Git 基本使用

安装

- Ubuntu

  `sudo apt install git` 直接安装即可

- Windows

  从 git 官网下载

下载安装完成后在终端或 git bash 中执行如下命令进行用户设置

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

创建仓库

```
git init
```

将当前目录初始化为一个 git 仓库

```
git add file.txt
```

将工作区中的一个文件添加到暂存区。也可以执行 `git add *` 添加所有文件（Ubuntu不包括隐藏文件和忽略的文件）。

```
git commit -m "description"
```

将暂存区中的所有文件提交为一个版本。`-m` 后的信息为对当前修改/版本的描述，方便日后查阅。

```
git status
```

查询当前工作区状况

版本操作

历史查询

```
git log
```

查询当前分支的历史版本。也可以执行 `git reflog` 查询所有分支的所有版本，包括已删除的提交记录和回退的版本。

```
git diff
```

查询修改。可以指定到特定文件和特定分支。

版本回退

```
git reset ID
```

回退至 ID 版本（ID 可以不写完整，一般写六七位即可），也可以用 HEAD 指针替代 ID。若想强制回退，需加上参数 `--hard

```
git checkout -- file.txt
```

将一个文件在工作区中的全部修改撤销。

```
git reset HEAD file.txt
```

将一个文件在暂存区中的全部修改撤销。

```
git rm file.txt
```

将一个文件从仓库中移除（并非删除文件）。

暂存

在你当前的工作没有完成（不适合提交）但又出现了更紧急的任务时（比如修复某个 bug），可以使用暂存功能保存当前的进度，去其他分支完成工作再回来继续。

```
git stash
```

把当前修改暂存后将工作区清空。

```
git stash list
```

查看当前所有的暂存

```
git stash apply stash@{0}
```

恢复某次暂存

```
git stash drop stash@{0}
```

丢弃某次暂存

远程仓库与协作

SSH 密钥

- Ubuntu

  ```
  ssh-keygen
  ```

- Windows

```
ssh-keygen -t rsa -C "youremail@example.com"
```

```
cat ~/.ssh/id_rsa.pub
```

获取 ssh key 公钥，然后将其保存到远程仓库的个人账号里即可。

创建远程库

```
git remote add origin git@remote.com:usrname/reponame
```

为本地仓库关联一个远程仓库，远程仓库的地址会在代码托管网站中给出。

```
git remote -v
```

查看远程库信息

```
git push origin master
```

将本地仓库的master分支推送到远程。如果是本地的第一次推送，需要将命令改为 `git push -u origin master`。

克隆远程库

```
git clone git@remote.com:usrname/reponame
```

将一个远程库克隆到本地。克隆时可以选择 http 协议或 ssh 协议，后者速度更快但需要事先添加 ssh key，前者每次克隆时都需要输入账号和密码，一般会选择使用 ssh 协议克隆。

```
git pull
```

从远程库拉取最新内容。在多人协作时可以及时更新其他人的贡献，以方便自己的开发匹配当前的最新进度。

删除远程库

```
git remote rm origin
```

解除与远程库的关联。若需真正删除一个远程库，还需在代码托管网站中进行操作。

分支管理

分支操作

```
git branch
```

查看本地所有分支。若想查看远程的分支，需加上参数 `-a`。

```
git checkout branchname
```

切换到一个不同的分分支。若想从当前分支创建并切换到一个新的分支，需在 `checkout` 后加上参数 `-b`。注意：当前工作区未清空前无法执行切换分支的操作。

```
git merge branchname
```

将指定分支的内容合并到当前分支。

```
git branch -d branchname
```

删除一个本地分支。若想强制删除一个分支（未合并的分支），可以将参数 `-d` 改为 `-D`。

冲突处理

当两个分支中有冲突内容的时候，git 无法执行自动合并，在 merge 之后会保留冲突并显示出来，在文件中可以看到尖括号指示的冲突内容。在这时，需要人为手动处理冲突，然后再提交修改。这时两个分支中的内容都会被修改合并。

分支管理

master 分支作为主分支，他的版本一般比较稳定。绝大部分仓库都会新建一个 dev 分支用于开发和测试，成功之后再合并到 master。但如果是一个多人的项目团队，一般都会以个人或开发特性新建独立的开发分支，从开发分支再合并到 dev 统一。

标签

标签可以作为一些重要版本的简称，可以替代提交 ID，方便回退或查找。

```
git tag v0.0
```

给当前版本打上标签。

```
git tag -a v0.0 -m "description" ID
```

给特定版本提交打上标签，并添加描述信息。

```
git tag
```

查看所有标签

```
git tag -d v0.0
```

删除一个标签。

```
git push origin v0.0
```

将一个标签推送到远程。若想一次推送所有标签，只需执行 `git push origin --tags`。

仓库设置

仓库描述

一般的仓库都会有一个 README.md 文件，用于描述该仓库的功能接口、运行环境、实现原理等基本信息。在远程仓库中，README.md 的信息会被直接展示在仓库的首页，以方便查阅。

