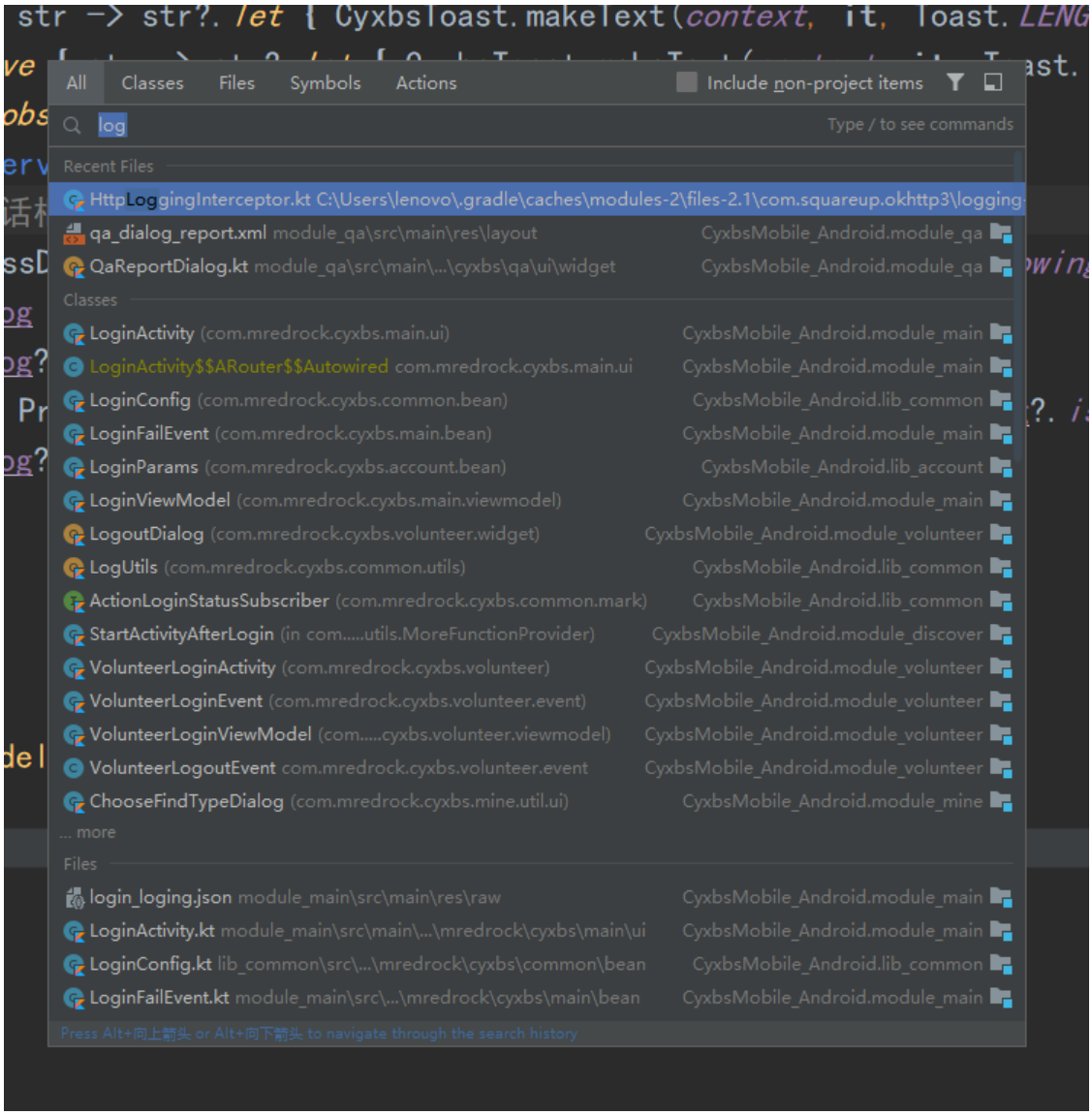
红岩网校春季第一次课

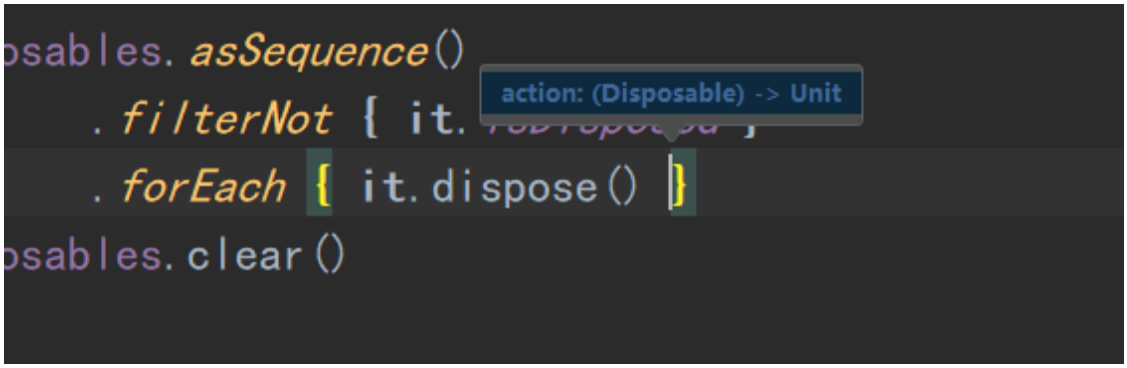
主讲：张哲

一、AS使用技巧

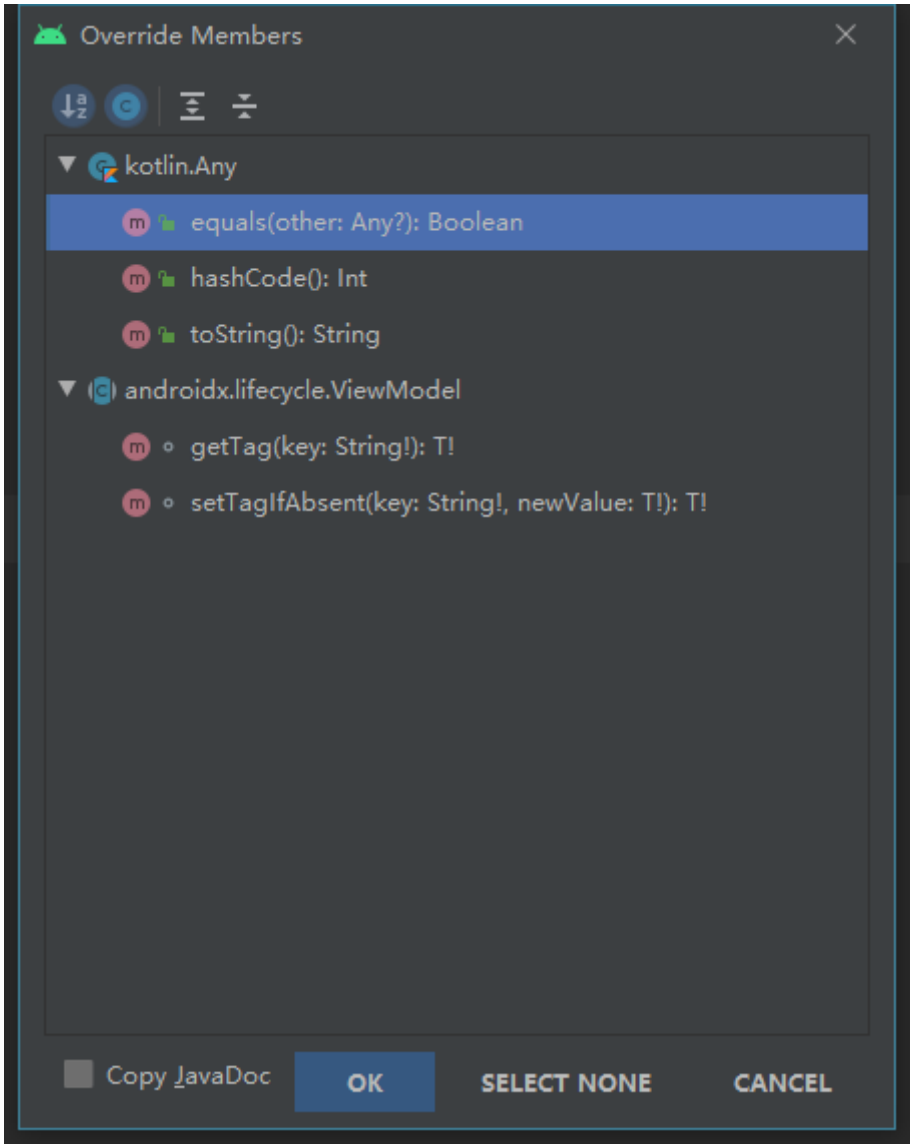
1. Shift+Shift 全局搜索/输入命令/设置



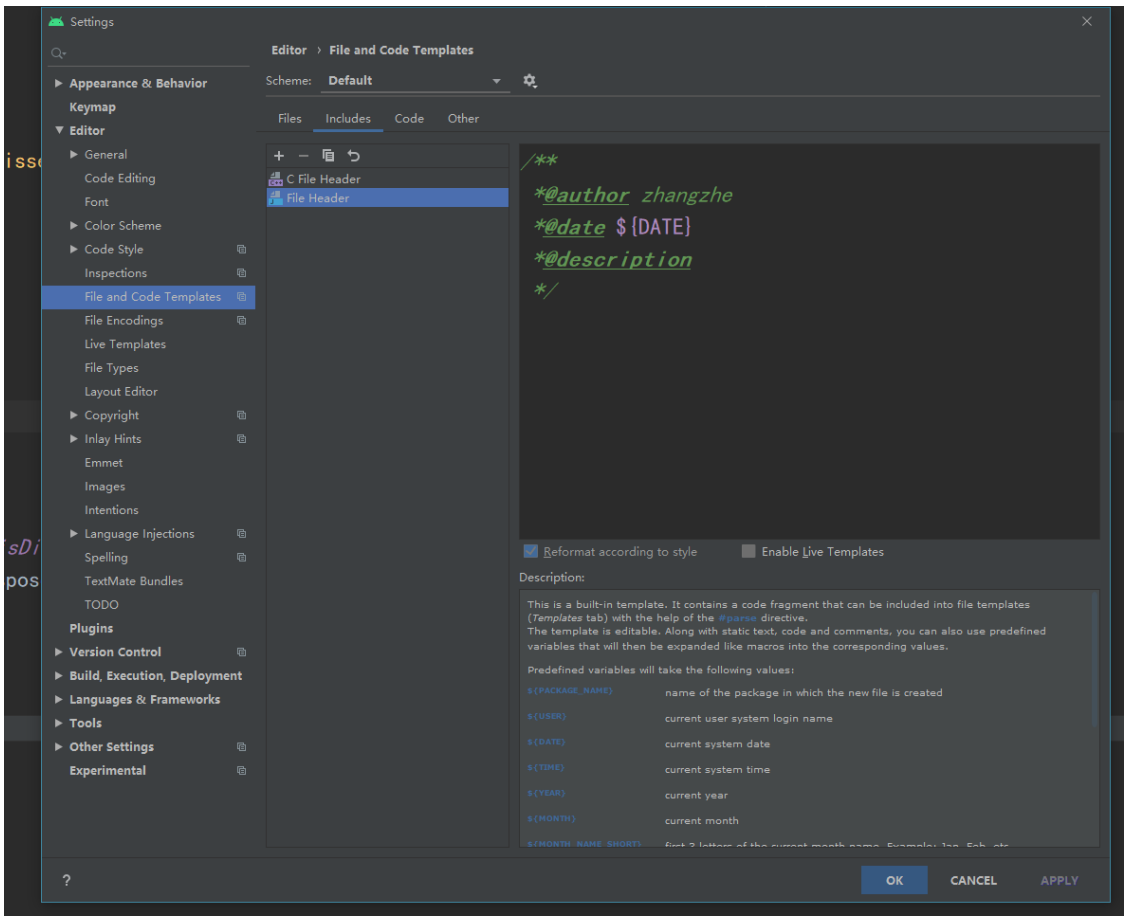
2. Ctrl+P 显示函数参数信息



3. Ctrl+O 显示所有类方法，选择可重写

