**初次运行Git前配置**

git config 可选参数--global

--system

--list

参数--global 设置当前用户的git配置，对应配置文件 ~/.gitconfig

--system 设置系统的git配置，对应配置文件 /etc/.gitconfig

--list 显示配置文件，包括用户与系统的

例子

git config --global user.name “xxx”

git config --system user.email xxx@xxx.com

git config --system core.editor “vim”

**现有目录中进行Git仓库的初始化**

git init 作用：在当前目录下生成.git目录(仓库)

**克隆仓库**

git clone [URL] URL有两种协议可选http、ssh

http协议 https://github.com/xx/xxx.git

ssh协议 git@github.com:xx/xxx.git

例子

git clone https://github.com/xx/xxx.git

git clone git@github.com:xx/xxx.git

http协议可以直接下载，但一般用这种协议只有只读权限(体现在如果我们要推送git push就需要填写账户和密码)

ssh 协议需要远程仓库这边具有我们的公钥才可以，用这种协议下载意味着我们对仓库具有读写权限(体现在如果我们要推送git push就不需要账户和密码，可以直接上传)

**检查对当前文件状态** | 工作区目录 | 暂存区 | .git(仓库)

git status 可选参数-s /--short

参数为空，则显示普通的文件状态

git status

参数-s/--short 显示简易文件状态

git status -s

git status –short

该命令显示的符号意思如下：?? 表示文件未被跟踪

A 表示文件已被放暂存区

M 表示文件已被修改

MM表示修改的文件已被放暂存区，同时又在工作区进行了修改

1.change to be commit 已暂存，未提交

git status 三种状态： 2.change not to staged for commit 已修改，未暂存

3.untracked filed 未跟踪

**跟踪文件，暂存已修改的文件**

git add 作用1：跟踪一个新文件

作用2：将被修改的跟踪文件添加到暂存区

例子

git add . 表示添加所有未跟踪/已跟踪且发生修改的文件

git add \* 效果同上

git add [文件] 将(已修改)文件添加到暂存区/将未跟踪的文

件加入到跟踪队列

**提交到仓库**

git commit 可选参数 -a

-m

参数为空，则启动默认文本编辑器来输入提交说明

git commit

参数-a 效果等效于git add . && git commit

git commit -a

参数-m 提交到仓库并提交简要说明

git commit -m “本次提交的简要说明”

**查看已暂存和未暂存的修改**

git diff 可选参数 --staged/--cached

参数为空，查看工作区的修改与暂存区版本的区别

参数--staged/--cached 查看已添加暂存区的修改与提交仓库的区别

**忽略特定文件**

1.所有空行或以“#”开头的行都认为是注释

2.匹配模式以“/”开头防止递归

编写.gitignore文件 3.匹配模式已“/”结尾指定目录

4.要忽略指定模式以外的文件，可以在前面加“!”

例子

.gitignore

#忽略所有.a后缀的文件

\*.a

#不忽略lib.a文件

!lib.a

#忽略devel文件夹下的内容，但对其下级子目录不忽略

/devel

#忽略build文件夹下的内容，包括其下级子目录

build/

#忽略catkin文件夹中所有lib的文件夹，如catkin/build/lib、catkin/devel/lib

catkin/\*\*/lib

.gitignore的作用就是我们在git add时，自动忽略这些文件，文件的内容修改和新增，建议git add之前先设置好，不然如果不小心加入进去就只能用git rm <不想跟踪的文件>来删除

