Git Cheatsheet

说明	命令
安装 Git	$sudo apt-get install git
配置用户名和 E-mail	$ git config --global user.name "YourName" $ git config --global user.email "email@example.com"
让 Git 显示颜色	$ git config --global color.ui true
初始化一个 Git 仓库	$ git init

添加文件到 Git 仓库	$ git add <file> #可反复多次使用，添加多个文件；
	$ git add –f <file> #强制添加到 git
	$ git commit -m <“添加说明”>

掌握工作区的状态	$ git status　#告诉你有文件被修改过
查看修改内容	$ git diff
查看提交历史 （回退到历史版本）	$ git log
查看命令历史 （回到未来版本）	$ git reflog
回到上一个版本	$ git reset --hard HEAD^
回到上上个版本	$ git reset --hard HEAD^^
回到 100 个版本	$ git reset --hard HEAD~100
回到任意版本的历史	$ git reset --hard commit_id
丢弃工作区的修改 （让这个文件回到最近一次 git commit 或 git add 时的状态）	$ git checkout -- file
撤销暂存区的修改	$ git reset HEAD file
删除版本库中的文件 （之后 git commit）	$ git rm file
创建 SSH Key （在用户主目录下 /Users/Cedric_M 的.ssh 目录有 id_rsa 和 id_rsa.pub 这两个文件）	$ ssh-keygen -t rsa -C "youremail@example.com"
在 GitHub 中添加 SSHKeys	登陆 GitHub，打开“Account settings”，“SSH Keys”页面，然后，点“Add SSH Key”，填上任意 Title，在 Key 文本框里粘贴 id_rsa.pub 文件的内容，点“Add Key”。
在 GitHub 中添加 Git 仓库	登陆 GitHub，然后，在右上角找到“Create a new repo”按钮，创建一个新的仓库，在 Repository name 填入仓库名称，其他保持默认设置，点击“Create repository”按钮，就成功地创建了一个新的 Git 仓

	库。
关联一个远程库	$ git remote add origin git@server-name:path/repo-name.git
第一次推送 master 分支的所有内容至远程库	$ git push -u origin master
推送最新修改至远程库 （非第一次）	$ git push origin master
创建远程库	登陆 GitHub，创建一个新的仓库，勾选 Initialize this repository with a README，这样 GitHub 会自动为我们创建一个 README.md 文件。
从远程库克隆一个本地库 （默认情况下只能看到 master 分支）	$ git clone git@server-name:path/repo-name.git

创建一个分支并切换至此分支	$ git checkout -b <name>
	或
	$ git branch <name> #创建一个分支
	$ git checkout <name> #切换至此分支

查看当前分支 （当前分支前面会标一个*号）	$ git branch
切换到某分支	$ git checkout <name>
合并指定分支到当前分支 （快速合并，删除分支后会丢掉分支信息）	$ git merge <name>
删除分支	$ git branch -d <name>
查看分支合并图	$ git log --graph --pretty=oneline --abbrev-commit
合并指定分支到当前分支	$ git merge --no-ff -m <“添加说明”> <name>
分支策略	在实际开发中，我们应该按照几个基本原则进行分支管理： 首先，master 分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活； 那在哪干活呢？干活都在 dev 分支上，也就是说，dev 分支是不稳定的，到某个时候，比如 1.0 版本发布时，再把 dev 分支合并到 master 上，在 master 分支发布 1.0 版本； 你和你的小伙伴们每个人都在 dev 分支上干活，每个人都有自己的分支（每添加一个新功能，最好新建一个 feature 分支，在上面开发，完成后，合并，最后，删除该 feature 分支），时不时地往 dev 分支上合并就

	可以了。
储存当前工作现场（可多次使用）	$ git stash
查看所储存的工作现场	$ git stash list

恢复工作现场	$ git stash pop stash@{stash 编号}
	或
	$ git stash apply stash@{stash 编号} #恢复工作现场
	$ git stash drop stash@{stash 编号} #删除 stash 内容