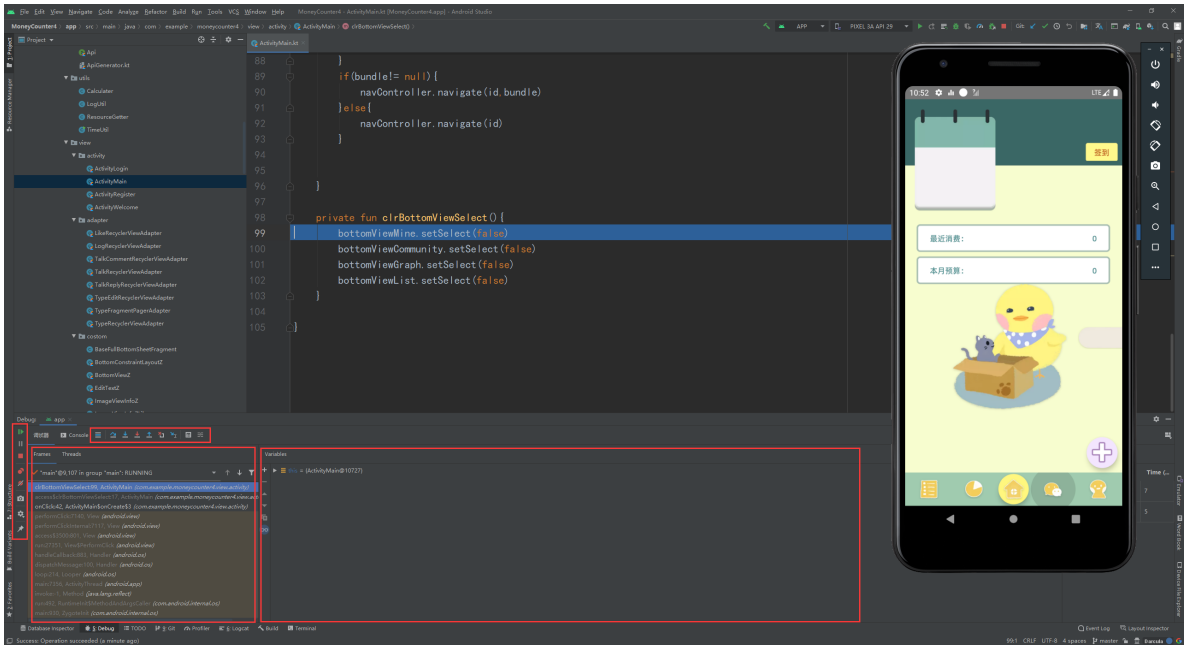


4. 编辑属于自己的头注释



5. Ctrl+C 快速复制一行代码
6. Ctrl+X 快速剪切一行代码，可用于快速删除
7. Ctrl+←→ 快速移动光标，可以跨越单词地移动光标
8. Alt+Insert 生成构造器/Getter/Setter/构造函数等
9. Ctrl+Z 撤销一次操作
10. Ctrl+Shit+Z 前进一次操作
11. Shift+F6 refactor重命名
12. Ctrl+Alt+L 格式化代码
13. Ctrl+W 扩展选择范围，选择单词继而语句继而函数
14. Ctrl+Shift+W 缩小选择选择范围，选择函数继而语句继而单词
15. Ctrl+A 全选
16. Ctrl+S 保存
17. Shift+R 全局替换内容，可以用于搜索，手动重命名
18. Ctrl+Alt+B 查找方法引用和声明（跟Ctrl+点击函数一样效果）
19. Ctrl+U 找到父类方法（常用）
20. Ctrl+Alt+Shift+T 打开refactor菜单
21. Ctrl+Alt+F 局部变量快速变为全局变量

