git-tutorial

Spike

2021 年 9 月 6 日

准备

- 安装 Git
- 配置 Git
  - git config –global user.name "Your Name" 设置你的名称
  - git config –global user.email yourname@example.com 设置你的邮箱
  - git config –global core.editor nano 设置编辑器，写提交信息时会用到
  - git config –list –show-origin 查看配置存储的位置
- 一个 Gitlab 帐号

Motivation

为什么要用版本管理

- 记录快照，可以随时恢复快照
- 支持分支功能: 可以同时进行多个功能开发而不相互影响
- ...

为什么用 Git

- 安装简单
- 非常流行，被广泛支持
- 分布式的

Git vs SVN

Git

- 分布式的，离线的，随时可以提交
- 分支操作很简单，切换很方便
- 可以完全使用命令行不依赖 UI，且文档比较全

SVN

- 集中式的，必须连上 SVN 服务器才能提交
- 更细的权限控制

什么是 Git

- Git 是一个版本控制工具
- 实际上是项目中的 .git 文件夹，里面就是 git 的本地仓库

使用 Git 跟踪麦当劳汉堡制作

初始化

```
$ mkdir mcdonald
$ cd mcdonald
$ git init
```

查看当前 Git 状态

```
$ git status

On branch trunk

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

创建两个文件：制作步骤 & 材料

steps.txt

- 放一片面包
- 放牛肉
- 放蔬菜
- 挤番茄酱
- 放另一片面包

ingredients.txt

- 面包
- 牛肉
- 蔬菜
- 番茄酱

查看仓库状态 (git status)

git status 是你的好朋友，你能通过他弄明白当前仓库状态是怎么样的

git status

```
$ git statusa
On branch trunk

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ingredients.txt
    steps.txt

nothing added to commit but untracked files present (use "git add" to track)
```

通过 git status 可以看到当前新增的两个文件是未跟踪状态 (Untracked)

让 Git 跟踪文件 (git add)

git add

```
$ git add steps.txt
$ git add -p ingredients.txt
$ git status
  On branch trunk

  No commits yet

  Changes to be committed:
    (use "git rm --cached <file>..." to unstage)
      new file:   ingredients.txt
      new file:   steps.txt
```

现在 Git 已经跟踪到这些文件了，它们被放到了暂存区 (staged)，接下来就可以修改提交了

提交改动，生成快照 (git commit)

git commit

```
$ git commit -m "adding steps and ingredients"
  [trunk (root-commit) 06dd8ec] adding steps and ingredients
   2 files changed, 10 insertions(+)
   create mode 100644 ingredients.txt
   create mode 100644 steps.txt
```

git status

```
$ git status
  On branch trunk
  nothing to commit, working tree clean
```

查看改动记录 (git log)

git log

```
$ git log
commit 06dd8ec0bcd79a534d0c9c0dba9c7a7752975717 (HEAD -> trunk)
Author: Spike <l-yanlei@hotmail.com>
Date:   Sun Sep 5 13:15:09 2021 +0800

adding steps and ingredients
```

git log 美化

git config –global alias.lg ”log –color –graph –pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' –abbrev-commit”

加一片芝士

steps.txt

- 放一片面包
- 放牛肉
- 放蔬菜
- 挤番茄酱
- 加一片芝士
- 放另一片面包
- 吃掉

ingredients.txt

- 面包
- 牛肉
- 蔬菜
- 番茄酱
- 芝士

查看更改 (git diff)

提交前最好检查一下改动，是否都需要提交，有没有一些测试代码等

git diff

```
$ git diff

...
diff --git a/steps.txt b/steps.txt
index 8ad4eed..c2046a5 100644
--- a/steps.txt
+++ b/steps.txt
@@ -2,4 +2,6 @@
 ...
 - 挤番茄酱
+- 加一片芝士
 - 放另一片面包
+- 吃掉
```

提交更改

```
$ git status
$ git add ingredients.txt
$ git status
$ git commit -m " 增加芝士"
$ git status
$ git add steps.txt
$ git commit                    # <-- we have left out -m "..."
```

回顾 Q&A

生成快照的基本步骤

