as its sole  parent commit is a mistake
  - git commit --allow-empty -m "kl+travis+"
```

3.4 git push

```
# git push的一般形式爲 git push <遠程主機名> <本地分支名> <遠程分支名> ，例如  git push origin
# git push origin master
# 如果 遠程分支被省略，如上則表示將本地分支推送到與之存在追踪關系的遠程分支（通常兩者同名），
# git push origin : refs/for/master
# 如果省略本地分支名，則表示刪除指定的遠程分支，因爲這等同于推送一個空的本地分支到 遠程分支，
delete master
# git push origin
# 如果當前分支與遠程分支存在追踪關系，則本地分支和遠程分支都可以省略，將當前分支推送到origin
# git push
# 如果當前分支祇有一個遠程分支，那麼主機名都可以省略，形如 git push，可以使用git branch -r ，查
#  關于 refs/for:
# refs/for 的意義在于我們提交代碼到服務器之後是需要經過code review  之後才能進行merge的，而re
# 原文鏈接：https://blog.csdn.net/qq_37577660/article/details/78565899
  - git push https://$GH_TOKEN_FULL@github.com/kevinluolog/gp-memo.git
```

3.5 git remote

```
git remote -v
git init
git add xxx
git commit -m 'xxx'
git remote add origin ssh://software@172.16.0.30/~/yafeng/.git
git push origin master
git remote show origin
git clone https://$GH_TOKEN_FULL@github.com/kevinluolog/gp-memo.git
```

3.6 git ls-files -z --eol 獲取目錄下文件名

顯示所有文件

www.git-scm.com/docs/git-ls-files

```
$ git ls-files -z --eol
i/lf    w/lf    attr/              000misc/extract.mdi/lf    w/lf    attr/              000misc/memo

-z: 缺省把漢字等字符以\轉義碼輸出，加z表示以正常顯示字符輸出，但是加z時沒有分行
--eol:  will show i/<eolinfo><SPACES>w/<eolinfo><SPACES>attr/<eolattr><SPACE*><TAB><file>,  wi
```

3.7 git log -1 --date=iso --format="%ad" -- "$filename" 文件提交時間

www.git-scm.com/docs/git-log

查看文件最後一行：tail -1 文件名，後面必須是文件，或者 | tail -1 管道輸出的內容

顯示各純文件名：

```
git ls-files -z --eol | sed -e "s/i\/lf[ \t]*w\/lf[ \t]*attr\/[ \t]*/\n/  g"

顯示各文件首次COMMIT時間,注意linux下是lf,not crlr:
git ls-files -z --eol | sed -e "s/i\/lf[ \t]*w\/lf[ \t]*attr\/[ \t]*/\n/  g" | while read filenam
輸出格式：可以直接被 touch 參數 --date ""識別
2019-09-26 15:09:54 +0800

# touch 回創建時間
# 下面去掉bash -c 就能工作了。 xargs可以直接傳參數給touch使用的。
# 整個過程就是用git ls-files取到文件名，再用sed取出真正的文件名，再用git  log取到全部的commit
# 這個文件修改時間更新好後，還需要hexo的一個腳本，在渲染前把創建時間設置爲修改時  間。因爲hex
# klBlog\themes\next\scripts\filters\kl-touch-file-time.js

# ?? - git ls-files -z --eol | sed -e "s/i\/lf[ \t]*w\/lf[ \t]*attr\/[ \t]  */\n/g" | while read fi
 - git ls-files -z --eol | sed -e "s/i\/lf[ \t]*w\/lf[ \t]*attr\/[ \t]*/  \n/g" | while read filen
```

網上參考源碼片段

```
#? ? echo "touch --date=\"$(git log -1 --date=iso --format="%ad" --  "$filename")\" -m $filenam

#??git ls-files | xargs -I{} bash -c 'touch "{}" --date=@$(git log -n1 --pretty=format:%ct -- "{}

#??xargs -I{} bash -c 'touch $filename --date="{}"'
```

網上參考源碼，sh 批處理

```
# getcheckin - Retrieve the last committed checkin date and time for
#              each of the files in the git project.  After a "pull"
#              of the project, you can update the timestamp on the
#              pulled files to match that date/time.  There are many
#              that don't believe that this is not a good idea, but
#              I found it useful to get the right source file dates
#
#              NOTE: This script produces commands suitable for
#                    piping into BASH or other shell
# License: Creative Commons Attribution 3.0 United States
# (CC by 3.0 US)