丢弃一个没有被合并过的分支（强行删除分支）	$ git branch -D <name>
查看远程库的信息	$ git remote
查看远程库更详细的信息	$ git remote -v
在本地创建和远程分支对应的分支	$ git checkout -b branch-name origin/branch-name
从本地推送分支	$ git push origin branch-name # 哪些分支需要推送。 master 分支是主分支，因此要时刻与远程同步； dev 分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步； bug 分支只用于在本地修复 bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个 bug； feature 分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。 总之，就是在 Git 中，分支完全可以在本地自己藏着玩，是否推送，视你的心情而定！
创建本地分支和远程分支的链接关系（如果 git pull 提示“no tracking information”时使用）	$ git branch --set-upstream branch-name origin/branch-name
把最新的提交从 origin 抓下来	$ git pull
给当前分支打标签(默认标签打在最新提交的 commit 上)	$ git tag <name>
查看所有标签	$ git tag
对某次提交打标签	$ git tag <name> <commit id>
创建带有说明的标签	$ git tag -a <tag name> -m <"说明文字"> <commit id>
查看标签信息	$ git show <tag name>
用私钥签名一个标签 (需安装 GnuPG)	$ git tag -s <tag name> -m <"说明文字"> <commit id>
删除一个本地标签	$ git tag -d <tagname>

推送一个本地标签	$ git push origin <tagname>
推送全部未推送过的本地标签	$ git push origin --tags
删除一个远程标签(先从本地删除)	$ git push origin :refs/tags/<tagname>
参与一个开源项目	访问它的项目主页，点“Fork”就在自己的账号下克隆了一个仓库，然后，从自己的账号下 clone，一定要从自己的账号下 clone 仓库，这样你才能推送修改。 如果你想修复项目的一个 bug，或者新增一个功能，立刻就可以开始干活，干完后，往自己的仓库推送。 如果你希望项目的官方库能接受你的修改，你就可以在 GitHub 上发起一个 pull request。当然，对方是否接受你的 pull request 就不一定了。
国内的 Git 托管服务	https://gitee.com/
忽略特殊文件	在 Git 工作区的根目录下创建一个特殊的.gitignore 文件（在文本编辑器里“保存”或者“另存为”就可以把文件保存为.gitignore 了），然后把要忽略的文件名填进去，Git 就会自动忽略这些文件。最后一步就是把.gitignore 也提交到 Git。 不需要从头写.gitignore 文件，GitHub 已经为我们准备了各种配置文件，只需要组合一下就可以使用了。所有配置文件可以直接在线浏览： https://github.com/github/gitignore
忽略文件的原则	1、忽略操作系统自动生成的文件，比如缩略图等； 2、忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如 Java 编译产生的.class 文件； 3、忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。 4、.gitignore 文件本身要放到版本库里，并且可以对.gitignore 做版本管理！
检查. gitignore 的正确性	$ git check-ignore -v <file>

配置别名	$ git config --global alias.<别名> <命令名>
	(加上--global 是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用)
	常见配置：
	$ git config --global alias.co checkout
	$ git config --global alias.ci commit

	$ git config --global alias.br branch
	$ git config --global alias.unstage 'reset HEAD'
	$ git config --global alias.last 'log -1'
	$ 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"
	(每个仓库的 Git 配置文件都放在.git/config 文件中，而当前用户的 Git 配置文件放在用户主目录下的一个隐藏文件.gitconfig 中，别名就在[alias]后面，要删除别名，直接把对应的行删掉即可。)

搭建 Git 服务器
通过 git clone 命令克隆远程仓库了，在各自的电脑上运行：
$ git clone git@server:/srv/sample.git
（如果团队很小，把每个人的公钥收集起来放到服务器的 /home/git/.ssh/authorized_keys 文件里就是可行的。如果团队有几百号人，就没法这么玩了，这时，可以用 Gitosis 来管理公钥。）
	$ sudo apt-get install git #安装 git
	$ sudo adduser git #创建一个 git 用户，用来运行 git 服务
	收集所有需要登录的用户的公钥，就是他们自己的 id_rsa.pub 文件，把所有公钥导入到 /home/git/.ssh/authorized_keys 文件里，一行一个。
	$ sudo git init --bare sample.git #初始化 Git 仓库（假定是/srv/sample.git）
	$ sudo chown -R git:git sample.git #把 owner 改为 git
	# 禁用 Shell 登录
	编辑/etc/passwd 文件完成。找到类似下面的一行：
	git:x:1001:1001:,,,:/home/git:/bin/bash
	改为：
	git:x:1001:1001:,,,:/home/git:/usr/bin/git-shell

Git 官网	http://git-scm.com

关于廖雪峰老湿的 Git 教程做的笔记，以后方便各位学习吧~ ~ ~

https://www.liaoxuefeng.com/wiki/001373951630592960dd18361248578c67b8067c8c017b000

本人 GitHub：https://github.com/Barzarrhey/git_cheatsheet

2018.5.13 22：45　By Barzarrhey