**1.安装Git**

Linux:sudo apt-get install git

Window:到Git官网下载安装：https://git-scm.com/downloads

**2.配置全局用户Name和E-mail**

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

**3.初始化仓库**

git init

**4.添加文件到Git仓库**

git add <file>

提示：可反复多次使用，添加多个文件；

**5.提交添加的文件到Git仓库**

git commit

然后会弹出一个Vim编辑器输入本次提交的内容；

或者

git commit -m "提交说明"

**6.查看仓库当前的状态**

git status

**7.比较当前文件的修改**

$ git diff <file>

**8.查看历史提交记录**

git log

或者加上参数查看就比较清晰了

$ git log --pretty=oneline

**9.回退版本**

$ git reset --hard HEAD^

说明：在Git中，用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，以此类推，如果需要回退几十个版本，写几十个^容易数不过来，所以可以写，例如回退30个版本为：HEAD~30。

如果你回退完版本又后悔了，想回来，一般情况下是回不来的，但是如果你可以找到你之前的commit id的话，也是可以的，使用如下即可：

$ git reset --hard + commit id

提示：commit id不需要写全，Git会自动查找；

补充说明：Git中，commit id是一个使用SHA1计算出来的一个非常大的数字，用十六进制表示，你提交时看到的一大串类似3628164...882e1e0的就是commit id（版本号）；

在Git中，版本回退速度非常快，因为Git在内部有个指向当前版本的HEAD指针，当你回退版本的时候，Git仅仅是把HEAD从指向回退的版本，然后顺便刷新工作区文件；

**10.查看操作的历史命令记录**

$ git reflog

结果会将你之前的操作的commit id和具体的操作类型及相关的信息打印出来，这个命令还有一个作用就是，当你过了几天，你想回退之前的某次提交，但是你不知道commit id了，通过这个你可查找出commit id,就可以轻松回退了，用一句话总结：穿越未来，回到过去，so easy！

**11.diff文件**

git diff HEAD -- <file>

说明：查看工作区和版本库里面最新版本文件的区别，也可以不加HEAD参数；

**12.丢弃工作区的修改**

$ git checkout -- <file>

说明：适用于工作区修改没有add的文件

**13.丢弃暂存区的文件**

$ git reset HEAD <file>

说明：适用于暂存区已经add的文件，注意执行完此命令，他会将暂存区的修改放回到工作区中，如果要想工作区的修改也丢弃，就执行第12条命令即可；

**14.删除文件**

$ rm <file>

然后提交即可；

如果不小心删错了，如果还没有提交的话使用下面命令即可恢复删除，注意的是它只能恢复最近版本提交的修改，你工作区的修改是不能被恢复的！

$ git checkout -- <file>

**15.创建SSH key**

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

一般本地Git仓库和远程Git仓库之间的传输是通过SSH加密的，所以我们可以将其生成的公钥添加到Git服务端的设置中即可，这样Git就可以知道是你提交的了；

**16.与远程仓库协作**

$ git remote add origin git@github.com:xinpengfei520/IM.git

删除本地库与远程库的关联：

$ git remote rm origin

作用：有时候我们需要关联其他远程库，需要先删除旧的关联，再添加新的关联，因为如果你已经关联过了就不能在关联了，不过想关联多个远程库也是可以的，前提是你的本地库没有关联任何远程库，操作如下：

先关联Github远程库：

$ git remote add github git@github.com:xinpengfei520/IM.git

接着关联码云远程库：

$ git remote add gitee git@gitee.com:xinpengfei521/IM.git

现在，我们用git remote -v查看远程库的关联信息，如果看到两组关联信息就说明关联成功了；

ok,现在我们的本地库可以和多个远程库协作了

如果要推送到GitHub，使用命令：

$ git push github master

如果要推送到码云，使用命令：

$ git push gitee master

**17.推送到远程仓库**

$ git push -u origin master

注意：第一次提交需要加一个参数-u,以后不需要

**18.克隆一个远程库**

$ git clone git@github.com:xinpengfei520/IM.git

**19.Git分支管理**

创建一个分支branch1

$ git branch branch1

切换到branch1分支：

$ git checkout branch1

创建并切换到branch1分支：

$ git checkout -b branch1

查看分支：

$ git branch

提示：显示的结果中，其中有一个分支前有个\*号，表示的是当前所在的分支；

合并branch1分支到master：

$ git merge branch1

删除分支：

$ git branch -d branch1

**20.查看提交的历史记录**

$ git log

命令可以看到分支合并图

git log --graph

**21.合并分支**

禁用Fast forward模式合并分支

$ git merge --no-ff -m "merge" branch1

说明：默认Git合并分支时使用的是Fast forward模式，这种模式合并，删除分支后，会丢掉分支信息，所以我们需要强制禁用此模式来合并；

补充内容：实际开发中分支管理的策略

master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面提交；

我们可以新开一个dev分支，也就是说dev分支是不稳定的，到版本发布时，再把dev分支合并到master上，在master分支发布新版本；

你和你的协作者平时都在dev分支上提交，每个人都有自己的分支，时不时地往dev分支上合并就可以了；

**22.保存工作现场**

$ git stash

作用：当你需要去修改其他内容时，这时候你的工作还没有做完，先临时保存起来，等干完其他事之后，再回来回复现场，再继续干活；为什么？因为暂存区是公用的，如果不通过stash命令隐藏，会带到其它分支去；

