# Java

Install Java:

sudo cp -r ~/Downloads/jdk-8u66-linux-x64.tar.gz /usr/local/java/

cd /usr/local/java/

sudo tar xzvf jdk-8u66-linux-x64.tar.gz

cd jdk1.8.0\_66/

sudo vi /etc/profile

JAVA\_HOME=/usr/local/java/jdk1.8.0\_20

JRE\_HOME=$JAVA\_HOME/jre

PATH=$PATH:$HOME/bin:$JAVA\_HOME/bin

export JAVA\_HOME

export JRE\_HOME

export PATH

sudo update-alternatives --install "/usr/local/java" "java" “/usr/local/java/jdk1.8.0\_66/jre/bin/java" 1

. /etc/profile

## Learn Java

2.4.1 实例变量如果没有显示的初始化会被初始化为0，false，null。使用null变量会引起assert.相反，方法的局部变量不会被初始化。

2.5.8 类的静态变量即使没有类实例也可以通过类名进行访问

2.6.1 线程可以共享变量和方法，同时可以有自己的局部变量

2.6.2 java.lang.Thread类的一个实例对应一个线程。线程通过start()方法开始执行，一旦开始执行，除非完成了工作，终端或者终止了应用，否则一直执行。Thread类实例总是希望执行run()方法。有两种方法指定run：

1， 派生Thread类，覆盖run方法。调用子类对象的start方法，将导致run在一个独立的线程中执行

2，用构造函数创建一个Thread对象，并提供一个参数，并调用其start方法，thread就会执行参数对象的run方法

2.6.6 当出现一个异常条件时，会创建一个对象，包含异常的相关信息。异常将导致代码段停止执行，并将控制转移到接受异常的代码。如果一直没有处理，java虚拟机会打印异常并退出。

2.6.6 如果有方法可能抛出异常，同时没有声明这段代码可能抛出该异常，则该异常必须被try/catch。

2.6.7 如果没有同步，同一个应用中的线程都在运行，但是不管其他线程在干什么，可以通过synchronized加入同步，在任何时刻，对象中只有一个方法能得到锁并运行。

3.3 java的main函数：public static void main(String[] myArgs)

3.3 java不推荐使用系统变量，而使用系统属性：

Java –dstreet=sesame -dname=value….

3.5 可以指定输出class目录而不是当前目录: java -d path \*.java

3.6.2 jar -tvf可以查看jar包的内容. jar -cvmf myManifest.mf space.jar space用于指定manifest文件。在manifest文件中可以指定Main-Class: com.oreilly.Game指定包含main方法的类，使jar包可以运行

4.3.2 java中的基本类型是按值传递的

4.3.3 java中的+是唯一得到重载的操作符

4.4 变量不能在定义它的大括号意外使用

4.4 增强for循环：

for( varDelare : iterable)

statement;

4.4.2 instanceof可以测试对象和类型是否是相同的类型，或者自类型，也可以用于测试数组，或者子类的数组

4.5.3 异常会被逐级上浮，直到被处理，然后从处理的位置继续执行。可以用printStackTrace()打印栈轨迹

4.5.6 重新抛出的异常显示为最初的位置

4.5.8 为了保证某些操作会运行，可以不用catch语句，而用finally语句

4.5.8 可以通过finally释放资源，但是finall中的释放语句还是有可能抛出异常，可以使用带资源的try形式代替：

try(

Socket sock = new Socket(“128.242.120.1”, 80;

FileWriter file = new FileWriter(“foo”);

){

//work with file and sock

} catch (IOException e){

….

}

4.7.2 数组必须指定大小，对数值数组，元素被初始化为默认值，对引用数组，元素被出示为null

4.7.3 如果访问超出数组范围的元素，会生成一个ArrayIndexOutOfBound Exception. 除非必要，捕获之并处理，一般情况下只要忽略

4.7.3 拷贝数组可以使用System.arraycopy(source, sourceStart, destination, destStart, length)或者copyOfRange (source, destlength). 如果copyOfRange中destlength比source长，剩余元素用0/null填充。

