# Git常用命令

## 初始化本地仓库（远程仓库克隆）

适用于远程仓库已创建好仓库，且本地没有创建本地仓库，将远程仓库直接克隆到本地：

git clone [url]

例：git clone <http://172.16.0.101:3000/jmax/jmq_app_android.git>

git checkout -b branchname <commitId>

例：git checkout -b master e4e7330eebf291324c60a31cc2ead4ea60c1b70e

## 初始化本地仓库（完全初始化重建模式）

适用于远程仓库已创建好仓库，且本地没有创建本地仓库，先创建本地仓库，之后再将远程仓库进行关联

### 初始化Git仓库

git init

### 关联远程仓库

git remote add origin <http://172.16.0.101:3000/jmax/jmq_app_android.git>

### 同步远程仓库

master为Git远程仓库的主分支，仓库可能有多个分支，比如开发dev，测试test，如果需要拉哪个分支，则写对应的名称，后面的命令同理。

git pull origin master

### 添加本地文件

添加本地文件，注意最后是一个点，意思是将你本地修改了的文件添加到暂存区：

git add .

### 提交日志信息

git commit -m “首次提交”

### 推送至远程仓库

git push origin master

### 强推至远程仓库(禁用)

git push origin HEAD --force

## 修改本地仓库关联的远程仓库地址

### 查看仓库关联的地址

git remote -v

### 切换地址

git remote set-url origin [url]

## 添加Tag标签

Tag的意义在于对不同时期的代码进行版本区分及进行标记，方便后期维护及查找代码。

### 添加本地标签

git tag 1.0.0

### 提交标签

git push --tag

### 删除本地标签

git tag -d 1.0.0

### 删除远程标签

git push origin :refs/tags/1.0.0

## 切换分支

### 查看所有分支

git branch -a

### 捡出分支

checkout远程的dev分支，在本地起名为dev分支，并切换到本地的dev分支：

git checkout -b [dev](https://www.baidu.com/s?wd=dev&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmWm1nW9hn1I-m1R3nWDd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWmYnjTkPHfkn1n4nHckPWfd) origin/[dev](https://www.baidu.com/s?wd=dev&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmWm1nW9hn1I-m1R3nWDd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWmYnjTkPHfkn1n4nHckPWfd)

### 切回分支

切换回dev分支，并开始开发：

git checkout dev

## 子分支同步Master

git checkout master

git pull

git checkout dev

git merge master

// 处理冲突 git add & git commit

git push dev

## 子分支合并到Master（现有分支）

git checkout dev

git add .

git commit -m "..."

git checkout master

git merge dev

// 处理冲突 git add & git commit

git push

## 合并分支(创建临时分支)

### 创建临时分支

将master分支作为temp分支：

git fetch origin master:temp

### 切换分支

git branch temp

### 将temp分支和本地分支合并

git merge temp

### 删除分支（禁用）

git branch -d temp

## 撤销本地提交

### 查看本地提交信息

git show

### 撤销某次提交(仅取消Commit信息，不还原代码，软撤销)

git reset --soft HEAD^

或 git reset --soft HEAD~1 //1表示撤销几次，如果是2则会连续撤销2次commit

### 撤销某次提交(不仅取消Commit信息，而且还原代码，硬撤销)

git reset --hard HEAD^

### 回到到某个提交

git reset --hard commit\_id // commit\_id为对应commit的sha码

### 修改提交信息

git commit --amend

## 支持大于50/100M文件上传

git config http.postBuffer 524288000

git config -l

## 取消文件跟踪

### 取消某个文件的跟踪(不删除本地文件)

git rm --cached file\_name.txt

### 取消某个文件的跟踪(删除本地文件，禁用)

git rm --f file\_name.txt

### 取消所有文件的跟踪(不删除本地文件)

git rm -r --cached .

### 取消所有文件的跟踪(删除本地文件，禁用)

git rm -r –f .