二、使用调试功能（debug）



单击右上角的虫子一样的按钮，进入debug模式。

强烈建议配合模拟器置顶使用。

功能说明

1. 转到当前执行行

2. 向下执行，会进入自定义方法，不进入官方库方法体

3. 向下执行一行，不进入方法

4. 强制进入方法内部

5. 觉得方法没有问题，跳出本方法回到方法调用处

6. 撤销上一次函数调用（后悔药）

7. 运行到光标处

8. 输入java/kotlin表达式，获取其值，不管是否是private

9. 继续执行

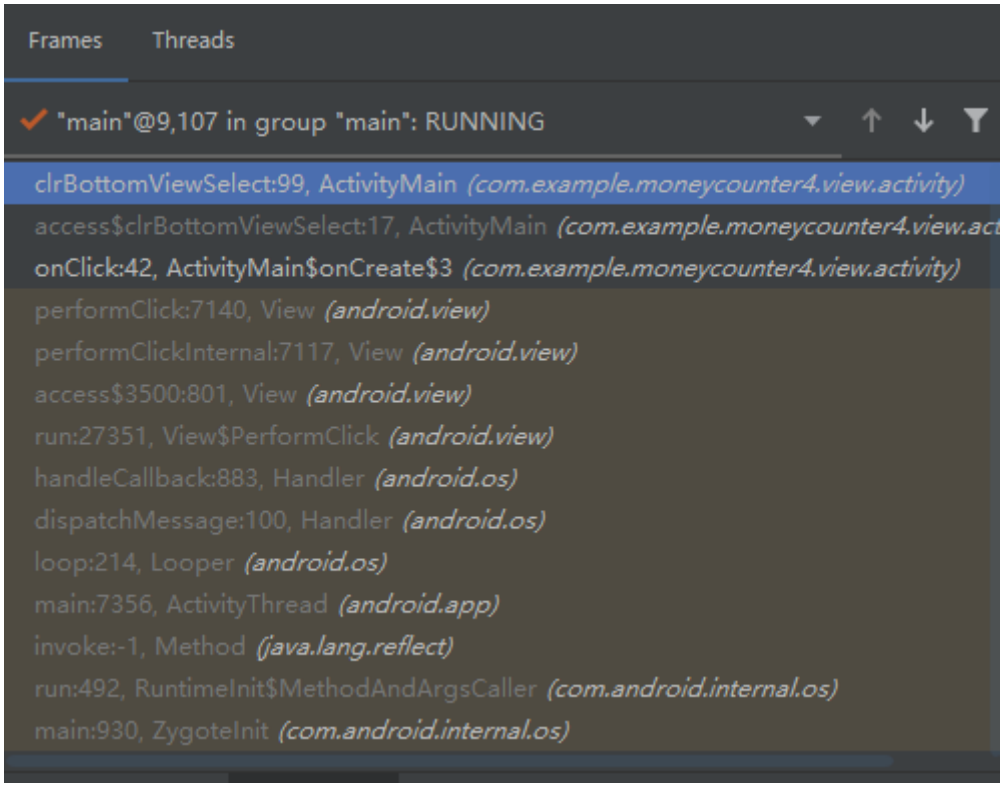
10. 暂停执行

11. 终止调试

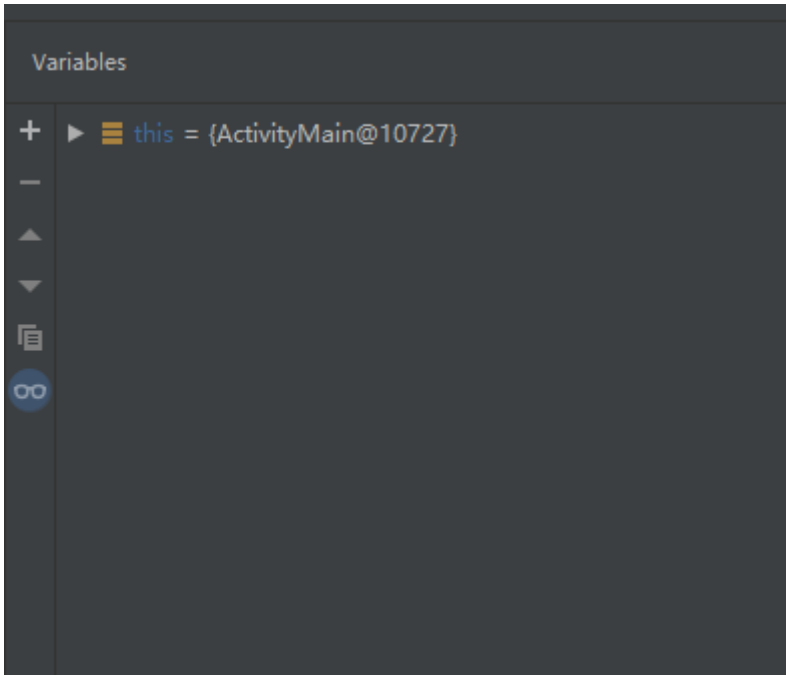
12. 显示所有断点列表/配置断点属性

13. 禁用/启用所有断点

14. 获得当前运行线程的快照



15. 方法调用栈

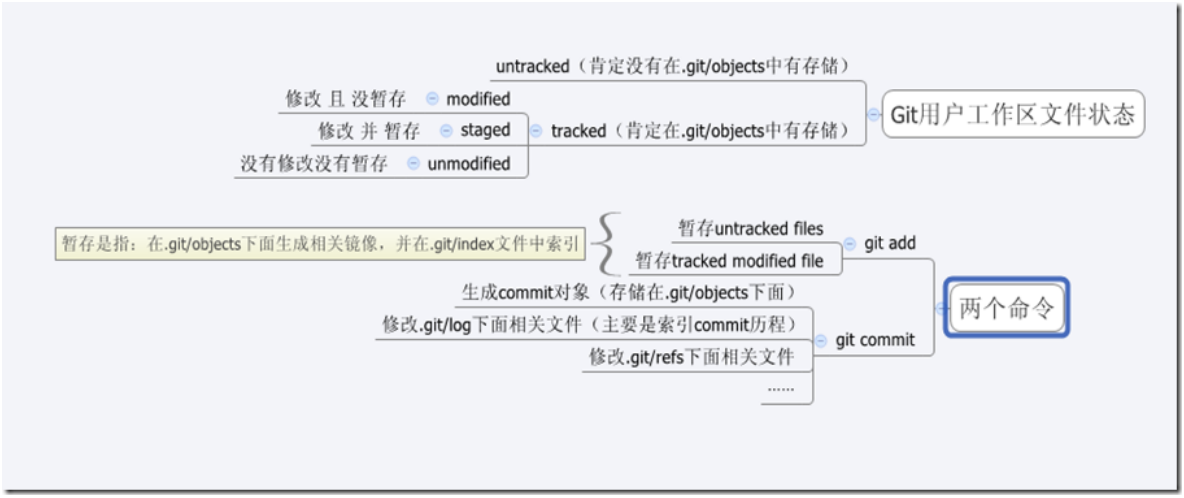


16. 变量列表，可以看到当前执行的变量的值，点击左边的加号可以增加跟踪的变量，不管是否是private

3、Git的基本使用

git的基本命令

```
git init    git status    git add    git commit    git push    git reset
git rm    git diff    git branch    git checkout    git log
git reflog    git merge    git rebase （-i）    git fetch    git stash
git revert    git cherry-pick
```



上图片在理解了git的工作后再看会有所收获。

工作区就是我们能看到的文件夹，暂存区又称索引（index）区。

暂存区改动只由add命令影响。相当于搭建了一个桥梁，从工作区到暂存区。

add的细节：

1、当一个文件被add了，他的修改自始至终都会被git追踪（tracked），包括删除。

2、当一个文件没有被add，git中的各种操作将对他毫无影响（untracked）。

3、当你准备提交一部分文件作为新的commit时，一般会在commit前进行add操作对吧，这里的add只是标注“哪些文件的修改需要添加到这次commit中”。而不是“将哪些文件添加到这次commit”。看似细微，我们用例子来理解。

（为了强调上面这部分，实际上是因为，网上很多教程说add命令就是把“文件”添加到暂存区，这导致我们以为没有add的文件就不在本次提交的范围内，以后回退也不受影响。其实没add的文件只是没提交修改罢了，git提交这些文件的时候只是忽略了你本次的修改。）

> 如果还不懂可以看看例子，如果懂了可以跳过。
>
> 当有两个文件：
>
> > a.txt 内容：123
> >
