02

OPEN ORIENTED

凹 凸 实 验 室

再聊一下git

王彩暖

再聊一下git

02

别名与shell 别名

写好commit信息

过一遍标签

过一遍分支

过一遍log

过一遍diff

简介

* 直接记录快照，而不是与初始版本的差异比较。
* 几乎所有操作是本地执行，在本地磁盘就有项目的完整历史。
* 保证完整性：所有数据在存储前都计算校验和（SHA-1散列，哈希），然后以校验和来引用，意味着Git总是知道更改了任何文件目录内容。
* 一旦提交快照，就难以再丢失数据。

简介

* 三种状态：已修改（modified，表示已修改，还没保存到暂存区域）、已暂存（staged，对已修改文件的当前版本做了标记，使之包含在下次提交的快照中）、已提交（committed，已安全保存在本地数据库；Git目录保存着特定版本文件了）。

* 三个工作区域概念：工作目录（在电脑里能看到的目录）、暂存区域（一个文件：保存了下次将提交的文件列表信息）、Git 仓库（用来保存项目的元数据和对象数据库的地方）。

别名

设置别名

```
git config --global alias.pl pull
git config --global alias.ph push
```

使用别名

```
git pl
git ph
```

shell别名

要显示所有别名，终端输入alias即可：

shell别名


```
gcb='git checkout -b'
gcd='git checkout develop'
gcf='git config --list'
gcl='git clone --recursive'
gclean='git clean -fd'
gcm='git checkout master'
gcmsg='git commit -m'
'gcn!'='git commit -v --no-edit --amend'
gco='git checkout'
gcount='git shortlog -sn'
gcp='git cherry-pick'
gcpa='git cherry-pick --abort'
gcpc='git cherry-pick --continue'
gcs='git commit -S'
gcsm='git commit -s -m'
gd='git diff'
gdca='git diff --cached'
gdct='git describe --tags `git rev-list --tags --max-count=1`'
gdt='git diff-tree --no-commit-id --name-only -r'
gdw='git diff --word-diff'
gf='git fetch'
gfa='git fetch --all --prune'
gfo='git fetch origin'
gg='git gui citool'
gga='git gui citool --amend'
ggpull='git pull origin $(git_current_branch)'
ggpur=ggu
ggpush='git push origin $(git_current_branch)'
ggsup='git branch --set-upstream-to=origin/$(git_current_branch)'
ghh='git help'
```

shell别名

例如:

```
ga='git add'
gcmsg='git commit -m'
gcam='git commit -a -m'

ggpull='git pull origin $(git_current_branch)'
ggpush='git push origin $(git_current_branch)'
gp='git push'

gf='git fetch'
gm='git merge'
```

shell别名

设置shell别名

1. （如果当前终端关闭的话所设置过的别名也就无效了）：
alias gpl="git pull"

2. 更常用的做法是把它加入到~/.zshrc中：
vim ~/.zshrc
添加行：
alias gpl="git pull"

写好commit信息

1. git commit 会启动文本编辑器以便输入本次提交的说明；
默认的提交消息包含最后一次运行 git status 的输出，放在注释行里；
另外开头还有一空行，供你输入提交说明。

2. git commit -v 会**将你所做的改变的 diff 输出放到编辑器中**从而使你知道本次提交具体做了哪 些修改。

3. git commit -m "提交的信息" **将提交信息与命令放在同一行**。

4. 给 git commit 加上 -a 选项，Git 就会自动把所有已经跟踪过的文件暂存起来一并提交，从而**跳过 git add 步骤**。

备注：git config --global core.editor vim 可设定编辑软件为vim

写好commit信息

02

git commit -m "**提交的信息**"

写好commit信息

约定规范：

提交的信息（标题） = type（类型） + scope（范围，可选） + subject（描述）

写好commit信息

约定规范：

提交的信息（标题） = type（类型） + scope（范围，可选） + subject（描述）

既然要写标题，总得有个主题，于是要求本身提交的代码粒度要小，
尽量保证这个 commit 只做一件事情，否则很难描述清楚。

写好commit信息

type 说明 commit 的类别，使用下面7个标识：

feat：新功能（feature）
fix：修补bug
docs：文档（documentation）
style：格式（不影响代码运行的变动）
refactor：重构（即不是新增功能，也不是修改bug的代码变动）
test：增加测试
chore：构建过程或辅助工具的变动

写好commit信息

scope 用于说明 commit 影响的范围，
比如数据层、控制层、视图层等等，视项目不同而不同。

写好commit信息

subject commit目的的简短描述，不超过50个字符（为了好看以及精简）。

以动词开头（祈使语气），使用第一人称现在时，比如change，而不是changed或changes
结尾不加句号（.）
首字母大写

写好commit信息

02

正文

用一个空行隔开标题和正文（为了好看）。
正文在 72 个字符处换行（为了好看）。
使用正文解释是什么和为什么，而不是如何做：

改动前是如何工作的（和出了什么问题），
现在是如何工作的，
以及为什么你要采用这种方法解决问题。

写好commit信息

还有...