##########
# walk back to the project parent or the relative pathnames don't make
# sense
##########
while [ ! -d ./.git ]
do
    cd ..
done
echo "cd $(pwd)"
##########
# Note that the date format is ISO so that touch will work
##########
git ls-tree -r --full-tree HEAD |\
    sed -e "s/.*\t//" | while read filename; do
   echo "touch --date=\"$(git log -1 --date=iso --format="%ad" --  "$filename")\" -m $filename"
done
```

3.7.1 跟踪遠程分支

從當前分支切換到‘dev’分支：
git checkout dev
建立并切換新分支：
git checkout -b 'dev'
查看當前詳細分支信息（可看到當前分支與對應的遠程追踪分支）：
git branch -vv
查看當前遠程倉庫信息
git remote -vv

如果用 git push 指令時，當前分支沒有跟踪遠程分支（沒有和遠程分支建立聯系），那麼就會 git 就會報錯

There is no tracking information for the current branch. Please specify which branch you want to merge with. 因爲當前分支沒有追踪遠程指定的分支的話，當前分支指定的版本快照不知道要作爲服務器哪一個分支的版本快照的子節點。簡單來説就是：不知道要推送給哪一個分支。那麼如何建立遠程分支：

克隆時自動將創建好的 master 分支追踪 origin/master 分支

```
git clone 服務器地址
git checkout -b develop origin/develop
```

在遠程分支的基礎上建立 develop 分支，并且讓 develop 分支追踪 origin/develop 遠程分支。

```
git branch --set-upstream branch-name origin/branch-name
```

將 branch-name 分支追踪遠程分支 origin/branch-name

```
git branch -u origin/serverfix
```

設置當前分支跟踪遠程分支 origin/serverfix

查看本地分支和遠程分支的跟踪關系

```
git branch -vv
```

比如輸入

```
$ git branch -vv
  develop   08775f9 [origin/develop] develop
  feature_1 b41865d [origin/feature_1] feature_1
* master    1399706 [my_github/master] init commit
```

develop 分支跟踪 origin/develop

feature_1 分支跟踪 origin/feature_1

master 跟踪了 my_github/master，且當前分支爲 master 分支

那麼假如我此時想要將 master 的改變推送到 origin 服務器的 master 分支上：

```
$ git checkout master//切換到master分支
...
$ git branch -u origin/master//將當前分支跟踪origin/master
```

Branch 'master' set up to track remote branch 'master' from 'origin'. 之後就可以執行 git add 和 git commit 了現在再查看一下本地和遠程的分支關系：

```
$ git branch -vv
  develop   08775f9 [origin/develop] develop
  feature_1 b41865d [origin/feature_1] feature_1
* master    1399706 [origin/master] init commit
```

master 已經跟踪了 origin/master 了

3.7.2 實際命令摘録，tortoiseGit

3.8 git.exe pull --progress -v --no-rebase "origin" hexo-next-Pisces

```
git.exe pull --progress -v --no-rebase "origin" hexo-next-Pisces
```

```
From github.com:kevinluolog/hexo-klblog-src
* branch            hexo-next-Pisces -> FETCH_HEAD
```

```
= [up to date]      hexo-next-Pisces -> origin/hexo-next-Pisces
Already up to date.

Success (7800 ms @ 2019/10/27 星期日 8:50:58)
```

3.8.1 雜散點滴

- git clone -b gh-pages https://$GH_TOKEN_FULL@github.com/kevinluolog/gp-memo.git：不寫目標目録時，會把 repo 名 gp-memo 作爲目録名
- git clone -b gh-pages https://$GH_TOKEN_FULL@github.com/kevinluolog/gp-memo.git /tmp/gp-memo：寫目標目録時，不會自動把 repo 名 gp-memo 作爲目録名，需要顯式地寫上，要不會把 repo 內容直接寫入目標目録。
- git commit --allow-empty -m ”kl+travis+” : --allow-empty 讓 commit 相同時不返回錯 exit(1), 如 travis CI 不會報錯