```
$ git add <file(s)>
$ git commit
```

常用指令

```
$ git init     # initialize new repository 初始化仓库
$ git add      # add files or stage file(s) 暂存文件
$ git commit   # commit staged file(s) 提交修改
$ git status   # see what is going on 查看仓库状态
$ git log      # see history 查看提交记录
$ git diff     # show unstaged/uncommitted modifications 查看改动
```

Q & A

有什么疑问?

提交记录的重要性

代码固然重要，但提交记录也是同样重要的，通过看提交记录，我们可以知道目前的代码是怎么来的。

Q & A

- 一个提交应该包含什么东西?
- 这些 提交信息 有什么问题?

暂存区 Staging Area

有的时候，做了改动，但还不想提交，但想先存一下当前的状态，怎么办？ => 暂存区 Staging-basic

相关命令

```
$ git add <path>          # stages all changes in file
$ git add -p <path>       # stages while letting you choose which lines to take
$ git commit              # commits the staged change
$ git diff                # see **unstaged** changes
$ git diff --staged       # see **staged** changes
$ git rm                  # removes a file
$ git reset               # unstages staged changes
                          # in latest Git: git restore --staged <path>
$ git checkout <path>     # check out the latest staged version ( or committed
                          # version if file has not been staged )
                          # in latest Git: git restore <path>
```

Q & A

- SVN 没有暂存区的概念，有什么问题?

暂存区例子 1

搬家

- 你要搬家，你有一些箱子用来打包行李
- 你可以把任何东西放进箱子，也可以从里面取出来
- 你不会把厨房的东西，和卧室的东西都放到一个箱子里
- 这里的箱子，对应的就是 Git 中的 暂存区
- 而 提交 相当于你给箱子贴上一个标签
- 你写的标签，不会是 一些东西 ，而是 电脑配件, 餐具

暂存区例子 2

购物

- 搬家好了，你要去买一些物品，一些是 生活用品 ，一些是 食品
- 你分别列了两张清单
- 你去到超市逛一遍，找到所有你要买的东西放到购物车
- 到收银台，你要检查两张清单上的东西是否都买了
- 你拿出 生活用品清单，从购物车找到对应的物品放到传送带 (git add)，都放放好后你会再检查一下 (git diff –staged)
- 都没问题，你就会付款（git commit ）
- 再重复步骤，处理 食品清单

暂存区用法 1: 一次性做了很多改动

- 你一次性改了很多东西，但你想分开提交（为什么?）
- git add 暂存这次要提交的文件
- git diff & git diff –staged 检查这些改动是否正确
- git commit 提交
- 重复上面步骤

暂存区用法 2：你想做一些记录点

- 你想在最终提交前，保存你现在在做的东西
- git add 把你认为写的没问题的东西放到暂存区
- git checkout 然后你改了些东西，发现写的不好，你要回到上次写的好的状态，则从暂存区中读取
- git commit 最终达到一个你想提交的状态，就提交（形成一个提交的基本单元，包含的不是特别多改动，也不是特别少）
- 什么叫多，什么叫少?

练习：牛肉成本高，把牛肉换成鸡肉

steps.txt

- 放一片面包
- 放鸡肉
- 放蔬菜
- 挤番茄酱
- 加一片芝士
- 放另一片面包
- 吃掉

ingredients.txt

- 面包
- 鸡肉
- 蔬菜
- 番茄酱
- 芝士

暂存区练习