5.2.3 静态方法能看到该类的其它静态成员，但是无法看到任何实例变量，也不能调用实例方法

5.3.1 构造函数不能声明为static, abstract, synchronized, final，因为……

5.3.2 构造函数可以使用this或者super引用类或者超类的另一个构造函数

5.3.3 初始化块在对象构建的时候执行一次，在构造函数之前。静态初始化块在类加载时就会执行，早于所有的非静态初始化块和构造函数

6.1.1 类实例变量在父类的函数中读取为父类的值，在派生类的函数中读取为派生类的值。参见c06.OverideTest.java. 实例变量会从父类到派生类寻找，即使是一个派生类对象，但当作基类使用时看不到派生类独有的变量

6.1.2 Overide关键字是为了防止覆盖编程重载. 动态方法在绑定过程中不能修改static属性

6.1.2 final用于方法，表示该方法不能被覆盖。用于class，表示该类不能被派生

6.1.2 派生类的返回值可以发生某种程度的细化，即基类方法的返回值的派生类是允许的,但不能是完全不相关的类型

6.1.3 super引用父类的作用域

6.1.4 java中的强制转换只是影响对引用的处理，不会改变实际对象的形式

6.1.5 如果希望调用基类的一个带参数的构造函数，需要使用super关键字

6.2 接口中所有的方法默认是public、abstract的，但是不用写明，默认就是了。如果一个类实现了一个接口所有的方法，则可以说这个类implement了这个接口

6.2.2 接口中定义的变量自动为static和final的，不需要使用这些修饰符，也可以显示的加以说明。这种用法不常用，常被enum和static import代替

6.3.4 static import用于引入静态成员

6.4.1 可访问性: public>protected>no>private

6.4.2 子类覆盖父类代码的时候可以用较高可见性的比如public覆盖较低的比如private的函数

6.4.2 protected函数在子类中是可见的，但是只能通过子类类型的对象访问

6.5 对数组而言，子类型的数组是父类性数组的子类，可以赋值给父类型。如果父类型数组实际代表某一个子类型数组，但是又插入了不是该子类型的变量，则会得到运行时错误，错误类型是ArrayStoreException

6.6 B是A的内部类，创建B对象的形式是A.B ab = a.new B();如果内部类是static的，那么可以直接new

# Android

## 第一行代码

Android包括好几层：

第一层Android基于linux内核，版本可以通过uname –r查看。

第二层为C/C++实现，提供Android的主要特性，如SQLite，OpenGL。还有Android运行时，允许开发者使用java开发应用，同时运行时还包括Dalvik虚拟机，相较于java虚拟机，做了多种优化。

第三层应用框架层，提供应用程序可能需要的各种API

第四层，应用层。手机上的应用程序就属于这一层

Android四大组件包括Activity, Service, Broadcast Receiver, Content Provider

1.3.3 项目目录下的build目录用来存放生成的文件，不能手动修改

1.3.4 drawable是用来放图片的，values是用来放字符串的，layout是用来放布局文件的

menu是用来存放菜单文件的

1.3.4 在代码中引用资源可以用R.string.hello\_world, 在XML中是通过@string/hello\_world

1.4.1. 在代码中可以用Log.v()/d()/i()/w()/e()来记录log。不用System.out.println的原因有日志不可控制，打印时间无法确定，不能过滤

1.4.3 ^data$可以用来匹配data全字

2.2.1 添加activity步骤：

1）建立FirstActivity继承Activity，并覆盖onCreate

2）建立layout目录，建立LinearLayout。

3）在FirstActivity的onCreate函数中加入对setContentView的调用

4）在AndroidManifest文件中注册FirstActivity，并设置为默认启动activity

2.2.4 隐藏标题栏requestWindowFeature(Window.FEATURE\_NO\_TITLE);

2.2.5 添加Toast步骤，添加button，然后

button1.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Toast.makeText(FirstActivity.this, "You clicked Button 1",

Toast.LENGTH\_SHORT).show();

}

});

2.2.7 销毁活动需要调用finish();

2.3.2 显式Intent：

Intent intent = new Intent(FirstActivity.this, SecondActivity.class);

startActivity(intent);

隐式intent:

在manifest中定义activity 响应的intent，然后

Intent intent = new Intent("com.han.jm.activitytest.ACTION\_START1");

startActivity(intent);

2.3.2 每一个intent只能有一个action，但是可以有多个category

2.3.3 intent不仅可以用来启动自己程序内的活动，也可以用来启动其他程序的活动，可以在多个应用程序间实现功能共享:

