Git 常用命令速查表

创建版本库

```
$ git clone <url>
$ git init
```

修改和提交

```
$ git status
$ git diff
$ git add .
$ git add <file>...
$ git mv <old> <new>
$ git rm <file>
$ git rm --cached <file>...
$ git commit -m "[branch] description"
$ git commit --amend
```

查看提交历史

```
$ git log
$ git log -p <file>...
$ git blame <file>...
```

创建

复制一个已创建的仓库:

```
$ git clone ssh://user@domain.com/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 Messag
e"
```

修改上次提交 *请勿修改已发布的提交记录!*

```
$ git commit --amend
```

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
```

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>
```

谁，在什么时间，修改了文件的什么内容：

```
$ git blame <file>
```

分支与标签

列出所有的分支：

```
$ git branch
```

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>
```

给当前版本打标签：

```
$ git tag <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
```

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

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>
```

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>
```

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>
```

Git Flow

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]
```

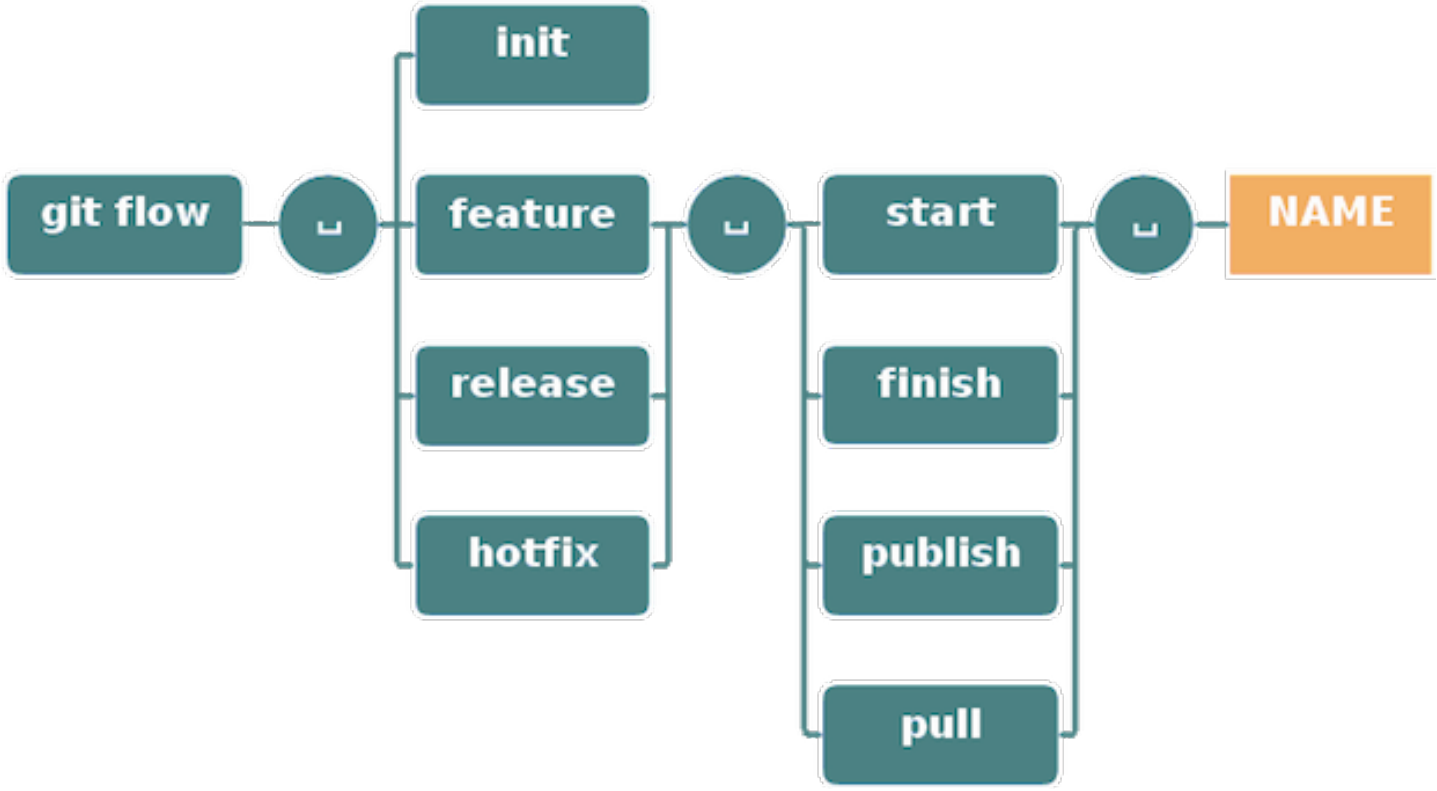
VERSION 参数标记着修正版本。你可以从`[BASENAME]开始，[BASENAME]`为finish release时填写的版本号

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
```

Commands



附录：Cheat Sheet

附上中文的Git Cheat Sheet

Git 常用命令速查表

```
master  :默认开发分支        Head   :默认开发分支
origin  :默认远程版本库      Head^  :Head 的父提交
```

创建版本库

```
$ git clone <url>              #克隆远程版本库
$ git init                     #初始化本地版本库
```

修改和提交

```
$ git status                   #查看状态
$ git diff                     #查看变更内容
$ git add .                    #跟踪所有改动过的文件
$ git add <file>               #跟踪指定的文件
$ git mv <old> <new>           #文件改名
$ git rm <file>                #删除文件
$ git rm --cached <file>       #停止跟踪文件但不删除
$ git commit -m "commit message"
                               #提交所有更新过的文件
$ git commit --amend           #修改最后一次提交
```

查看提交历史

```
$ git log                      #查看提交历史
$ git log -p <file>            #查看指定文件的提交历史
$ git blame <file>             #以列表方式查看指定文件
                               的提交历史
```

撤消

```
$ git reset --hard HEAD        #撤消工作目录中所有未提交
                               文件的修改内容
$ git checkout HEAD <file>     #撤消指定的未提交文件的修
                               改内容
$ git revert <commit>          #撤消指定的提交
```

分支与标签

```
$ git branch                   #显示所有本地分支
$ git checkout <branch/tag>    #切换到指定分支或标签
$ git branch <new-branch>      #创建新分支
$ git branch -d <branch>       #删除本地分支
$ git tag                      #列出所有本地标签
$ git tag <tagname>            #基于最新提交创建标签
$ git tag -d <tagname>         #删除标签
```

合并与衍合

```
$ git merge <branch>           #合并指定分支到当前分支
$ git rebase <branch>          #衍合指定分支到当前分支
```

远程操作

```
$ git remote -v                #查看远程版本库信息
$ git remote show <remote>     #查看指定远程版本库信息
$ git remote add <remote> <url>
                               #添加远程版本库
$ git fetch <remote>           #从远程库获取代码
$ git pull <remote> <branch>   #下载代码及快速合并
$ git push <remote> <branch>   #上传代码及快速合并
$ git push <remote> :<branch/tag-name>
                               #删除远程分支或标签
$ git push --tags              #上传所有标签
```