## Git老仓库迁移至新仓库

1、使用git remote set-url origin [url]切换远程仓库；

2、查看本地关联的地址：git remote -v，如果仓库地址没有切换，直接编辑.git/config中的url换成新仓库地址；

3、推本地代码送新的远程仓库：git push -u origin master；

4、如果出现错误，忽略及同步远程仓库历史关联：git pull origin master --allow-unrelated-histories

5、再次：git push -u origin master

## SVN仓库迁移至Git仓库

1. 提交本地代码并推送至SVN仓库；
2. 在SVN本地仓库同级目录(即.svn隐藏文件夹同目录)下，新建GitProject文件夹，执行以下命令(svn仓库地址需要更换成对应SVN远程仓库的地址)：

svn log 远程SVN仓库地址 -q | awk -F '|' '/^r/ {sub("^ ", "", $2); sub(" $", "", $2); print $2"="$2" <"$2"@iconcox.com>"}' | sort -u > ./users.txt

1. 编辑user.txt，把=后面的姓名改成对应的名称(即=号后面的名称需要换成git账号名称)，比如lizhijian=lizhijian<[lizhijian@jimilab.com](mailto:lizhijian@jimilab.com)>，修改成lizhijian=zhijian.li<[lizhijian@jimilab.com](mailto:lizhijian@jimilab.com)>
2. 执行拷贝svn代码命令：

git svn clone 远程SVN仓库地址 --no-metadata --authors-file=users.txt GitProject

1. 新建或拷贝忽略文件配置文件.gitignore在GitProjec下(若是新建文件，则需要写如[忽略文件配置](#_12、.gitignore忽略文件配置)中的配置内容)；
2. 进入GitProject文件夹，执行推送命令：

git push -u origin master

1. 若提示推送失败，执行忽略Git仓库历史记录命令，之后再执行第5步：

git pull origin master --allow-unrelated-histories

## .gitignore忽略文件配置

.gitignore是Git对整个仓库文件需要忽略上传的一个配置文件，默认每个仓库都应该具备。配置可以为如下内容（已包含Xcode、AS、RN的需要忽略的文件）：

# ---> Objective-C

# Xcode

#

# gitignore contributors: remember to update Global/Xcode.gitignore, Objective-C.gitignore & Swift.gitignore

## Build generated

build/

DerivedData

## Various settings

\*.pbxuser

!default.pbxuser

\*.mode1v3

!default.mode1v3

\*.mode2v3

!default.mode2v3

\*.perspectivev3

!default.perspectivev3

xcuserdata

## Other

\*.xccheckout

\*.moved-aside

\*.xcuserstate

\*.xcscmblueprint

\*/.DS\_Store

.DS\_Store

## Obj-C/Swift specific

\*.hmap

\*.ipa

\*.dSYM.zip

\*.dSYM

.build/

# CocoaPods

#

# We recommend against adding the Pods directory to your .gitignore. However

# you should judge for yourself, the pros and cons are mentioned at:

# http://guides.cocoapods.org/using/using-cocoapods.html#should-i-check-the-pods-directory-into-source-control

#

#Pods/

Pods/

Podfile.lock

\*.xcworkspace

# Carthage

#

# Add this line if you want to avoid checking in source code from Carthage dependencies.

# Carthage/Checkouts

Carthage/Build

# fastlane

fastlane/report.xml

fastlane/Preview.html

fastlane/screenshots/\*\*/\*.png

fastlane/test\_output

# ---> Swift

# Xcode

#

# gitignore contributors: remember to update Global/Xcode.gitignore, Objective-C.gitignore & Swift.gitignore

## Build generated

build/

DerivedData

## Various settings

\*.pbxuser

!default.pbxuser

\*.mode1v3

!default.mode1v3

\*.mode2v3

!default.mode2v3

\*.perspectivev3

!default.perspectivev3

xcuserdata

## Other

\*.xccheckout

\*.moved-aside

\*.xcuserstate

\*.xcscmblueprint

## Obj-C/Swift specific

\*.hmap

\*.ipa

# CocoaPods

#

# We recommend against adding the Pods directory to your .gitignore. However

# you should judge for yourself, the pros and cons are mentioned at:

# http://guides.cocoapods.org/using/using-cocoapods.html#should-i-ignore-the-pods-directory-in-source-control

#

# Pods/

# Carthage

#

# Add this line if you want to avoid checking in source code from Carthage dependencies.

# Carthage/Checkouts

Carthage/Build

## RN

node\_modules

yarn.lock

bundles

yarn-error.log

\*.jsbundle