github

kevinluo

Contents

contents

1 参考链接

git-scm 官方 doc 网址

Git push 常见用法

git add -A 和 git add . 的区别

git 命令之 git remote 的用法

git 跟踪远程分支，查看本地分支追踪和远程分支的关系

git 跟踪远程分支，查看本地分支追踪和远程分支的关系简书

Git 本地分支与远程分支的追踪关系

2 工具

git 恢复保存 REPO 文件时间信息的工具----------

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

2.1 metastore 额外加上文件 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 恢复文件时间等功能 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 参考链接 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/
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 克隆更改文件修改时间 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 经验点滴

3.0.1 命令

3.1 git clone

```
# 注意: 如果直接显式指明clone目标目录，则一定要把repo名字写上，不然不会自动加上； 如果省略,则
# <directory>The name of a new directory to clone into. The "humanish"  part of the source reposi

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

3.2 git add

- git add -A 提交所有变化
- git add -u 提交被修改 (modified) 和被删除 (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，不包括被删除 (deleted) 文件

```
git add .: 他会监控工作区的状态树，使用它会把工作时的所有变化提交到暂存区，包括文件内容修改(
git add -u : 他仅监控已经被add的文件（即tracked  file），他会将被修改的文件提交到暂存区。add
git add -A : 是上面两个功能的合集（git add --all的缩写）
```

3.3 git commit

```
# git commit -m 'message'
# -m 参数表示可以直接输入后面的"message"，如果不加 -m参数，那么是不能直接输入mess  age的，而
# message即是我们用来简要说明这次提交的语句。
# git commit -am 'message' -am等同于-a -m
# -a参数可以将所有已跟踪文件中的执行修改或删除操作的文件都提交到本地仓库，即使它们  没有经过g
# 注意: 新加的文件（即没有被git系统管理的文件）是不能被提交到本地仓库的。
# --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的一般形式为 git push <远程主机名> <本地分支名> <远程分支名>，例如  git push origin
# git push origin master
# 如果 远程分支被省略，如上则表示将本地分支推送到与之存在追踪关系的远程分支（通常两者同名），
# git push origin : refs/for/master
```

```
# 如果省略本地分支名，则表示删除指定的远程分支，因为这等同于推送一个空的本地分支到 远程分支，
delete master
# git push origin
# 如果当前分支与远程分支存在追踪关系，则本地分支和远程分支都可以省略，将当前分支推送到origin
# git push
# 如果当前分支只有一个远程分支，那么主机名都可以省略，形如 git push，可以使用git branch -r ，查
#  关于 refs/for:
# refs/for 的意义在于我们提交代码到服务器之后是需要经过code review  之后才能进行merge的，而re
# 原文链接：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 获取目录下文件名

显示所有文件

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: 缺省把汉字等字符以\转义码输出，加z表示以正常显示字符输出,但是加z时没有分行
--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” 文件提交时间

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

查看文件最后一行：tail -1 文件名，后面必须是文件，或者 | tail -1 管道输出的内容

显示各纯文件名：

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

显示各文件首次COMMIT时间,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 filename
```

输出格式：可以直接被 touch 参数 --date ""识别

```
2019-09-26 15:09:54 +0800
```

```
# touch 回创建时间
# 下面去掉bash -c 就能工作了。 xargs可以直接传参数给touch使用的。
# 整个过程就是用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
# 这个文件修改时间更新好后，还需要hexo的一个脚本，在渲染前把创建时间设置为修改时  间。因为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
```

网上参考源码片段

```
#? ? 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="{}"'
```

网上参考源码，sh 批处理

```
# 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 跟踪远程分支

从当前分支切换到‘dev’分支：

```
git checkout dev
```

建立并切换新分支：

```
git checkout -b 'dev'
```

查看当前详细分支信息（可看到当前分支与对应的远程追踪分支）：

```
git branch -vv
```

查看当前远程仓库信息

```
git remote -vv
```

如果用 git push 指令时，当前分支没有跟踪远程分支（没有和远程分支建立联系），那么就会 git 就会报错

There is no tracking information for the current branch. Please specify which branch you want to merge with. 因为当前分支没有追踪远程指定的分支的话，当前分支指定的版本快照不知道要作为服务器哪一个分支的版本快照的子节点。简单来说就是：不知道要推送给哪一个分支。那么如何建立远程分支：

克隆时自动将创建好的 master 分支追踪 origin/master 分支

```
git clone 服务器地址
git checkout -b develop origin/develop
```

在远程分支的基础上建立 develop 分支，并且让 develop 分支追踪 origin/develop 远程分支。

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

将 branch-name 分支追踪远程分支 origin/branch-name

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

设置当前分支跟踪远程分支 origin/serverfix

查看本地分支和远程分支的跟踪关系

```
git branch -vv
```

比如输入

```
$ 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，且当前分支为 master 分支

那么假如我此时想要将 master 的改变推送到 origin 服务器的 master 分支上：

```
$ git checkout master//切换到master分支
...
$ git branch -u origin/master//将当前分支跟踪origin/master
```

Branch 'master' set up to track remote branch 'master' from 'origin'. 之后就可以执行 git add 和 git commit 了

现在再查看一下本地和远程的分支关系：

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

master 已经跟踪了 origin/master 了

3.7.2 实际命令摘录，**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 离散点滴

- git clone -b gh-pages https://$GH_TOKEN_FULL@github.com/kevinluolog/gp-memo.git：不写目标目录时，会把 repo 名 gp-memo 作为目录名
- git clone -b gh-pages https://$GH_TOKEN_FULL@github.com/kevinluolog/gp-memo.git /tmp/gp-memo：写目标目录时，不会自动把 repo 名 gp-memo 作为目录名，需要显式地写上，要不会把 repo 内容直接写入目标目录。
- git commit --allow-empty -m "kl+travis+"：--allow-empty 让 commit 相同时不返回错 exit(1), 如 travis CI 不会报错