Intent intent = new Intent("Intent.ACTION\_VIEW");

intent.setData(Uri.parse("http://www.baidu.com"));

startActivity(intent);

2.3.3 在intent中指定data，可以相应要响应的数据，可以为http，geo，tel

2.3.4 向一个活动传递数据：

发送方：

String data = "Hello SecondActivity";

intent.putExtra("extra\_data", data);

接收方：

Intent intent = getIntent();

String data = intent.getStringExtra("extra\_data");

2.3.5 如果希望得到返回的结果，在启动的时候：

startActivityForResult(intent, 1);

并覆盖onActivityResult, 此方法用于Activity销毁的时候将结果返回给上一个活动

在被启动的活动结束的时候, 可以是button.onClickLitener, onBackPressed

Intent intent = new Intent();

intent.putExtra("data\_return", "Hello FirstActivity");

setResult(RESULT\_OK, intent);

finish();

2.4.2 Activity的活动周期：

onCreate, onStart

onResume---另一个活动来到前台---> onPause

onStop, onDestroy, onRestart

2.4.4 android:theme属性可以为活动指定主题

2.4.4 声明周期的每一个函数，如果覆盖第一件事是要调用父类的函数

2.4.5 活动回收之前保存数据使用：

@Override

protected void onSaveInstanceState(Bundle outState){

super.onSaveInstanceState(outState);

String tmpData = "Sth you just typed";

outState.putString("data\_key", tmpData);

}

然后在onCreate函数中恢复：

if(savedInstanceState != null){

String tmpData = savedInstanceState.getString("data\_key");

Log.d(TAG, tmpData);

}

2.5 活动的启动模式包括4种:

Standard, 返回栈管理活动，不管栈中是否已经存在，每次启动都创建新的活动实例

, singleTop, singleTask, singleInstance

# Linux

## 常用linux命令：

#查找目录下所有文件中的”””，并一同显示之前的一行

find ./ -type f|xargs grep –B 1 “\”\”\””

# Assembly Language

## 王爽

1.2. 指令和数据放在内存中，CPU控制整个计算机的运作并进行运算

1.7 CPU和内存间有地址总线，数据总线，控制总线。其中地址总线决定CPU的寻址能力，也就是支持多达的内存。数据总线决定CPU和外界传输数据的速度，8086有16跟数据总线，一次可以传送16位数据。控制总线决定CPU对外部器件的控制能力。

1.10 CPU对外部器件发出命令是通过控制总线.控制总线的宽度决定了CPU对外部组件的控制能力

1.14 内存分为随机存储器和只读存储器，前者可以读写，后者只能读，向ROM写入数据是没有作用的，ROM数据不会被改变。

2. 内部总线用来连接CPU的内部组件，外部总线实现CPU和主板上的其他器件的联系

8086有14个寄存器- AX, BX, CX, DX, SI, DI, SP, BP, IP, CS, SS, DS, ES, PSW

AX, BX, CX, DX为通用寄存器，既可以用作16位寄存器，也可以作为8为寄存器。

2.3 当作为8位寄存器使用的时候，低位和高位之间没有关系，不会溢出。进行数据传送的时候，指令的两个操作对象应当一致，不能一个16位一个8位。

2.6 有20位地址总线，同时又是16位的CPU结构，当进行地址寻址时，需要进行地址加法，具体为物理地址=段地址\*16 + 偏移地址。

2.8 段地址的起始地址为16的倍数，段地址的大小最大为64K，因为偏移地址为16位

2.9 CS, DS, SS, ES为段寄存器，用来提供段地址。CS为代码段寄存器，IP为指令指针寄存器。在8086的机器中，任意时刻代码从16×CS+IP开始执行。

2.10 执行过程为从CS：IP读取指令，IP=IP+所读取的指令长度，执行指令。8086加电启动后，CS：IP被设置为FFFFH：0000H，所以第一条执行的指令为FFFF0H。内存中的数据即可以作为数据，也可以作为指令，根据是否是CS：IP指向的地址来确定

2.11 MOV指令并不能更改CS：IP的值，而应该使用jmp命令

Ex1 Debug是DOS、windows提供的实模式调试工具，可以查看各种寄存器的内容，内存的情况和跟踪程序运行。Debug的功能：

R：查看寄存器内容

D：查看内存中的内容

E: 改写内存中的内容

U：将内存中的机器指令翻译成汇编指令

T：执行一条机器指令

A：以汇编指令的形式在内存中写入机器指令

# Python

## Python核心编程

1.5.2 当不确定python在哪个目录下时，可以使用/bin/env python

2.0 计算绝对值abs(-43)