> > b.txt 内容：456
>
> 先add . 全部产生一次commit后（这里只是保证a和b都在git管辖之中），再修改a.txt的内容为123123，b.txt修改内容为456456，然后只add a.txt后commit，这次节点看上去只提交了a，如果在以后的版本回退到这个节点上，是不是b的内容保持原样不变呢，a变为了123是在意料之中的。其实不是这样，因为只add a.txt说明只提交了a的修改，b的修改没有提交（所以提交的b是上次commit中b的状态），所以以后版本回退到这个节点，b的内容会变为456，而不是保持原样“456456”。
>
> 有了这些你就可以简单地理解为：每次工作完，要提交哪些文件就add哪些文件，剩下的一部分没有add的文件，其实也提交了，只是提交为“跟上次一样的文件”（不管你在工作区是否修改了；相当于“不提交本次修改”）。

命令用法

> 在linux下参数有两种用法：1、双横线后面加参数的全称，2、单横线加参数的首字母

git status 显示暂存区状态

git status -s 以精简方式显示暂存区文件状态

```
lenovo@LAPTOP-PHNUOLVV MINGW64 ~/Desktop/g (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   a.txt

no changes added to commit (use "git add" and/or "git commit -a")
lenovo@LAPTOP-PHNUOLVV MINGW64 ~/Desktop/g (master)
$ git status -s
 M a.txt
```

git add . 将所有“没被忽略”的文件的修改添加到暂存区（有些文件被.gitignore过滤了）

git add * 将所有文件的修改（不管.gitignore）都添加到暂存区

rm [文件名] 是linux命令，仅仅只是删除工作区文件的命令（此时还没有add）

git rm [文件名] 会删除暂存区或者分支上的文件，工作区的文件也会被删除（相当于rm后再git add）

git rm --cached 是从git取消对本文件的追踪（不会删除工作区文件）。当你不想对本文件进行版本控制了，就用本命令

git commit -m "[提交描述]" 直接提交信息

git commit 会弹出一个vim编辑器（linux的一个文本编辑器），按i进入insert插入模式，输入一些commit文本后按Esc进入一般模式（vim编辑器的一种模式），按“:”键输入命令“wq”按回车会自动保存后退出，如果输入“q”表示不保存退出。这个git commit命令实际上和git commit -m 命令是等效的。

git commit -a -m "[提交描述]"相当于执行git commit -m前先执行git add .