```
$ git diff
$ git add
$ git status
$ git diff
$ git diff --staged
$ git commit
```

Q & A

暂存区可以让我们更好的去构造一个提交单元。

- 有什么问题

撤销 & 恢复 Undoing and recovering

- 回退到某次提交，丢弃最近的改动 (git reset –hard)
- 选择某个提交，回滚这个提交做的改动 (git revert)
- 提交后，立即对这个提交进行修改 (git commit –amend)

撤销还没有提交的改动

Staging-basic

- git checkout -f master 撤销所有工作区的改动，回到 master 当前的暂存区状态
- git checkout -p | git checkout $file 选择性的撤销改动
- git reset -p | git reset $file 选择性地撤销工作区和暂存区的改动

练习

- 在 steps.txt 上做一些改动
- git status 看看当前状态
- git add steps.txt 将改动暂存
- git status 看看当前状态
- 继续在 steps.txt 上做一些改动
- git diff & git diff –staged 看看
- 然后 git checkout -p | git checkout steps.txt 看看会发生什么
- git status 看看当前状态
- git diff & git diff –staged 看看

回滚修改 Reverting commits

git revert

- git revert <hash> 回滚某个提交
- 回滚提交时，会生成一个新的提交，而不会修改原来的提交记录 (有什么好处?)

练习

- 对 steps.txt 和 ingredients.txt 做一些修改，然后提交
- git revert <hash> 回退这些修改
- git lg 看看提交记录
- git show <hash> 看看回退的提交做了什么

提交后马上修改提交 (git commit –amend)

场景

- 提交后发现忘了提交某个文件
- 发现提交信息不对

命令

git commit –amend

注意

- 仅对最新的提交有效
- 这个命令会修改提交的 hash，会修改提交记录，相当于把原来的提交换成一个新的提交

练习 (git commit –amend)

- 对 steps.txt 和 ingredients.txt 做一些修改并提交 (git commit)
  - 给你的汉堡包加些东西
- git lg 看看提交日志
- 对 steps.txt 和 ingredients.txt 做一些修改，暂存 (git add)
  - 你是不是忘了加某种酱了，加上吧
- git commit –amend 修改最新的提交，把新作的改动加入进去
- git lg 看看提交日志

将提交记录重置到某个提交节点 (git reset)

命令

git reset <hash> –hard

注意

- 这会改动你的提交记录，并且会将工作区和暂存区的内容，重置为你指定的提交的状态
- 如果你还有没提交的内容，使用这个命令要小心，会导致你的改动丢失

练习 (git reset)

- git lg 查看提交记录，找到一个你想回去的状态，记住它的 commit hash
- git reset <hash> –hard 回退到这个提交
- git lg 看看当前的提交记录
- git reflog 查看操作日志, 找到 reset 操作的 commit hash
- git reset <hash> –hard
- git lg

研发新的汉堡！

- 之前做的汉堡销量不错，现在我们打算改进一下汉堡，看看能不能做的更好
- 尝试改进一下面包
- 尝试改进肉
- 尝试改进酱料
- 由于现在的汉堡一直在卖，不能把它下架
- 如果改进的好，就把它们融合起来

分支 (git branch)

Git Branching

常用命令

```
git branch            # 查看当前所在的分支
git branch to from    # 创建从 from 分支创建一条 to 分支
git checkout to       # 切换到 to 分支
```

git graph 查看所有分支的提交记录

```
git config --global alias.graph "log --all --graph --decorate --oneline"
```

练习 (git branch)

创建一条面包改进分支改进面包

- git branch feat-bread trunk 从 主分支 创建一条 feat-bread 分支
- git checkout feat-bread 切换到 feat-bread
- 修改 ingredients.txt 中的面包，改为 全麦面包, 并提交修改 (git commit)

创建一条肉改进分支改进肉

- 回到主分支 git checkout master
- 从 主分支 创建一条 feat-meat 分支
- 修改 ingredients.txt 中的牛肉，改为 M9 和牛, 并提交修改 (git commit)

查看当前的提交记录

- git graph

合并分支 (git merge)

面包改进和肉改进都获得一致好评，是时候出一个新的汉堡了！把这些改进都融合起来吧。

合并面包的改进

```
git checkout trunk # 切换到我们要整合的分支
git merge feat-bread # 合并面包改进的分支
git graph # 查看一下合并后的记录
```

合并肉的改进

```
git checkout trunk # 切换到我们要整合的分支
git merge feat-meat # 合并面包改进的分支
git graph # 查看一下合并后的记录
```

删除分支 (git branch -d)

- 功能分支合并后，基本没啥用处，可以把分支删除掉
- 实际上，分支上的提交不会被删除，只是移除了 分支的指针

命令

```
git branch --merged                    # 查看已经被合并过的分支
git branch -d feat-bread feat-meat     # 删除分支
git branch -D <name>                   # 强行删除分支，不管分支是否已经合并
git graph                              # 删除后看看提交记录，留意分支的变化
```

分支合并策略

Pull Request Merge Strategies: The Great Debate

- Merge
- Rebasing
- Squahing commits

练习：把提交移动到别的分支

在分支上做了很多的提交，突然发现提交错了。