2.1 定义字符串myString = ‘This is a test String’

2.1 \_表示最后一个表达式的值

2.1 >>用于重定向

2.2 打印输出的格式有:

print ‘%s version is %d’%(“Python”,2),在此格式中若要输出%需要两个%%

print ‘your stirng is:’, user

Btw, 传递错误的类型并不会导致严重错误

2.2 标准输入user=raw\_input(‘Enter login name: ‘)

2.3 #表示comment

2.4 and or not 表示逻辑运算，比如2 < 4 and 2 == 4

2.5 python支持n \*= 10，但是不支持- -n/++n

2.6 python支持的数字类型为int, float, complex，其中int包括long bool

2.6 bool可以用int表示，false为0，true为1

2.7 [n:m]表示[n,m)区间

2.7 +代表拼接字符串，’’’用来去除特殊转义字符，\*代表重复字符串，比如’-‘\*20表示-----------------------

2.8 aList=[1,2,3,4]是列表，aTuple=(1,2,”a”,4)表示元组。列表元素可以更改,元组不可以

2.9 aDict={‘host’:’earth’}表示字典, aDict.keys()表示键值表。遍历字典：

For key in aDict:

print key, aDict[key]

2.11

if x < .0:

print “xxx”

elif x ==0:

print “xxx”

else:

print “xxx”

counter = 0:

while counter<3:

counter += 1

for item in [‘email’, ‘net’, ‘homework’]:

print item

for eachNum in [0,1,2]:

print eachNum

for eachNum in range(3):

print eachNum

for I, ch in enumerate(“abc”):

print ch, ‘%d’%i

2.14 列表解析：square=[x\*\*2 for x in range(6)]

2.15 读文件时open函数会把换行符一起读入,因此用print输出时需要加,来抑制重复换行

handle=open(file\_name, access\_mode=’r’)

如果不提供mode，默认是r，后续的文件操作都必须通过open返回的句柄

for eachline in handle:

print eachline,

handle.close()

2.18所有名字开始和结束都有两个下划线的函数都是特殊方法

类实例被创建时被执行，但不是构建器，而是类实例被创建后第一个被执行的函数.可以给用户自定义用来覆盖什么都不做的默认实现

\_\_init\_\_()

class的定义：

class ClassName (base\_class[es]):

"optional documentation string"

static\_member\_declarations

method\_declarations

self是实例自身的引用实例

创建实例fool = FooClass()

2.19 import模块时，这个模块应该被放置在我们输入它的程序的同一个目录中，或者在sys.path所列目录之一。

3.1.2 \用来连接多行,当有String用’’’包围，或者列表元祖用括号和方括号包围的时候不需要用\来连接多行

3.2 赋值可以连续赋值也,多重赋值或者交换，x = y = z = 1; x, y, z = 1, 2, 'a string'; x, y = y, x

3.3.1 Python变量名大小写敏感

3.3.4 \_XXX表示private变量

3.4.1 Python文件的结构一般是1. Startup line2. Module documentation3. Module imports4. Variable declarations5. Class declarations6. Function declarations7. "main" body

3.4.1 如果py文件是通过import运行，那么\_\_name\_\_是module name，如果是直接运行，则是\_\_main\_\_

3.5.1 Python中的变量不需要提前声明，在第一个赋值语句的时候会被自然初始化，未初始化的变量是不能被引用的

3.5.3 Python不用关心内存分配，在当对象被创建的时候，一个指向它的引用也生成了。当1）被赋值给其它变量2）函数调用3）编程容器的一部分时，引用计数会被增加

当1）超出作用域2）del x 3）引用其他对象4）从容器中删除时，引用计数会被减小

3.6 try/except中的else块用于当try块中没有exception抛出时需要执行的代码

3.7 pdb用于调试python代码，logging在python2.3中增加，用来添加log

4.2 Python中的基础类型有数字（Integer、float、复数），String，List，Tuple，Dictionary

4.3.1 type()用于显示一个数据的类型,type()的返回类型是type，也是所有类型的父类

4.3.2 None相当于void/NULL。在Python中None，false，0，0.0等表示boolean的false

4.5 a is b/a is not b用于检测同一性，等同于id(a) == id(b)

