**Git初始化**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

配置用户名和邮件地址

git config --global user.name "xxx"

git config --global user.email xxx@gmail.com

在git命令输出中开启颜色显示

git config --global color.ui true

新建工作目录，执行git init创建版本库

git init

有文件新建后执行

git add xx

如果是文件修改

git add

提交文件

git commit -m "xxxx"

其他命令

git log --stat

git log --pretty=oneling 精简输出来显示日志

git status -s 精简格式的状态输出

git diff /git diff HEAD / git diff --cached

Git 使用.gitignore文件来忽略不愿意提交的目录或者文件

然后就直接写要忽略的文件或者文件夹，例如忽略到cache目录

/cache

这样即可

git pull = git fetch + git merge

git pull实际包含两个动作： 获取远程版本库的最新提交，以及将获取到的远程版本库提交与本地提交进行合并。

git push 推送到远程版本库

git stash 保存进度 保存当前的工作进度。会分别对暂存区和工作区的状态进行保存。

git stash apply 恢复进度

git stash list 显示进度列表。此命令显然暗示了git stash 可以多次保存工作进度，并用在恢复时候选择。

本地删除如果要反映到暂存区中应该用git rm命令，对不想删除的文件执行git checkout --<file> 可以让文件在工作区重现。

git add -u 快速标志删除 将本地有改动（包括修改和删除）的文件标记到暂存区。

git --git-dir --work-tree

使用git的时候，通常的用法是 cd 到 repository目录下进行相关的操作，比如说有这么一个repository， D:\pygit2（通过 git clone <https://github.com/libgit2/pygit2.git>）

通常的做法是 cd D:\pygit2,然后再执行git的相关指令，比如 git status, git log 等等，这种情况下，git命令默认的是 .gitr和working-tree在同一个目录下.

其实还有一种做法，在执行git命令的时候显式指定git-dir(.git目录)和working-tree，这就是本节要讲述的 --git-dir 和 --work-tree 参数

如果想将pygit2 repository中的某些commit copy到D:\pygit2以外的目录中，该怎么办？当然通过git clone 再创建一个repository可以做到，但是否是最简洁的方式？而且git clone存在其他一些不必要的开销，在某些情况下（比如含有大量的commit，branch）或许不是很合适。此时可以考虑使用git checkout 或者 git reset命令来实现，只需显式指定working-tree即可。具体实现方法如下：

假定想将D:\pygit2中的master copy到D:\bak中

方法一：

1.1 git --git-dir=D:\\pygit2\\.git --work-tree=D:\\bak status 此时可以看到很多提示信息，意思是D:\\bak和D:\pygit2 repository相比很多文件缺失

1.2 git --git-dir=D:\\pygit2\\.git --work-tree=D:\\bak checkout -b tmp -f master 使用-f强制checkout，并创建一个tmp的branch

1.3 git --git-dir=D:\\pygit2\\.git --work-tree=D:\\bak branch -d tmp

删除tmp branch

1.4 此时可以看到D:\\bak中的内容已经被更新为D:\\pygit2\\.git master branch的内容

方法二、

2.1 git --git-dir=D:\\pygit2\\.git --work-tree=D:\\bak reset --quiet --hard

2.2 此时可以看到D:\\bak中的内容已经被更新为D:\\pygit2\\.git master branch的内容,不过这种方法存在一定的弊端，那就是index中的内容都已经丢失，想要恢复　　　　　　的话存在一定的难度，所以在使用reset之前还是慎重一些比较好。

需要注意的是， git-dir传入的一定是git的repository，也就是.git文件夹。

**Git 冲突解决**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**A:**

git add -u

git commit -m "xxxx"

git push

**B:**

git add -u

git commit -m "xxx"

git pull

提示冲突：

查看暂存区中有关冲突文件的多个不同版本的记录：

git ls-files 命令查看

编号为1 的暂存区用于保存冲突文件修改之前的副本

编号为2 的暂存区用于保存当前冲突文件在当前分支中修改的副本

编号为3 的暂存区用于保存当前冲突文件在合并中修改的副本

手动修改后， git add -u

git commit -m "xxxxx"

通过git log --oneline --graph-3 可以查看

**图形界面解决冲突：**

git mergetool

**Git 分支**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**Git 子模块**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**创建子模块：**

git submodule add <repo(需要引用的外部仓库)> <path(在本仓库下子模块的路径)>

git commit -m "xxx"

git push

**克隆带有子模块的版本库：**

**git clone 完后，需要在进行**

git submodule init

git submodule update