**索引**

* [配置](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E9%85%8D%E7%BD%AE)
* [配置文件](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E9%85%8D%E7%BD%AE%E6%96%87%E4%BB%B6)
* [创建](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E5%88%9B%E5%BB%BA)
* [本地修改](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E6%9C%AC%E5%9C%B0%E4%BF%AE%E6%94%B9)
* [搜索](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E6%90%9C%E7%B4%A2)
* [提交历史](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E6%8F%90%E4%BA%A4%E5%8E%86%E5%8F%B2)
* [分支与标签](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E5%88%86%E6%94%AF%E4%B8%8E%E6%A0%87%E7%AD%BE)
* [更新与发布](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E6%9B%B4%E6%96%B0%E4%B8%8E%E5%8F%91%E5%B8%83)
* [合并与重置](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E5%90%88%E5%B9%B6%E4%B8%8E%E9%87%8D%E7%BD%AE)
* [撤销](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E6%92%A4%E9%94%80)
* [Git Flow](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#Git-Flow)

**配置**

**列出当前配置：**

$ git config --list

**列出repository配置：**

$ git config --local --list

**列出全局配置：**

$ git config --global --list

**列出系统配置：**

$ git config --system --list

**设置用户名：**

$ git config --global user.name “[firstname lastname]”

**设置用户邮箱：**

$ git config --global user.email “[valid-email]”

**设置git命令输出为彩色：**

$ git config --global color.ui auto

**设置git使用的文本编辑器设：**

$ git config --global core.editor vi

**配置文件**

**Repository配置对应的配置文件路径[--local]：**

<repo>/.git/config

**用户全局配置对应的配置文件路径[--global]：**

~/.gitconfig

**系统配置对应的配置文件路径[--local]：**

/etc/gitconfig

**创建**

**复制一个已创建的仓库:**

*# 通过 SSH*

$ git clone ssh://user@domain.com/repo.git

*#通过 HTTP*

$ git clone http://domain.com/user/repo.git

**创建一个新的本地仓库:**

$ git init

**本地修改**

**显示工作路径下已修改的文件：**

$ git status

**显示与上次提交版本文件的不同：**

$ git diff

**把当前所有修改添加到下次提交中：**

$ git add .

**把对某个文件的修改添加到下次提交中：**

$ git add -p <file>

**提交本地的所有修改：**

$ git commit -a

**提交之前已标记的变化：**

$ git commit

**附加消息提交：**

$ git commit -m 'message here'

**提交，并将提交时间设置为之前的某个日期:**

git commit --date="`date --date='n day ago'`" -am "Commit Message"

**修改上次提交**

请勿修改已发布的提交记录!

$ git commit --amend

**修改上次提交的committer date：**

GIT\_COMMITTER\_DATE="date" git commit --amend

**修改上次提交的author date：**

git commit --amend --date="date"

**把当前分支中未提交的修改移动到其他分支：**

git stash

git checkout branch2

git stash pop

**将 stashed changes 应用到当前分支：**

git stash apply

**删除最新一次的 stashed changes：**

git stash drop

**搜索**

**从当前目录的所有文件中查找文本内容：**

$ git grep "Hello"

**在某一版本中搜索文本：**

$ git grep "Hello" v2.5

**提交历史**

**从最新提交开始，显示所有的提交记录（显示hash， 作者信息，提交的标题和时间）：**

$ git log

**显示所有提交（仅显示提交的hash和message）：**

$ git log --oneline

**显示某个用户的所有提交：**

$ git log --author="username"

**显示某个文件的所有修改：**

$ git log -p <file>

**仅显示远端分支与远端分支提交记录的差集：**

$ git log --oneline <origin/master>..<remote/master> --left-right

**谁，在什么时间，修改了文件的什么内容：**

$ git blame <file>

**显示reflog：**

$ git reflog show

**删除reflog：**

$ git reflog **delete**

**分支与标签**

**列出所有的分支：**

$ git branch

**列出所有的远端分支：**

$ git branch -r

**切换分支：**

$ git checkout <branch>

**创建并切换到新分支:**

$ git checkout -b <branch>

**基于当前分支创建新分支：**

$ git branch <**new**-branch>

**基于远程分支创建新的可追溯的分支：**

$ git branch --track <**new**-branch> <remote-branch>

**删除本地分支:**

$ git branch -d <branch>

**强制删除一个本地分支：**

将会丢失未合并的修改！

$ git branch -D <branch>

**给当前版本打标签：**

$ git tag <tag-name>

**给当前版本打标签并附加消息：**

$ git tag -a <tag-name>

**更新与发布**

**列出当前配置的远程端：**

$ git remote -v

**显示远程端的信息：**

$ git remote show <remote>

**添加新的远程端：**

$ git remote add <remote> <url>

**下载远程端版本，但不合并到HEAD中：**

$ git fetch <remote>

**下载远程端版本，并自动与HEAD版本合并：**

$ git remote pull <remote> <url>

**将远程端版本合并到本地版本中：**

$ git pull origin master

**以rebase方式将远端分支与本地合并：**

git pull --rebase <remote> <branch>

**将本地版本发布到远程端：**

$ git push remote <remote> <branch>

**删除远程端分支：**

$ git push <remote> :<branch> (since Git v1.5.0)

*# or*

git push <remote> --delete <branch> (since Git v1.7.0)

**发布标签:**

$ git push --tags

**合并与重置(Rebase)**

**将分支合并到当前HEAD中：**

$ git merge <branch>

**将当前HEAD版本重置到分支中:**

请勿重置已发布的提交!