忽略的文件

在 .gitignore 文件中，可以自定义仓库中忽略的文件，一般包括 build/，*.png，*.avi 等编译文件或大型文件等不适合提交到版本仓库的文件。

模块

在 .gitmodule 文件中可以自定义仓库包含的子模块。

```
git submodule repoaddress repopath
```

将某远程的仓库指定为本仓库的子模块。

```
git submodule update --init --recursive
```

在克隆下来的仓库中初始化子模块。

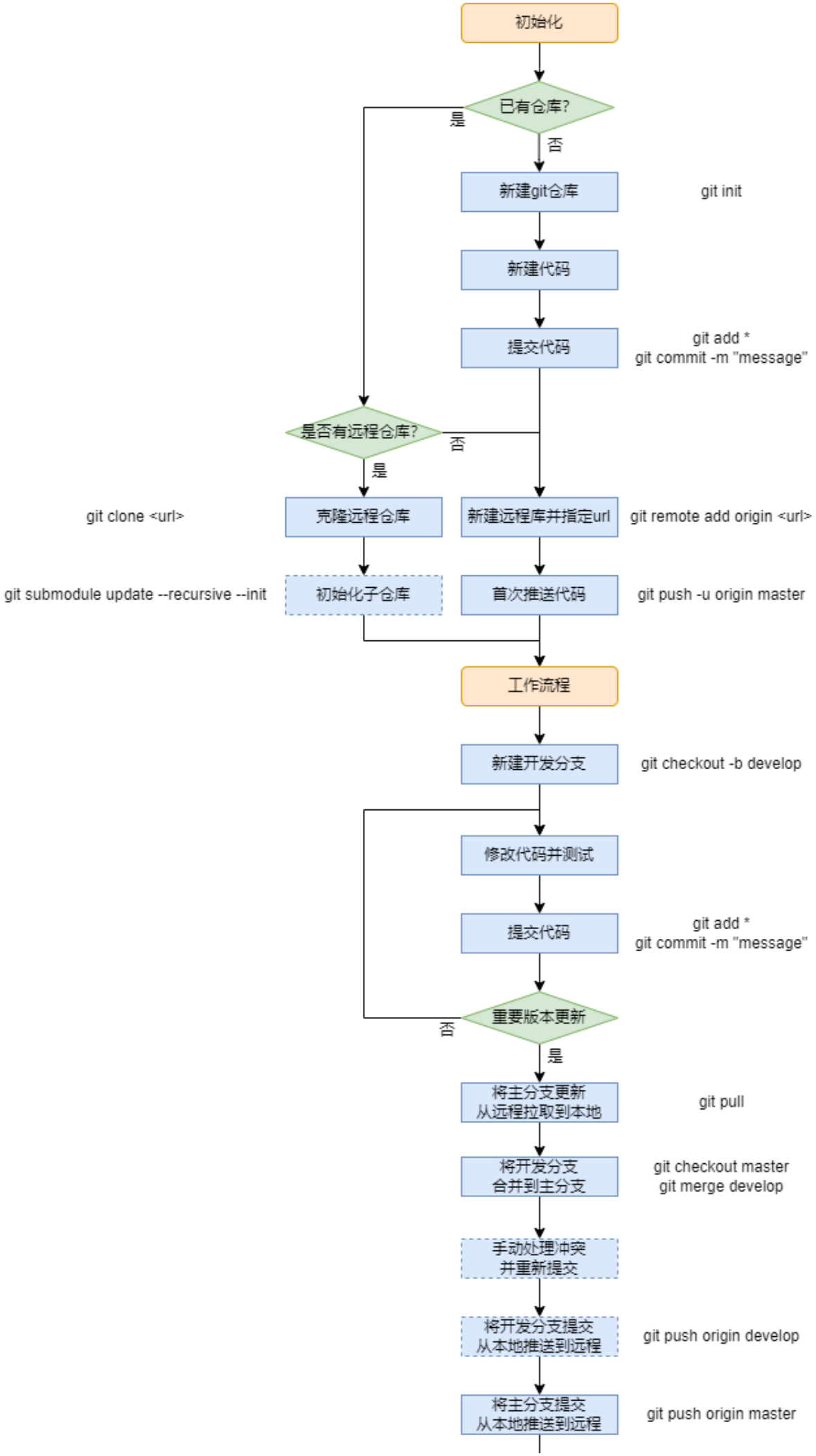
子模块的相关操作较为复杂，在此不做赘述。

其他

```
git blame file.txt
```

找锅。

工作流程


初始化
已有仓库?
是
否
新建git仓库
git init
新建代码
提交代码
git add *
git commit -m "message"
是否有远程仓库?
否
是
git clone <url>
克隆远程仓库
新建远程库并指定url
git remote add origin <url>
git submodule update --recursive --init
初始化子仓库
首次推送代码
git push -u origin master
工作流程
新建开发分支
git checkout -b develop
修改代码并测试
提交代码
git add *
git commit -m "message"
重要版本更新
否
是
将主分支更新
从远程拉取到本地
git pull
将开发分支
合并到主分支
git checkout master
git merge develop
手动处理冲突
并重新提交
将开发分支提交
从本地推送到远程
git push origin develop
将主分支提交
从本地推送到远程
git push origin master

参考教程