git branch [名称]新建一个分支

git branch -r 列出远程分支

git branch 列出本地分支，并且在当前分支前面加*号

git branch -d [名称] 删除本地分支

git branch -d -r [名称] 删除远程分支

git branch -m [旧名称] [新名称] 分支重命名

git branch -vv 查看本地仓库和远程仓库的对应关系

```
lenovo@LAPTOP-PHNU0LVV MINGW64 ~/Desktop/g (master)
$ git branch -a
* master
  remotes/abc/master

lenovo@LAPTOP-PHNU0LVV MINGW64 ~/Desktop/g (master)
$ git branch -vv
* master 4085629 [abc/master] ii4ajiasasdfasdfk ihjkhkh
```

git checkout -b [名称]新建一个分支并切换到该分支上

git checkout [名称]切换分支

git checkout -q [名称]无提示切换分支

git checkout -f [名称]强制切换（如果工作区有修改，则丢弃本次修改，强制切换到分支）

git checkout [某次版本的hash码]这样是没有意义的（一般HEAD指针指向的是分支名），仅仅可以通过这个方法去看“那次版本的文件的内容”，但是不推荐在这里commit（这算是在游离的分支）。

git log 显示所有分支的提交历史记录（把所有分支的提交汇总了），如果一个屏幕显示不下，可以按回车浏览下部，按q退出浏览

git log -p 会显示每次提交的差异细节（常用，方便）

```
commit 7651016953ca2e167fbfef7f897727718dd41ffd
Author: 737138090 <737138090@qq.com>
Date:   Fri Aug 28 10:09:09 2020 +0800

    [fix]所有图片里添加上下间距

diff --git a/module_map/src/main/res/layout/map_recycle_item_all_picture.xml b/module_map/src/
index fcd6e07..26a79a2 100644
--- a/module_map/src/main/res/layout/map_recycle_item_all_picture.xml
+++ b/module_map/src/main/res/layout/map_recycle_item_all_picture.xml
@@ -4,6 +4,7 @@
     android:layout_height="wrap_content"
     android:layout_marginLeft="10dp"
     android:layout_marginRight="10dp"
+    android:layout_marginBottom="12dp"
     android:gravity="center">
```

git log --stat 显示每次提交的统计信息（行数变化，修改过哪些文件）

git log --oneline --graph [你要比较的n个分支] 这几个参数组合有妙用：oneline表示单行显示，graph表示显示左方的图形化。这样就能看到详细的提交信息了。

```
lenovo@LAPTOP-PHNUOLVV MINGW64 /c/AAmyprog/Android/ypao/CQUPT_RunTogether (p/zh
ngzhe)
$ git log --oneline --graph master ktwork
*   9f4dcea (ktwork) Merge pull request #7 from RedrockMobile/p/zhangtao
|\
| * 46074ff (origin/p/zhangtao) [fix]黑夜模式切换，图标显示错乱
* | 1310d3b Merge pull request #6 from RedrockMobile/p/zhangtao
|\|
| * f9f9fda [fix]修复黑夜模式切换viewpager不显示
|/
*   353ddbc Merge branch 'ktwork' of github.com:RedrockMobile/CQUPT_RunTogether
into ktwork
|\
| *   52f6b3e Merge pull request #5 from RedrockMobile/p/zhangtao
| |\
| | * 05eaf8c [optimize]将下载地图放在了MainActivity中，优化运动记录年份
* | | 88cff0e [optimize] 修改一些注释
|/ /
* | b461244 Merge pull request #3 from RedrockMobile/p/zhangtao
|\|
| * 47874dc [fix]修复安装器打包重复启动
|/
* a876b79 [fix] 修复运动记录图表的显示
*   6071a86 Merge branch 'ktwork' of github.com:RedrockMobile/CQUPT_RunTogether
into ktwork
|\
| * a161eeb [fix] 修改运动图表超出5km的情况
* | ef79086 [feature] 友盟多渠道打包
|/
*   ffbbe0f Merge branch 'ktwork' into p/yuanbing
|\
| * 1635868 [fix] 修改图标颜色不对的问题
* | 0bea76c [fix] 修复若干BUG
|/
* 5a0bf2d [fix] [optimize] 修复一些BUG、删除未使用的import
*   24e2d5f Merge branch 'p/yuanbing' into ktwork
|\
| * 792bc3f [feature] 添加后台权限设置详情页
* | 69fea1b [fix] 修改快速点击导致的问题
* | 581e3f6 [fix] 修改部分机型FloatingActionButton会被遮挡的问题
* | 2fa442f [fix] 1.修改了信息后，数据能够同步；2.完善了自启动机型的跳转；3.修改
一些显示错误
* | f037745 [fix] 1.先给 HttpCommonInterceptor 的 String? 变量加上 ?:""       2
修改因为时间参数计算错误导致当日图表无法显示的问题
* | a2685ef [fix] 修改Dialog显示时，状态栏因为设置透明范围过长发黑的问题
|/
```