**将跟踪文件从暂存区中移除**

git rm 可选参数-f

--cached

参数为空，将跟踪文件从暂存区中删除(不再跟踪该文件)，同时也将其从工作区中删除

git rm <不想跟踪的文件>

参数-f 跟踪文件已经发生修改时，强行从暂存区中删除，同时也将其从工作区中删除

git rm -f <不想跟踪的文件>

参数--cached 仅将跟踪文件从暂存区中删除(仅删除跟踪文件)，保留工作区中的文件

git rm --cached <不想跟踪的文件>

**移动跟踪文件/重命名跟踪文件** 1. mv <文件a><文件b>

git mv <文件a> <文件b> 等效于 2. git rm <文件a>

3. git add <文件b>

**查看提交历史**

git log 可选参数 -p

-n

--pretty

--oneline

--graph

--decorate

参数为空，显示提交历史

git log

参数-p 显示提交同时显示提交的内容差异

git log -p

参数-n n表示数字，显示最近n次提交

git log -5 显示最近5次的提交

参数--pretty 格式化输出，如--pretty=oneline

git log --pretty=oneline

参数--oneline 仅显示一行

git log --oneline

参数--graph 图形化显示提交历史

git log --graph

参数--decorate 显示提交历史同时显示分支情况

git log --decorate

**取消暂存文件或撤销工作区文件的修改**

使用git status命令，根据提示进行操作

**查看远程仓库**

git remote 可选参数-v

add

show

rename

remove

参数为空，显示目前已有的远程仓库

git remote

参数-v 显示已有的远程仓库同时显示该远程仓库的URL

git remote -v

参数add 添加远程仓库，将远程仓库的URL与<远程仓库名>关联起来

git remote add <远程仓库名> <URL>

注意：<URL>有http和ssh两种协议。使用http关联的远程仓库在每次git push <远程仓库名>时需要输入账户和密码；使用ssh则不需要，但需要远程仓库那边有我们的公钥

参数show 显示远程仓库的详细信息

git remote show <远程仓库名>

参数rename 对远程仓库改名

git remote rename <旧远程仓库名> <新远程仓库名>

参数remove 删除远程仓库

git remote remove <远程仓库名>

**从远程仓库中拉取数据**

git fetch <远程仓库名>

作用：将远程仓库(服务器)上的数据拉到本地远程仓库进行数据同步

**分支合并更新**

git merge <本地分支>/<远程分支名> 反正就是你想合并的

作用：对分支进行合并，特别地，当我们想合并远程仓库的分支时，先使用

git fetch对本地的远程仓库进行更新，因为本地的远程仓库不会主动同步更新

服务器的远程仓库

**推送到远程仓库**

git push 可选参数 --delete

参数为空，将我们想要分享的某本地分支推送到远程仓库的某个分支上

git push <远程仓库名> <本地分支>:<远程分支>

注意地方有3个：<远程仓库名>是有git remote add设置的，使用http在推送时需要账户和密码，想要http免密码看《progit》，ssh则不需要；另外想要推送的分支不要求是当前分支，也就是可以在当前分支(分支a)推送其他分支(分支b)到远程仓库的某分支；如果推送的远程分支不存在，则自动创建该分支，并把内容上传到该分支上。

参数--delete 删除远程仓库的分支(从本地删除服务器上的分支)远程删除分支

git push <远程仓库名> --delete <远程分支名>

**拉取远程分支并进行分支合并**

git pull 注意该命令等于git fetch && git merge，使用该命令前，确保当前分支已经与远程仓库的某分支进行了关联，将分支设置为跟踪分支的方法git branch与git checkout

**查看、创建、设置分支**

git branch 可选参数-v

-vv

可选参数-d

--merged

--no-merged

-a

-r

-u

--set-upstream-to=

--track

参数为空，查看本地分支

git branch

参数为空，创建新分支

git branch <新分支名>

参数-v 显示分支详细信息

git branch -v

参数-vv 显示更详细信息

git branch -vv

参数-d 删除分支

git branch -d <想要删掉的分支>

参数--merged 显示有哪些分支与当前分支进行过合并

参数 --no-merged 反之

git branch --merged

参数-a 显示所有分支，包括远程与本地

git branch -a

参数-r 仅显示远程分支

git branch -r

参数-u 设置远程跟踪分支

参数--set-upstream-to= 同上

git branch <已有分支> -u <远程分支>

git branch <已有分支> --set-upstream-to= <远程分支>

参数--track 创建新分支并将其设为跟踪分支

git branch <新建分支> --track <远程分支>

**切换分支(标签，历史)**

git checkout 可选参数-b

--track

参数为空，从当前分支切换到另一个分支

git checkout <想要切换分支名>

参数为空，从当前提交历史切换到特定标签

git checkout <想要切换标签名>

参数为空，从当前提交历史切换到特定的提交历史

git checkout <想要切换提交历史>

参数-b 创建新分支并切换到该分支

git checkout -b <新分支>

效果等效于git branch <新分支>&&git checkout <新分支>

参数--track 创建新分支，将其设置为跟踪分支，并切换到该分支

git checkout -b <新分支> --track <远程分支>

效果等效于git checkout -b <新分支>&&git branch <新分支> -u <远程分支>

创建与远程分支同名的新分支，将其设置为跟踪分支，并切换到该分支

git checkout -b --track <远程分支>

例子

git checkout -b --track origin/dev

等效于 git checkout -b dev –track origin/dev

**创建标签**

git tag <标签名> 可选参数-l

-a

参数为空，创建轻量标签，进去默认编辑器进行编辑

git tag

参数-l 列出含有特定信息的标签

git tag -l “特定信息”

例子

git tag -l “first commit”

列出所有含有first commit信息的标签

参数-a 创建附注标签

git tag -a <标签名>