4.5.3 and, or, not 用于逻辑运算，如not (x < 5.0)，(x < 5.0) or (y > 2.718281828)

4.6 cmp(obj1, obj2),比较两个对象，返回integer

repr(obj), 也可以用’obj’表示，用于取得字符串表示， Python friendly

str(obj),用于取得可打印的字符串表示, human friendly

4.6.4 Python不支持函数重载

4.6.4 判断一个变量的类型：

if isinstance(num, (int, long, float, complex)):

print 'a number of type:', type(num).\_\_name\_\_

4.8.2 List, Dictionary可变，数字，字符串，tuple不可变。可以通过id()来验证

4.9 Python没有char和byte，可以用一个字符的字符串和integer代替

Python没有指针，但是有个id()可以查看对象的数字

Python不分int，short，long，都是根据需要自动变化的

Python的float就是C 的double

5.1 del x, 可以删除一个变量

6.1.2 当索引一个列表之外的值时，会抛出IndexError 异常. 但是当取一个范围的时候并不会产生error

6.1.2 [0], 表示第一个元素。[-1]表示最后一个元素。[:-3]表示从开始到倒数第三个元素，但是不包括倒数第三个元素。[-3:]表示倒数三个值，经常用于处理后缀名，比如.py为3个字

6.2 改变字符串中的一个字符会引起一个新的string对象被创建出来

6.2 单引号创建字符串，变量没有引号。双引号创建字符串，变量带引号

6.2 可以把字符串变量赋值为’’或者显示的调用del删除字符串变量

6.3 in, not in可以用来判断一个字符串时候在另一个里面，返回True/False

6.3 string.letters用来返回全部26个字母的大小写，string.digit用来返回数字0-9

6.3.2 在loop中，尽量不要采用重复的计算

6.3.2 while/for循环中的else，如果在for循环中没有break就会被执行

6.3.2 单引号引起来的字符串可以不用\在多行中显示

6.4.1 Python用%格式化字符串"MM/DD/YY = %02d/%02d/%d" % (2, 15, 67)

'There are %(howmany)d %(lang)s Quotation Symbols' % \

... {'lang': 'Python', 'howmany': 3}

'There are 3 Python Quotation Symbols'

6.5.3 chr()输入一个0-255之内的数字，返回字符

unichr()输入的是0-65535，返回unicode

ord()则输入字符，返回数字

6.6 String BIF, upper(), lower(),

6.7.1 三引号用来表示包括特殊字符的字符串

6.7.3 String是immutable的，modifly like append将修改字符串的id()返回值

6.8.5 decode, encode用来解码/压码字符串

6.9 re模块提供正则表达式支持

6.11 list可以用append增加元素. 如果知道index可以用del alist[1]删除，如果不知道index，可以用alist.remove(123)删除，但是通常都不需要手动删除

6.12 list可以in/not in 判断元素是否在list中，也可以用+连接两个list，或者append str

6.13 list有一些BIF，例如cmp, len,max, min, sorted, reversed,tuple也有类似的BIF

6.14 可以用dir(type)查看type类型的属性和内置函数

6.14 extend可以用来把一个iterable增加到另外一个list上

>>> motd = []

>>> motd.append('MSG OF THE DAY')

>>> f = open('/etc/motd', 'r')

>>> motd.extend(f)

>>> f.close()

>>> motd

['MSG OF THE DAY', 'Welcome to Darwin!\n']

pop(index)可以用来pop最后的元素，也可以指定要pop的元素

6.15 stack/ queue为list的特殊应用，有append，pop函数

6.18.2 tuple的每一个元素不可以更改，但支持加法，like t = t + ('fifth', 'sixth')

6.18.5 通过tuple(), list()可以在tuple和list之间互相转化

6.20 可以通过分割或者工厂函数拷贝hubby = person[:]或者wifey = list(person)。这种情况是浅拷贝，浅拷贝的意思是对list/tuple中的每一个元素，如果可变，那么两个引用指向的是同一个对象。

如果不可变，也指向同一个，但是如果改变了他们的值，就会自动产生一个新的对象。如果要进行深拷贝，需要调用copy.deepcopy()函数，但要注意，深拷贝当元素不可变的时候也是指向同一个对象，尽管进行的是深拷贝。当元素可变的时候，深拷贝才有用

