

GIT 指令速查表

presented by **TOWER** > Version control with Git - made easy



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

显示工作路径下全部已修改的文件

```
$ 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 .
```

指定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
```

修改上次提交

请勿修改已发布的提交记录

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

提交历史

从最新提交开始显示所有的提交记录

```
$ git log
```

显示指定文件的所有修改

```
$ 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

显示所有分支

```
$ git branch -av
```

切换当前分支

```
$ git checkout <branch>
```

创建新分支

基于当前分支

```
$ git branch <new-branch>
```

创建新的可追溯的分支

基于远程分支

```
$ git checkout --track <remote/bran-
```

删除本地分支

```
$ git branch -d <branch>
```

给当前的提交打标签

```
$ git tag <tag-name>
```

更新与发布

列出当前配置的远程端

```
$ git remote -v
```

显示远程端信息

```
$ git remote show <remote>
```

添加新的远程端

```
$ git remote add <shortname> <url>
```

下载远程端的所有改动到本地
不会自动合并到当前

```
$ git fetch <remote>
```

下载远程端的所有改动到本地
自动合并到当前

```
$ git pull <remote> <branch>
```

将本地版本发布到远程端

```
$ git push <remote> <branch>
```

删除远程端分支

```
$ git branch -dr <remote/branch>
```

发布标签

```
$ git push --tags
```

合并与重置

将分支合并到当前

```
$ git merge <branch>
```

将当前版本重置到分支中

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
```

使用配置好的合并工具去解决冲突

```
$ 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
```

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重置到上一次提交的版本
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>
```

版本控制

最佳实践



提交要对应修改

一次提交应该对应一个相关的改动。例如，两个不同的错误应该对应两次不同的提交，使它更容易让其他开发人员明白这个改动。如果这次改动存在问题，也可以方便的回滚到改动之前的状态。通过暂存区标记功能，Git可以轻松打造非常精确的提交。

提交前进行代码测试

不要提交还没有经过完整测试的改动，只有经过测试，并确定无误的改动才能提交。把改动发送给开发团队其他成员前，必须确定所有修改已经完整测试过。这样才算是真正的完成。

使用分支功能

自始至终，Git的核心就是提供一个快速、简单和灵活的分支功能。分支是一个非常优秀的工具，用来帮助开发人员解决在日常团队开发中存在的代码冲突的问题。因此分支功能应该广泛的运用在不同的开发流程中。比如：开发新的功能，修错等等。

经常性的提交修改

经常的提交改动可以更方便为它作注释，从而更容易确保提交的注释和改动的一致性。通过频繁快速的提交来与其他的开发人员共享这些改动，那样就会避免或减少代码整合时带来的冲突。反之，非常庞大的提交将会增大整合时出现冲突的风险。

高质量的提交注释

提交注释的开头需要一个少于50个字的简短说明。在一个空白的分割行之后要写出一个详细的提交细节。比如回答如下的两个问题：

- 出于什么理由需要这个修改？
- 基于当前版本，具体改动了什么？

为了和自动生成的注释保持一致（例如：`git merge`），一定要使用现在时态祈使句（比如使用 `change` 而不要使用 `changed` 和 `changes`）。

合理的工作流程

Git可以支持很多不同流程：长期分支，特性分支，合并或是重置，`git-flow` 等等。选择哪一种流程要取决于如下一些因素：什么项目，什么样的开发，部署模式和（可能是最重要的）开发团队人员的个人习惯。不管怎样，选择什么样的流程都要得到所用开发人员的认同并且一直遵循它。

不要提交不完整的改动

对于一个很大的功能模块来说，完成后再提交并不意味着必须整体完成后才可以，而是要把它正确分割成小的完整的逻辑模块进行经常性的提交。一定不要提交一些不完整的改动，仅仅是因为下班。

同样，如果只是为了得到一个干净的工作区域也不需要立即提交，可以通过Git的`<<Stash>>`命令把这些改动移到另外的分支。

版本控制 不是备份

版本控制系统具有一个很强大的附带功能，那就是服务器端的备份功能。但是不要把VCS当成一个备份系统。一定要注意，只需要提交那些有意义的改动。而不要仅仅作为文件存储系统来使用。（请阅读段落<提交要对应修改>）。

使用帮助文档

显示给定git指令的帮助文档

```
$ git help <command>
```

开放的在线资源

<http://www.git-tower.com/learn>

<http://rogerdudler.github.io/git-guide/>

<http://www.git-scm.org/>