查看已经保存的工作现场列表：

$ git stash list

恢复工作现场(恢复并从stash list删除)：

$ git stash pop

或者：

git stash apply

恢复工作现场，但stash内容并不删除，如果你需要删除执行如下命令：

$ git stash drop

恢复指定的stash:

$ git stash apply stash@{0}

说明：其中stash@{0}为git stash list中的一种编号

**23.丢弃一个没有被合并过的分支**

强行删除即可：

$ git branch -D <name>

作用：实际开发中，添加一个新feature，最好新建一个分支，如果要丢弃这个没有被合并过的分支，可以通过上面的命令强行删除；

**24.查看远程库的信息**

$ git remote

显示更详细的信息：

$ git remote -v

25.推送分支

推送master到远程库

$ git push origin master

推送branch1到远程库

$ git push origin branch1

26.创建本地分支

$ git checkout -b branch1 origin/branch1

说明：如果远程库中有分支，clone之后默认只有master分支的，所以需要执行如上命令来创建本地分支才能与远程的分支关联起来；

27.指定本地branch1分支与远程origin/branch1分支的链接

$ git branch --set-upstream branch1 origin/branch1

作用：如果你本地新建的branch1分支，远程库中也有一个branch1分支(别人创建的)，而刚好你也没有提交过到这个分支，即没有关联过，会报一个no tracking information信息，通过上面命令关联即可；

28.创建标签

$ git tag <name>

例如：git tag v1.0

查看所有标签：

$ git tag

对历史提交打tag

先使用$ git log --pretty=oneline --abbrev-commit命令找到历史提交的commit id

例如对commit id 为123456的提交打一个tag:

$ git tag v0.9 123456

查看标签信息：

$ git show <tagname>

eg:git show v1.0

创建带有说明的标签，用-a指定标签名，-m指定说明文字，123456为commit id：

$ git tag -a v1.0 -m "V1.0 released" 123456

用私钥签名一个标签：

$ git tag -s v2.0 -m "signed V2.0 released" 345678

说明：签名采用PGP签名，因此，必须先要安装gpg（GnuPG），如果没有找到gpg，或者没有gpg密钥对，就会报错，具体请参考GnuPG帮助文档配置Key；

作用：用PGP签名的标签是不可伪造的，因为可以验证PGP签名；

删除标签：

$ git tag -d <tagname>

删除远程库中的标签：

比如要删除远程库中的 V1.0 标签，分两步：

[1] 先删除本地标签：$ git tag -d V1.0

[2] 再推送删除即可：$ git push origin :refs/tags/V1.0

推送标签到远程库：

$ git push origin <tagname>

推送所有标签到远程库：

$ git push origin --tags

29.自定义Git设置

Git显示颜色，会让命令输出看起来更清晰、醒目：

$ git config --global color.ui true

设置命令别名：

$ git config --global alias.st status

说明：--global表示全局，即设置完之后全局生效，st表示别名，status表示原始名

好了，现在敲git st就相当于是git status命令了，是不是方便？

当然还有其他命令可以简写，这里举几个：很多人都用co表示checkout，ci表示commit，br表示branch...  
根据自己的喜好可以设置即可，个人觉得不是很推荐使用别名的方式；

推荐一个比较丧心病狂的别名设置：

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"

效果自己去体会...

其他说明：配置的时候加上--global是针对当前用户起作用的，如果不加只对当前的仓库起作用；每个仓库的Git配置文件都放在 .git/config 文件中，我们可以打开对其中的配置作修改，可以删除设置的别名；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，我们也可以对其进行配置和修改。

30.忽略文件规则

原则：

忽略系统自动生成的文件等；

忽略编译生成的中间文件、可执行文件等，比如Java编译产生的.class文件，自动生成的文件就没必要提交；

忽略你自己的带有敏感信息的配置文件，个人相关配置文件；

忽略与自己相关开发环境相关的配置文件；

...

使用：在Git工作区的根目录下创建一个特殊的 .gitignore 文件，然后把要忽略的文件名或者相关规则填进去，Git就会自动忽略这些文件，不知道怎么写的可参考：<https://github.com/github/gitignore>,这里提供了一些忽略的规则，可供参考；

如果你想添加一个被 .gitignore 忽略的文件到Git中，但发现是添加不了的，所以我们可以使用强制添加$ git add -f <file>

或者我们可以检查及修改 .gitignore 文件的忽略规则：

$ git check-ignore -v <file>

Git会告诉我们具体的 .gitignore 文件中的第几行规则忽略了该文件，这样我们就知道应该修改哪个规则了；

如何忽略已经提交到远程库中的文件？  
如果你已经将一些文件提交到远程库中了，然后你想忽略掉此文件，然后在 .gitignore 文件中添加忽略，然而你会发现并没有生效，因为Git添加忽略时只有对没有跟踪的文件才生效，也就是说你没有add过和提交过的文件才生效，按如下命令：

比如说：我们要忽略.idea目录，先删除已经提交到本地库的文件目录

git rm --cached .idea

格式：git rm --cached + 路径

如果提示：fatal: not removing '.idea' recursively without -r

加个参数 -r 即可强制删除

$ git rm -r --cached .idea

然后，执行git status会提示你已经删除.idea目录了，然后执行commit再push就可以了，此时的.idea目录是没有被跟踪的，将.idea目录添加到 .gitignore 文件中就可以忽略了。