6.21 String, list, tuple的BIF

7.1 创建dictionary：1) dict2 = {'name': 'earth', 'port': 80}

2) fdict = dict((['x', 1], ['y', 2]))

3) ddict = {}.fromkeys(('x', 'y'), -1)

7.1 遍历dictionary：for key in dict2.keys():

... print 'key=%s, value=%s' % (key, dict2[key]).1 如果尝试访问dictionary中不存在的元素，抛出KeyError

7.1 如果索引的元素不再dictionary中，会被自动加入

7.1 删除dictionary中的元素：del dict2['name']

7.2 in/not in可以用来查看dictionary中是否存在key

7.4 dictionationary BIF, dic.get(key, default)，取得key的值，不存在就返回default:

msgs = {'create': 'create item',

'delete': 'delete item',

'update': 'update item'}

default = 'invalid choice... try again!'

action = msgs.get(user.cmd, default)

7.5.1 dictionary不允许重复key.同一个key只能有一个值 List和dictionary不能用作key

7.6 set是没有排序的哈希值集合，包括set和frozenset，区别是一个可变，一个不可变

7.6 set的更新s.add/update(类似于add)/remove/-=

7.6 set支持|/&/-/&=/-=/^=

7.9 set BIF issubset/issuperset/union/intersection/copy/update/add/remove/pop/clear

8.1 if块中可以用not或者and，if not warn and (system > 10):

8.2 python根据缩进判断if/else属于哪个块

8.7 break既可以用在while中，也可以用在for中

8.9 pass可以作为占位符使用

8.11.6 创建一个iter, iter=iter(obj)

8.10 while或者for循环后的else，如果while没有被break，那么else将被执行

9.2.1 file和open是同样的功能

9.3.1 read函数把文件从头读到尾，并返回bytes

readline读取文件，并返回string

readlines读取文件，并返回list

CH10

11.3.3 Python的函数必须先声明在引用

11.3.5 inner函数不能在函数的外部引用

11.4 函数也可以向对象一样引用

11.6.1 argument之前的\*号表示该参数用来保存所有的剩余参数

12.2.2 增加系统路径sys.path.append('/home/wesc/py/lib')

12.4.4 别名：

import Tkinter as tk

from cgi import FieldStorage as form

12.6.1 import库时实际调用，该函数可以被覆盖

\_\_import\_\_ ()

14.1.4正常情况下，类实例不可以运行，但是覆盖了\_\_call\_\_()后则可以被call

14.3.4 f=open(‘abc.py’)

exec f

f.close()

f.seek(0)

14.3.5 raw\_input返回string，input则不不一定，有可能返回list

Q1: Python中单引号，双引号，三引号的区别.

A1: 单引号和双引号类似，都是用来表示字符串，三引号用来包括特殊字符的字符串

Q2: Python中\*, \*\*, \*\*\*的区别

# Continous Integration

## Pro Git

### 常用命令

2.4 #删除已经添加的文件

git reset HEAD .

2.7 创建别名

git config --global alias.co checkout

git config --global alias.br branch

git config --global alias.ci commit

git config --global alias.st status

3.1

创建分支 git co –b testing, -b表示创建完了就切换到该分支

切换到分支 git co testing, 切换到分支之前最好全部保存提交

3.2

合并分支,方法1：

git co master

git merge hotfix

如果和并的时候有一个分支可以走下去直接到达另外一个分支，合并是fast-forward

如果没有直线，则会做一个3方合并

如果有冲突，git会用<<</====/>>>>标出，可以通过git status查看，并编辑再git add&ci提交

方法2： experiment上有一些工作需要合并到master, rebase的好处是历史更加清晰

git co experiment

git rebase master

git co master

git merge experiment

#取出 client 分支，找出处于 client 分支和 server 分支的共同祖先之后的修改，然后把它们在 master 分支上重演一遍

git rebase --onto master server client

#将 server 分支中的修改也整合进来。 使用 git rebase [basebranch] [topicbranch] 命令可以直接将特性分支（即本例中的 server）变基到目标分支, 这样做能省去你先切换到 server 分支，再对其执行变基命令

git rebase master server

3.2

删除分支

git br –d hotfix

3.5

获取远程分支git clone

#获得远程分支

git remote show (remote)