```
# 在当前分支做几个提交
# 发现分支提交错了
$ git branch feat-a                 # 创建一条新的分支，存储这些提交错的改动
$ git reset --hard <hash>           # 将当前分支重置正确的提交
$ git graph                         # 看看当前分支提交记录
```

练习: Rebasing

- git checkout -b feat 创建一条新的分支 feat ，做几个提交
- 切回到主分支，再做几个提交
  - git checkout trunk
  - (git add + git commit) * N
- git graph 看看当前的提交记录
- 切换到 feat 分支，rebase 主分支
  - git checkout feat
  - git rebase trunk
- git graph 看看分支记录

练习: Squashing commits

(Optional) Exercise: Squashing commits

总结

```
git branch                  # 查看当前分支
git branch <name>           # 创建名为 <name> 的分支
git checkout <name>         # 切换分支到 <name>
git checkout -b <name>      # 创建分支 <name> , 并切换到 <name> 分支
git merge <name>            # 合并 <name> 分支到当前分支
git branch -d <name>        # 删除分支 <name>
git branch -D <name>        # 删除没有还没被合并过的分支 <name>
```

常见的工作流 1

```
git checkout -b new-feature   # 创建一条功能分支并切换过去，准备开始做一个功能
git commit                    # 持续地开发，然后提交
                                # 功能完成后进行测试
                                # 测试完成，没啥问题，可以合回到主分支了
git checkout trunk            # 切换到主分支
git merge new-feature         # 将功能分支合并到主分支
git branch -d new-feature     # 删除合并后的功能分支
```

常见的工作流 2

```
git checkout -b wild-idea     # 你有一个大胆的想法，想去尝试一下，创建并切换到分支
                                # 持续地开发，提交
                                # 搞半天你发现没啥用，你不想要了
git checkout trunk            # 切回到主分支
git branch -D wild-idea       # 将这个大胆的想法舍弃掉
```

冲突解决

解决冲突的步骤

- git status & git diff 查看冲突情况
- 打开冲突的文件，根据情况，决定要保留谁的代码，移除冲突标记
- git status & git diff 查看冲突情况
- 移除冲突标记，没啥问题，git add $file 把冲突文件加入到暂存区，告诉 Git 冲突解决
- git status 查看情况
- git commit 把合并的代码提交

放弃合并

看 Git 给出的提示，可以使用 git merge –abort 放弃合并的动作

远程仓库 (git remote)

- Git 远程操作详解

命令

- git clone < 远程仓库 URL> 拉取远程仓库到本地
- git-remote 设置远程仓库，增加，重命名，删除等
- git fetch 拉取远程的更新到本地仓库
- git pull 拉取远程更新到本地仓库，并且与工作区的内容合并
- git pull –rebase 拉取到本地后，采用 rebase 的方式和工作区进行合并
- git push 把本地的提交推送到远程仓库
- git push -u origin trunk 设置默认上有为 origin, 并且把分支 trunk 推送到 origin

查找提交记录

- git grep 什么文件中包含你搜索的文字
- git log -S 什么提交包含你搜索的文字（搜素的是文件）
- git log –grep 什么提交包含你搜索的文字（搜索的提交信息）
- git show <hash> 查看某个提交的改动
- git blame $file 查看某个文件的每一行的最后的提交信息
- git checkout -b <hash> 从某个提交节点切换一个分支，查看在这个提交的状态
- git switch –create branchname somehash 等同上面 checkout 的指令，新版本 Git 支持

查找的工作流

- Exercise: basic archaeology commands
- git grep "abc" 找到 abc 所在的文件
- git blame $file 查看文件的每一行的提交记录，然后看看包含 abc 的提交 <hash>
  - 如果在 git blame 中找不到，使用 git log -S "abc" 去找在什么提交中
- git show <hash> 找到提交后，可以看这个提交的改动
- git branch past-code <hash> 创建分支，看这个提交下的项目快照

git bisect 二分法定位问题

- Finding out when something broke/changed with git bisect
- git bisect 命令教程

处理突然插进来的任务

我们在做一个功能，做了一半，突然来一个紧急任务得马上开发完，怎么办?

- git stash
- git branch

方法 1: git stash

git-stash