$ git rebase <branch>

**退出重置:**

$ git rebase --abort

**解决冲突后继续重置：**

$ git rebase --**continue**

**使用配置好的merge tool 解决冲突：**

$ git mergetool

**在编辑器中手动解决冲突后，标记文件为已解决冲突：**

$ git add <resolved-file>

$ git rm <resolved-file>

**合并提交：**

$ git rebase -i <commit-just-before-first>

把上面的内容替换为下面的内容：

原内容：

pick <commit\_id>

pick <commit\_id2>

pick <commit\_id3>

替换为：

pick <commit\_id>

squash <commit\_id2>

squash <commit\_id3>

**撤销**

**放弃工作目录下的所有修改：**

$ git reset --hard HEAD

**移除缓存区的所有文件（i.e. 撤销上次git add）:**

$ git reset HEAD

**放弃某个文件的所有本地修改：**

$ git checkout HEAD <file>

**重置一个提交（通过创建一个截然不同的新提交）**

$ git revert <commit>

**将HEAD重置到指定的版本，并抛弃该版本之后的所有修改：**

$ git reset --hard <commit>

**用远端分支强制覆盖本地分支：**

git reset --hard <remote/branch> e.g., upstream/master, origin/my-feature

**将HEAD重置到上一次提交的版本，并将之后的修改标记为未添加到缓存区的修改：**

$ git reset <commit>

**将HEAD重置到上一次提交的版本，并保留未提交的本地修改：**

$ git reset --keep <commit>

**删除添加.gitignore文件前错误提交的文件：**

$ git rm -r --cached .

$ git add .

$ git commit -m "remove xyz file"

**Git-Flow**

**索引**

* [安装](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E5%AE%89%E8%A3%85)
* [开始](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E5%BC%80%E5%A7%8B)
* [特性](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E7%89%B9%E6%80%A7)
* [做一个release版本](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E5%81%9A%E4%B8%80%E4%B8%AArelease%E7%89%88%E6%9C%AC)
* [紧急修复](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#%E7%B4%A7%E6%80%A5%E4%BF%AE%E5%A4%8D)
* [Commands](https://gold.xitu.io/post/584650a4128fe1006c57e7f2#Commands)

**安装**

* 你需要有一个可以工作的 git 作为前提。
* Git flow 可以工作在 OSX, Linux 和 Windows之下

**OSX Homebrew:**

$ brew install git-flow

**OSX Macports:**

$ port install git-flow

**Linux:**

$ apt-get install git-flow

**Windows (Cygwin):**

安装 git-flow, 你需要 wget 和 util-linux。

$ wget -q -O - --no-check-certificate https:*//github.com/nvie/gitflow/raw/develop/contrib/gitflow-installer.sh | bash*

**开始**

* 为了自定义你的项目，Git flow 需要初始化过程。
* 使用 git-flow，从初始化一个现有的 git 库内开始。
* 初始化，你必须回答几个关于分支的命名约定的问题。建议使用默认值。

git flow init

**特性**

* 为即将发布的版本开发新功能特性。
* 这通常只存在开发者的库中。

**创建一个新特性:**

下面操作创建了一个新的feature分支，并切换到该分支

git flow feature start MYFEATURE

**完成新特性的开发:**

完成开发新特性。这个动作执行下面的操作：

1. 合并 MYFEATURE 分支到 'develop'
2. 删除这个新特性分支
3. 切换回 'develop' 分支

git flow feature finish MYFEATURE

**发布新特性:**

你是否合作开发一项新特性？  
发布新特性分支到远程服务器，所以，其它用户也可以使用这分支。

git flow feature publish MYFEATURE

**取得一个发布的新特性分支:**

取得其它用户发布的新特性分支。

git flow feature pull origin MYFEATURE

**追溯远端上的特性:**

通过下面命令追溯远端上的特性

git flow feature track MYFEATURE

**做一个release版本**

* 支持一个新的用于生产环境的发布版本。
* 允许修正小问题，并为发布版本准备元数据。

**开始创建release版本:**

* 开始创建release版本，使用 git flow release 命令。
* 'release' 分支的创建基于 'develop' 分支。
* 你可以选择提供一个 [BASE]参数，即提交记录的 sha-1 hash 值，来开启动 release 分支。
* 这个提交记录的 sha-1 hash 值必须是'develop' 分支下的。

git flow release start RELEASE [BASE]

创建 release 分支之后立即发布允许其它用户向这个 release 分支提交内容是个明智的做法。命令十分类似发布新特性：

git flow release publish RELEASE

(你可以通过  
git flow release track RELEASE 命令追溯远端的 release 版本)

**完成 release 版本:**

完成 release 版本是一个大 git 分支操作。它执行下面几个动作：

1. 归并 release 分支到 'master' 分支。
2. 用 release 分支名打 Tag
3. 归并 release 分支到 'develop'
4. 移除 release 分支。

git flow release finish RELEASE

不要忘记使用git push --tags将tags推送到远端

**紧急修复**

紧急修复来自这样的需求：生产环境的版本处于一个不预期状态，需要立即修正。有可能是需要修正 master 分支上某个 TAG 标记的生产版本。

**开始 git flow 紧急修复:**

像其它 git flow 命令一样, 紧急修复分支开始自：

$ git flow hotfix start VERSION [BASENAME]

VERSION 参数标记着修正版本。你可以从 [BASENAME]开始，[BASENAME]`为finish release时填写的版本号

**完成紧急修复:**

当完成紧急修复分支，代码归并回 develop 和 master 分支。相应地，master 分支打上修正版本的 TAG。

git flow hotfix finish VERSION