#从远程抓取分支, 当抓取到新的远程跟踪分支时，本地不会自动生成一份可编辑的副本

git fetch origin

#如果要将这些工作合并到当前所在的分支

git merge origin/serverfix

#会查找当前分支所跟踪的服务器与分支，从服务器上抓取数据然后尝试合并入那个远程分支

git pull ebook master

#将修改提交到远程分支, :表示远程分支要另外起名字

git push origin serverfix

git push origin serverfix:awesomebranch

#如果合并的时候建立新的分支

git checkout -b serverfix origin/serverfix #1

git checkout --track origin/serverfix #2和1相同的功能

#设置已有的本地分支跟踪一个刚刚拉取下来的远程分支，或者想要修改正在跟踪的上游分支

git branch -u origin/serverfix

#查看跟踪的远程分支

git branch –vv

##删除分支

git push origin --delete serverfix

#revert本地的修改

git co <filepath>

#设置密码缓存时间：

git config --global credential.helper 'cache --timeout 360000'

### 初始化depot

git init

git config --global user.email "13240943@qq.com"

git config --global user.name "JM Han"

git add note/\*

git commit -m 'add note folder'

git remote add ebook https://github.com/dracohan/ebook.git

git remote set-url --add ebook https://github.com/dracohan/ebook.git

git remote –v

git push --set-upstream ebook master

git push ebook

### 强制更新：

git fetch --all

git reset --hard origin/master

## Practical Perforce

### 通用规则

1.1.2 …表示所有当前一级以及下一级文件

1.2 p4的命令参数，如果用双引号修饰，表明在传给p4之前不进行展开

### 常用命令

#查看用户信息

vmw\_ldap\_info hanb

#登录用户

p4\_login -Aa

#设置workspace

p4 client

#查看p4server 信息

p4 info

#设置p4 client

p4 set P4CONFIG=p4.config

Next, create files named p4.config in each of your workspace root directories. In each file, set the value of P4CLIENT

#查看文件

p4 files "//..."

#查找文件(包括delete的)

p4 files "//.../if.py"

#列出所有本地存在的文件

p4 have //…

#查找文件（不包括delete的）

p4 have //.../if.py

#查看目录

p4 dirs "//\*"

#查看file的map关系

#查看filelog

p4 filelog "//.../if.py"

#查看filelog，并包含integration历史

p4 filelog -i "//.../if.py"

#查看文件内容随版本的变化, -c代表输出changelist，而不是revision

p4 annotate -c //public/revml/revml.dtd

#（只显示plain text）

p4 print //public/revml/revml.dtd

#查看更改过的文件

p4 diff -sa

#查看integrate和resolve的文件

p4 diff -sb

#查找丢失的文件

p4 diff -sd

#强制sync未被打开但是丢失的文件, -x-使p4读取标准输入，而不是p4的参数

p4 diff -sd | p4 -x- sync -f

#check out未被打开但是更改过的文件

p4 diff -se | p4 -x- edit

#查看所有未被打开的文件的状态

p4 diff -sl

#查看打开了但是与depot相同的文件

p4 diff -sr

#ignore空格键的不同

p4 diff -db

#ignore空格键的存在与否

p4 diff -dw

#比较depot的两个文件

p4 diff2

p4 diff2 //public/revml/README#2 //public/revml/README#3

#查看p4文件类型

p4 help filetypes

#编辑

p4 edit index.html locations.html

#查看已经打开的文件

p4 opened

#查看谁在更改文件

p4 opened -a if.py

#查看更改的内容

p4 diff

#互换两个文件的内容

p4 integrate -i readme.txt misc.txt

p4 integrate -i misc.txt readme.txt

p4 resolve -at

p4 submit

#打印文件的内容, -q是省略p4加的文件头

p4 print -q ads.html#5

#增加文件

p4 add using\_mod.py

p4 add \*.py

p4 submit

#增加整个目录的文件

dir /s/b/a-d python | p4 -x- add -f

#删除

p4 delete using\_func.py

p4 submit

#提交修改

p4 submit

p4 submit -c 1406

p4 submit python/... #提交某一个目录下的所有修改, 必须在default changelist中执行

p4 submit –d ‘description’直接提交

#新建changelist

p4 change

#删除空白changelist

p4 change -d 1406

#查看pending的changelist

p4 changes -s pending -c scriptws

#移动文件到changelist中

p4 reopen -c 28 //.../\*

#查看changelist中的修改, -c表示希望的changelist

p4 opened -c 28

#删除client

#查看文件的changelist

p4 changes //depot/.../\*.html

#查看最近的5个修改

p4 changes -m5 "//.../if.py"