git diff [文件名] 显示工作区和暂存区文件的差异，如果没有add过（暂存区没有它），则对比的是最近一次版本和工作区的差异。

git diff [分支名] [文件名] 对比工作区的文件和分支的文件的差别

git diff --cached [文件名] 对比暂存区和git仓库的区别（就是暂存区和最近一次commit的差异）

git diff [哈希码] [文件名] 对比工作区和git仓库中某次提交的区别

git diff --cached [哈希码] [文件名] 对比暂存区和git仓库中某次提交的区别

git diff [哈希码] [哈希码] 对比任意两次提交（可以来自不同分支呀，因为每次提交都有唯一的哈希码）

上述diff命令可以不指定文件名，则对全部文件操作。

上述哈希吗可以换成相对引用（“HEAD^^”或者“HEAD~2”这样的）

git reset --hard [哈希码] 强制回退到某次commit（工作区会恢复为和那次commit一样的状态，暂存区会清空）

git reset --soft [哈希吗] 软回退到某次commit（工作区和暂存区内容保留不动，实际上是分支指针的移动）

reset和checkout的区别：git checkout [哈希吗]只是将本地文件恢复到那次commit的状态，这是HEAD指针指向的不是分支而是某一个commit，是不能再提交的（提交没有意义），但是reset是“HEAD所在的分支”的头指针回退到历史commit上，是可以提交的，相当于回溯。checkout [哈希码] 的作用只是浏览。

git revert [哈希码] 新增一个commit，内容是抵消之前的修改（等价于回退，但是与reset不同的是，reset会真的删除一些commit，而revert会保留之前的任何commit，新增一个commit）

git reflog可以理解为显示本地操作记录。很多操作都会保存在这里：

```
lenovo@LAPTOP-PHNUOLVV MINGW64 /c/AAmyprog/Android/ypao/CQUPT_RunTogether (p/zha
ngzhe)
$ git reflog
212a680 (HEAD -> p/zhangzhe, origin/p/zhangzhe) HEAD@{0}: checkout: moving from
origin/p/zhangzhe to p/zhangzhe
212a680 (HEAD -> p/zhangzhe, origin/p/zhangzhe) HEAD@{1}: commit: [optimize]收起
bottomsheet时自动将地图中自己的位置居中
afa6f87 HEAD@{2}: checkout: moving from ktwork to origin/p/zhangzhe
9f4dcea (ktwork) HEAD@{3}: checkout: moving from origin/p/zhangzhe to ktwork
afa6f87 HEAD@{4}: reset: moving to afa6f8
9f4dcea (ktwork) HEAD@{5}: checkout: moving from origin/p/zhangzhe to origin/p/z
hangzhe
9f4dcea (ktwork) HEAD@{6}: checkout: moving from ktwork to origin/p/zhangzhe
9f4dcea (ktwork) HEAD@{7}: reset: moving to HEAD^
afa6f87 HEAD@{8}: reset: moving to afa6f8
9f4dcea (ktwork) HEAD@{9}: reset: moving to HEAD^
afa6f87 HEAD@{10}: commit: [feature]添加平均速度监控：每半分钟计算一次平均速度，
如果超过指定速度则认为不是在跑步，弹出提示，第二次超过指定速度弹出提示后会终止本
次运动并不保存
9f4dcea (ktwork) HEAD@{11}: rebase (finish): returning to refs/heads/ktwork
9f4dcea (ktwork) HEAD@{12}: rebase (start): checkout origin/ktwork
a161eeb HEAD@{13}: reset: moving to HEAD
a161eeb HEAD@{14}: checkout: moving from master to ktwork
9245dfa (origin/master, origin/HEAD, master) HEAD@{15}: clone: from git@github.c
om:RedrockMobile/CQUPT_RunTogether.git
```

这里前面是哈希码，用reset就可以轻松回滚：

```
lenovo@LAPTOP-PHNUOLVV MINGW64 ~/Desktop/g (master)
$ git reflog
4085629 (HEAD -> master) HEAD@{0}: checkout: moving from c424028e388fd71c7c6495d
920eb5960e4c1824a to master
c424028 HEAD@{1}: checkout: moving from master to c42402
4085629 (HEAD -> master) HEAD@{2}: commit: ii4ajiasasdfasdfk
a6db9ed HEAD@{3}: commit: hhhhhhhh
e7af927 HEAD@{4}: reset: moving to HEAD
e7af927 HEAD@{5}: commit: a c
c424028 HEAD@{6}: reset: moving to c424028
c424028 HEAD@{7}: reset: moving to c424028
4d48265 HEAD@{8}: reset: moving to HEAD^
c424028 HEAD@{9}: commit: a changed
4d48265 HEAD@{10}: commit (initial): init

lenovo@LAPTOP-PHNUOLVV MINGW64 ~/Desktop/g (master)
$ git log --oneline
4085629 (HEAD -> master) ii4ajiasasdfasdfk ihjkhkh
a6db9ed hhhhhhhh
e7af927 a c
c424028 a changed
4d48265 init
```

