Git

## 彩色配置命令

git config --global color.ui.auto

## 查看提交日誌

git log

退出提交日誌，按q鍵

## 兩個版本對比

git diff 老版本號 新版本號

## Git本地初始化目錄，然後上傳到GitHub上的新倉庫

一、進入需要git管理的文件夾

二、右擊找到Git Bash Here，彈出git命令框

三、初始化git

$git init

四、將文檔添加到本地暫存區

$git add --all

~~~~~~~~~~~~

git add相關常用命令。

如果要將單個文檔加入到暫存區，可採用如下類似代碼：

$ git add readme.txt

如果要將多個文檔加入到暫存區，可以採用如下類似代碼：

$ git add readme.txt ant.txt

文檔與文檔之間用空格分隔；也可以使用通配符方式批量提交文檔：

$ git add \*.html

上面代碼可以將所有的html文檔提交到暫存區。還有兩個殺傷力更強的方式：

$ git add all

$ git add .

兩者都可以將工作區中所有未跟蹤或者修改的文檔添加到暫存區。

但是兩者還是有一些區別的，下面分別做一下介紹：

一.版本導致的差別： 1.x版本：

（1）.git add all可以提交未跟蹤、修改和刪除文檔。

（2）.git add .可以提交未跟蹤和修改文檔，但是不處理刪除文檔。

2.x版本： 兩者功能在提交類型方面是相同的。

二.所在目錄不同導致的差異：

（1）.git add all無論在哪個目錄執行都會提交相應文檔。

（2）.git add .只能夠提交目前的目錄或者它後代目錄下相應文檔。

~~~~~~~~~~~~

五、git commit -m用於提交存儲於暫存區的文件

$git commit -m "首次提交"

六、與GitHub上的分支建立連結

$git remote add origin [https://github.com/\*\*\*\*\*/shop.git](https://github.com/*****/shop.git)

七、將剛剛提交的文件推送到GitHub上

$git push -u origin master -f

~~~~~~~~~~~~

謹慎使用git push -f，它將本地歷史覆蓋到遠端倉庫。

* 一般形式 git push <遠程主機名> <本地分支名>:<遠程分支名>

git push origin master:refs/for/master，將本地master分支推送到遠程主機origin（在第六步裡添加了origin）上對應的master分支。

其中refs/for表示代碼提交到服務器之後要經過code review後才能merge，而refs/heads不需要。

* git push origin master

遠程分支被省略，表示將本地分支推送到與之存在追蹤關係的遠程分支。通常兩者同名。如果該遠程分支不存在，則會被新建。

* git push origin : refs/for/master

git push origin : master

如果省略本地分支名，則表示刪除指定的遠程分支，因為這等同於推送一個空的本地分支到遠程分支，等同於git push origin --delete master。

* git push origin

當前分支與遠程分支之間存在追蹤關係，怎本地分支和遠程分支都可以省略

* git push

如果當前分支只有一個追蹤分支，那麼主機名都可以省去

* git push -u origin master

如果當前分支與多個主機存在追蹤關係，可以使用-u指定一個默認主機，這樣後面就可以直接使用git push

八 查看遠程倉庫地址：git remote -v

九 修改遠程倉庫地址：

git remote set-url origin git@10.1.22.30:LauncherGroup/Launcher.git

## Git退回版本

git log 查看commit提交記錄，退出查看log，按q

git reset --hard 52602b0b04b57b33310fa256707518b126d2648c 回退到指定commit id的版本

回退到某一个文件之后，又需要返回到最近更新的某个版本：

（1）git reflog 来查看每一次命令的记录

（2）git reset --hard 580361e 回退到最近更新的版本（比目前所在的版本新）

## Git創建分支

git checkout -b iss53 創建並切換到分支iss53

## git clone、git pull和git fetch的用法及区别

### 1.git clone

git clone顾名思义就是将其他仓库克隆到本地，包括被clone仓库的版本变化。举个例子，你当前目录比方说是在e:/course/中，此时若想下载远程仓库，本地无需git init,直接git clone url（url是你远程仓库的地址，直接复制就可以了）。执行git clone等待clone结束，e:/course/目录下自动会有一个.git的隐藏文件夹（如果看不见，请尝试设置隐藏文件夹可见），因为是clone来的，所以.git文件夹里存放着与远程仓库一模一样的版本库记录。clone操作是一个从无到有的克隆操作，再次强调不需要git init初始化。

git clone的用法：

$ git clone <版本库的url>

例如克隆TensorFlow：

$ git clone https://github.com/tensorflow/tensorflow.git

或者使用SSH协议：

$ git clone git@github.com:tensorflow/tensorflow.git

这样就会在本地生成一个目录，该目录与远程仓库同名。

However，如果本地目录不想与远程仓库同名怎么办？？也有办法，将目录名作为git clone命令的第二个参数:

$ git clone <版本库的网址> <本地目录名>

### 2.git pull

git pull是拉取远程分支更新到本地仓库的操作。比如远程仓库里的学习资料有了新内容，需要把新内容下载下来的时候，就可以使用git pull命令。事实上，git pull是相当于从远程仓库获取最新版本，然后再与本地分支merge（合并）。

　　即：git pull = git fetch + git merge

注：git fetch不会进行合并，执行后需要手动执行git merge合并，而git pull拉取远程分之后直接与本地分支进行合并。更准确地说，git pull是使用给定的参数运行git fetch，并调用git merge将检索到的分支头合并到当前分支中。

git pull的用法：

$ git pull <远程主机名> <远程分支名>:<本地分支名>

举例：将远程主机origin的master分支拉取过来，与本地的branchtest分支合并。

$ git pull origin master:branchtest

如果将冒号和后面的branchtest去掉，则表示将远程origin仓库的master分支拉取下来与本地当前分支合并。

以上的git pull操作如果用git fetch来表示：

$ git fetch origin master:brantest

$ git merge brantest

相比起来，git fetch更安全也更符合实际要求，因为可以在merge前，我们可以查看更新情况，根据实际情况再决定是否合并。

### 3.git fetch 更新远程代码到本地仓库

理解 fetch 的关键, 是理解 FETCH\_HEAD，FETCH\_HEAD指的是: 某个branch在服务器上的最新状态’。这个列表保存在 .Git/FETCH\_HEAD 文件中, 其中每一行对应于远程服务器的一个分支。

当前分支指向的FETCH\_HEAD, 就是这个文件第一行对应的那个分支.

一般来说, 存在两种情况:

如果没有显式的指定远程分支, 则远程分支的master将作为默认的FETCH\_HEAD

如果指定了远程分支, 就将这个远程分支作为FETCH\_HEAD

git fetch更新本地仓库的两种用法：

# 方法一

$ git fetch origin master #从远程的origin仓库的master分支下载代码到本地的origin maste

$ git log -p master.. origin/master #比较本地的仓库和远程参考的区别

$ git merge origin/master #把远程下载下来的代码合并到本地仓库，远程的和本地的合并

# 方法二

$ git fetch origin master:temp #从远程的origin仓库的master分支下载到本地并新建一个分支temp

$ git diff temp #比较master分支和temp分支的不同

$ git merge temp #合并temp分支到master分支

$ git branch -d temp #删除temp