#查看changelist的信息

p4 describe -s 1

#查看changelist影响的文件

p4 files @=3947829

# resolve冲突

p4 resolve

#查看需要resolve的file, -o表示显示base

p4 resolve -n -o

#自动resolve

p4 resolve -as

#只接受theirs的resolve

p4 resolve -at

#只接受你自己的版本, always accept yours

p4 resolve -ay

#让p4merge

p4 resolve -am

#查看resolved结果

p4 resolved -o

#revert

p4 revert

#revert所有没有修改的或者丢失的文件，注意，resolve过的不会被自动revert

#-a表示opened的文件，而不是revert后面的参数

p4 revert -a

#sync代码

p4 sync -n #预览要sync的内容

p4 sync -f #强制sync

p4 sync "//depot/.../\*.py"

p4 sync //depot/dev/www/...@2004/08/10

p4 sync [//.../if.py@26](mailto://.../if.py@26)

#在local删除文件，并让depot知晓，相当于git的--cached

p4 sync c:\ws\dev\www\...#none

#直接delete文件

p4 delete //depot/…/\*.html

#查看最近一个提交

p4 changes -m1

#查看大于某一个changelist的所有changelist

p4 changes ... "@>83478"

#查看本地文件sync到哪个版本

p4 changes -m1 ...#have

#branch

p4 integ //Ace/MAIN/...@2004/10/12 //Ace/V1/...

#revert branch

p4 revert //depot/script/v1/...

#只integrate某一个changelist

p4 integ //Ace/V1/...@3459,@3459 //Ace/MAIN/...

#只创建不拷贝文件

p4 integ –v //Ace/MAIN/… //Ace/V1/…

#利用branch view操作integ

p4 integ -b V1toMAIN

#查看branches

p4 branches

#branch可以翻转

p4 integ -r -b V1toMAIN

### #Revert一个修改(1245)

p4 sync @1244

p4 files @=1245 | sed -n -e "s/#.\* - delete .\*//p" | p4 -x- add

p4 files @=1245 | sed -n -e "s/#.\* - edit .\*//p" | p4 -x- edit

p4 sync

p4 files @=1245 | sed -n -e "s/#.\* - add .\*//p" | p4 -x- delete

p4 resolve -ay

p4 submit

## Perforce mannual:

c02. p4 login –s #查看剩余的登录时间

c03. p4 delete path #删除文件，路径可以为本地目录\depot目录\或者client目录，都一样

c03. 如果路径名有空格，用双引号括起来

c03. 指定版本：file#n, file@cl, file#none(删除文件),file#head(最新版本), file#have(有的版本)

c03, p4 changes //depot/dev/main/…@2010/7/1,2010/8/1

## Real Command:

Merge process:

0, p4 sync …>/dev/null

1, source ../../env.global

2, source ../env.bfg.merge

2, export fromCLN=4015138

export logdir=$(date "+$mergetopdir/logs/$fromBranch-$branch-%Y-%m-%d-%H%M")

cbsMerge1 $partialArg -c $P4CLIENT -It -l $logdir $fromBranch $branch @$fromCLN

exit 1

cat $logdir/dbsIntegrate1.summary

p4 resolve –n ...

p4 change $pendingCLN(remove files with confict)

p5 submit –c $pendingCLN

cbsmergedb --database=cbsmerge --file=$logdir/json --host=osdc-st1-bms-db001 --change=@{changelist} > $logdir/mergedb 2>&1

如果有-d的情况：

1. 重新跑cbsMerge1生成description
2. integInfo=$(fromto2branchr $fromBranch $branch 2>/dev/null)
3. integBranch=$(echo $integInfo | cut -d' ' -f1)
4. p4 integ -c 4015952 -d -t -b ${integBranch} //...@${fromCLN}

同样的步骤在main到stage再来一遍

yes | sh -x $mergeprocessbin/build\_sandbox --branch $branch --cln 4015970 > $logdir/build\_sandbox 2>&1（pending README number）

run\_test-esx $branch 4015973 > $logdir/run\_test-esx 2>&1（latest submitted）