```
git stash # 将暂存区和工作区的改动”藏起来”，恢复到最近一次提交的状态
git stash save NAME # stash 的同时给个名字，对于会放很久的 stash 是有用的
git stash list  # 查看所有”藏起来”的内容
git stash pop   # 将 stash 栈中第一个 stash 弹出并应用
git stash drop  # 丢弃栈中的第一个 stash
git stash apply # 应用栈中的第一个 stash，但不会弹出；你也可以根据序号，应用其他 stash
git stash clear # 清空 stash
```

git stash 练习

- 做一些改动
- git status 看状态
- git stash
- git stash list 看看 stash 栈中有什么
- git status 看状态
- git graph 看看
- 再做一些改动
- git stash pop 弹出并应用第一个 stash
- git status 看状态
- git stash list 看看 stash 栈

方法 2: 使用分支

```
$ git checkout -b temporary   # 创建并切换分支，那些未提交的修改也还在
$ git add <paths>             # 把这些未提交的修改暂存
$ git commit                  # 提交改动
$ git checkout trunk          # 回到主分支，由于那些为提交的修改，
                              # 已经为提交到 temporary 分支，所以主分支是干净的
                              # 在主分支处理那些突然来的任务

$ git checkout temporary      # 切回到那条临时分支，继续没做完的事
```

提交规范

- 约定式提交
- Writing useful commit messages
- How to Write a Git Commit Message

git cherry pick

命令

- git cherry -v <upstream> <head> 列出：<head> 里有的提交，但 <upstream> 没有的
- git cherry-pick <hash> 把 <hash> pick 到当前分支，你可以一次性 pick 多个

工作流

- git checkout trunk 切换到要要合入的分支
- git cherry trunk dev 看看 dev 上哪些提交是 trunk 没有的，记录对应的 <hash>
- git cherry-pick <hash> pick dev 上的提交到 trunk

链接

git-cherry , git-cherry-pick

git tag

命令

```
git tag                    # 列出所有的 tag
git tag v1.0               # 创建一个叫 v1.0 的 tag
git show v1.0              # 查看 v1.0 的内容
git tag -a v2.0 <hash>     # * 推荐 * 给提交 <hash> 打上 v2.0，并且携带标签信息 (hash 省
git push --tags            # 推送到远程分支时，把 tag 也推送上去
git tag -d <tagname>       # 删除 tag
git checkout v2.0          # 切换到 v2.0
```

链接

- Tagging git tag 步骤
- git-tag git tag 手册
- 语义化版本 2.0.0 版本号规范

git reflog (Reference logs)

- git-reflog
- 你可以通过 git reflog 查看你的操作，例如 reset，commit 等，然后找到对应的 <hash>, 可以还原到 <hash> 的状态

.gitignore 文件

gitignore

- 有的文件或目录是不需要放到 Git 中的，可以在项目根目录创建一个 .gitignore 目录，指定忽略的文件 / 目录
- 例如一些依赖库，一些打包后产生的文件等

参考

- Ignoring files and paths with .gitignore
- gitignore
- git-rm 有的文件已经被 Git 跟踪，如果想忽略，要先从 Git 中移除

Git 工作流

- Git 工作流程
  - git flow
  - github flow
  - gitlab flow

排查问题

- 看 Git 的提示
- Google 和手册是你的好朋友
- 多提交，提交后的改动只要不删掉 .git 文件夹，都能找回来

git-config & .git/config

- git-config 可以通过 git-config 对 Git 进行配置
- 本地的配置，实际保存在 .git/config 中

一些工具

- gitui 一个用 rust 写的命令行 Git 界面

参考链接

- Introduction to version control with Git 一个 Git 教程，本文基本参照这个教程整理的
- Pull Request Merge Strategies: The Great Debate 文章中的图能直观地理解 merge, rebase
- 廖雪峰 Git 教程 一个中文教程，花个三四小时看一遍操作一遍，Git 的基本操作就会了
- 保姆级 Git 入门教程，万字详解 罗列了一些常用的 Git 指令
- Learn Git Branching 像玩游戏一样，学习 Git 的分支操作