可以看到本次commit的哈希码是e7af927，所以回退本次操作是“commit后的操作”。

git remote 查看现有的仓库有哪些（一般只有一个origin仓库，这是默认的名称）

git remote -v 可以查看仓库的地址

```
lenovo@LAPTOP-PHNUOLVV MINGW64 ~/Desktop/g (master)
$ git remote -v
abc     https://github.com/sandyz987/test (fetch)
abc     https://github.com/sandyz987/test (push)
```

这里我没有origin仓库而是abc仓库，这里列举了下载地址和上传地址。

接下来我们可以git push abc 就会自动根据abc中的上传地址去上传。

git push 的一般用法：

git push [远程主机名/仓库名] [本地分支名]:[远程分支名] 会将某个分支推送到远程分支上

如：git push origin master这里省略了远程分支名，表示将本地的master分支推送到origin仓库。

如：git push origin :master这里省略的本地分支名，是删除远程分支master的命令，等价于：git push origin --delete master

git push [远程主机名/仓库名]表示将当前分支推送到[远程主机名]的对应分支。

git push如果只有一个主机则[远程主机名]可以省略，如果有多个远程主机，则需要git push -u origin master先上传一次，这样会将本地的master分支推送到origin主机，同时指定origin为默认主机，后面就可以不加任何参数使用git push了。这里的-u就是--set-upstream

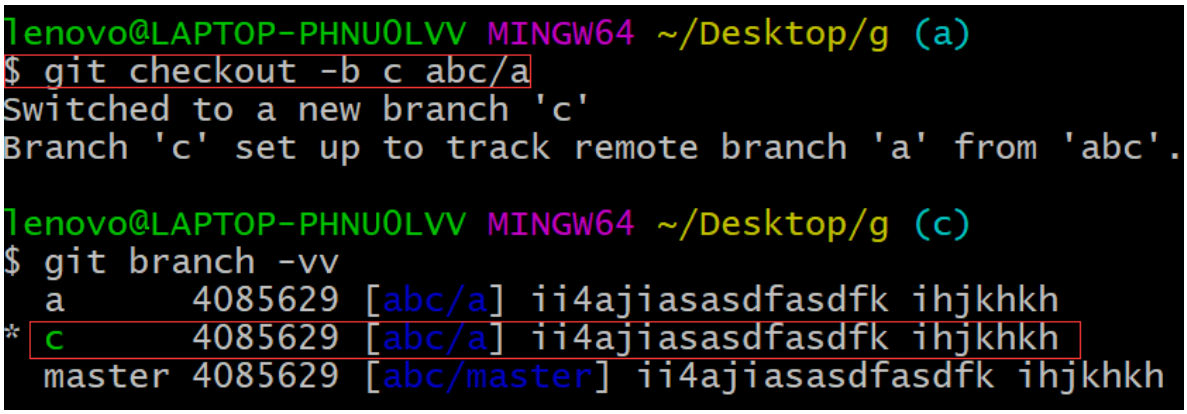
```
lenovo@LAPTOP-PHNUOLVV MINGW64 ~/Desktop/g (d)
$ git push abc
fatal: The current branch d has no upstream branch.
To push the current branch and set the remote as upstrea
m, use

    git push --set-upstream abc d
```

git push [origin 分支名] 会上传本地的分支到服务器。

参数-f表示强制推送，这样本地的会覆盖远程仓库的记录

git branch --set-upstream-to=[远程分支名] [本地分支名]这样设置本地和远程分支的对应关系（因为本地分支可能是你起的别名，没必要和远程分支同名）

git checkout -b [新建分支名] [要对映的远程分支] 一个快捷的用法，可以快速建立映射。本地如果直接创建一个分支，则是在当前分支上分离出一个分支（内容和原分支相同）



git fetch创建并更新本地的远程分支。即更新[主机名]/[分支名]，相当于仅仅下载（同步）远程仓库和本地仓库的“远程分支”。

git merge [分支名]

参数--commit（默认）自动生成一个提交，包含两个节点合并的结果。

参数--no-commit为了防止合并失败，先不提交，让用户手动提交

将当前分支向[分支名]合并，并生成一个合并后的commit

如果有冲突则要先fix conflicts，git会列出冲突的文件，我们需要打开这些文件一个一个地合并（git自动帮我们合并了全部，我们只用删掉不需要的部分即可），然后重新commit就行了（这时不用再merge了）

git rebase [分支名] 跟merge大同小异。

rebase和merge的区别：rebase是将你当前的修改作为[分支名]之后的修改（正如其名：在..基础上）

然后不产生提交，而merge是产生了一个新的提交。

git pull [远程主机名] [远程分支名]:[本地分支名] 相当于先执行git fetch [远程主机名]，然后再merge。

git stash 等同于git stash save，能够将所有未提交的修改（工作区和暂存区）快照至堆栈中，用于后续恢复当前工作目录。

git stash list 查看当前stash中的内容

git stash pop 将当前stash中的内容弹出，并应用到当前分支对应的工作目录上。相当于回退

git stash apply [stash名字] 将堆栈中的内容应用到当前目录

如：git stash apply stash@{1}

git stash drop [stash名字]

git stash clear 清空stash列表

git stash show 查看堆栈中最新保存的stash和当前目录的差异