尝试给提交的信息插入表情~ 😝

Commit message工具

标签

02

```
# 列出所有tag
$ git tag

# 查看tag信息
$ git show [tag]
```

标签

02

```
# 列出所有tag
$ git tag

# 查看tag信息
$ git show [tag]
```

```
# 新建一个tag在指定commit
$ git tag [tag] [commit]

# 删除本地tag
$ git tag -d [tag]

# 删除远程tag
$ git push origin :refs/tags/[tagName]
```

标签

02

```
# 列出所有tag
$ git tag

# 查看tag信息
$ git show [tag]
```

```
# 新建一个tag在指定commit
$ git tag [tag] [commit]

# 删除本地tag
$ git tag -d [tag]

# 删除远程tag
$ git push origin :refs/tags/[tagName]
```

```
# 提交指定tag
$ git push [remote] [tag]

# 提交所有tag
$ git push [remote] --tags

# 新建一个分支，指向某个tag
$ git checkout -b [branch] [tag]
```

分支

02

```
# 列出所有本地分支
$ git branch

# 列出所有远程分支
$ git branch -r

# 列出所有本地分支和远程分支
$ git branch -a
```

分支


```
# 列出所有本地分支
$ git branch

# 列出所有远程分支
$ git branch -r

# 列出所有本地分支和远程分支
$ git branch -a
```

```
# 新建一个分支，但依然停留在当前分支
$ git branch [branch-name]

# 新建一个分支，并切换到该分支
$ git checkout -b [branch]

# 新建一个分支，指向指定commit
$ git branch [branch] [commit]

# 新建一个分支，与指定的远程分支建立追踪关系
$ git branch --track [branch] [remote-branch]
```

分支


```
# 切换到指定分支，并更新工作区
$ git checkout [branch-name]

# 切换到上一个分支
$ git checkout -

# 建立追踪关系，在现有分支与指定的远程分支之间
$ git branch --set-upstream [branch] [remote-branch]

# 合并指定分支到当前分支
$ git merge [branch]

# 删除分支
$ git branch -d [branch-name]

# 删除远程分支
$ git push origin --delete [branch-name]
```

log

git log 默认按提交时间列出所有的更新，最近的更新排在最上面。

git log -p 用来显示每次提交的内容差异**diff**。
git log -n **仅显示最近n次的提交**。
git log --stat： 看每次提交的**简略的统计信息**；
在每次提交的下面**列出所有被修改过的文件**、
有多少文件被修改了以及被修改过的文件的哪些行被移除或是添加了；
在每次提交的最后还有一个总结。

log

git log 的常用选项 列：

选项	说明
-p	按补丁格式显示**每个更新之间的差异**。
--stat	显示每次更新的**文件修改统计信息**。
--shortstat	不显示 --stat 中列出d的所有被修改过的文件。
--name-only	仅在提交信息后显示已修改的文件清单。
--name-status	显示新增、修改、删除的文件清单。
--abbrev-commit	仅显示 SHA-1 的前几个字符，而非所有的 40 个字符。
--relative-date	使用较短的相对时间显示（比如，“2 weeks ago”）。
--graph	显示 ASCII 图形表示的分支合并历史。
--pretty	**使用其他格式显示历史提交信息**。可用的选项包括 oneline，short，full，fuller 和 format（后跟指定格式）。

log

git log --pretty=format 常用的选项,
如git log --pretty=format:"%h - %an, %ar : %s"

选项 说明
%H 提交对象（commit）的完整哈希字串
%h 提交对象的简短哈希字串
%T 树对象（tree）的完整哈希字串
%t 树对象的简短哈希字串
%P 父对象（parent）的完整哈希字串
%p 父对象的简短哈希字串
%an 作者（author）的名字
%ae 作者的电子邮件地址
%ad 作者修订日期（可以用 --date= 选项定制格式）
%ar 作者修订日期，按多久以前的方式显示
%cn 提交者(committer)的名字
%ce 提交者的电子邮件地址
%cd 提交日期
%cr 提交日期，按多久以前的方式显示
%s 提交说明

diff

02

```
# 显示暂存区和工作区的差异
$ git diff

# 显示暂存区和上一个commit的差异
$ git diff --cached [file]

# 显示工作区与当前分支最新commit之间的差异
$ git diff HEAD

# 显示两次提交之间的差异
$ git diff [first-branch]...[second-branch]

# 显示今天你写了多少行代码
$ git diff --shortstat "@{0 day ago}"
```

02

OPEN ORIENTED

凹 凸 实 验 室

THANKS FOR LISTENING

王彩暖