■■■■ Ubuntu

1. **快捷键**

|  |  |
| --- | --- |
| 功能 | 快捷键 |
| 打开主文件夹 | Alt + Home |
| 打开终端 | Ctrl + Shift(Alt) + T |
| 显示快捷键 | 长按 windows 键 |
| 让刚才的配置生效 | source /etc/profile |
| 切换到 tty | Ctrl + Alt + Fn |
| 锁屏 | Ctrl + Alt + L |
| 显示文件属性 | Alt + Enter |
| 局部截图工具 | Shift+PritScreen |
|  |  |
|  |  |

1. 命令

|  |  |
| --- | --- |
| 功能 | 快捷键 |
| 查看硬件详细信息 | lshw |
| 查看几个CPU | more /proc/cpuinfo |grep "physical id"|uniq|wc -l |
| 查看几个核 | more /proc/cpuinfo |grep "physical id"|grep "0"|wc -l |
| 查看当前操作系统内核信息 | uname -a |
| 查看内存 | free |
| 复制文件夹 | cp –r |
| 删除文件夹 | rm –rf |
| 显示文件的有关信息，类型、时间等 | file |
| 显示现有程序 | ps -aux | grep jdk 用于查找含有“jdk”的所有的程序 |
| kill一个程序 | pstree –p 找到进程 PID  kill PID |
| find 路径 -name 文件名 (-print)  find 路径 -iname 文件名 (-print) 不分大小写 | find |
| grep "被查找的字符串" 文件名  例子：在当前目录里第一级文件夹中寻找包含指定字符串的.in文件  grep "thermcontact" \*/\*.in | grep |
|  |  |

1. 终端快捷键

|  |  |
| --- | --- |
| 快捷键 | 功能 |
| Ctrl + k | 删除从光标到行尾的部分 |
| Ctrl + u | 删除从光标到行首的部分 |
| Ctrl + l | 相当于clear |
| Shift + PgUp | 向前翻看,一般翻13页左右 |
| Shift + PgDown | 向后翻看,一般翻13页左右 |
| Ctrl+Shift+up | 上滚屏 |
| Ctrl+Shift+Down | 下滚屏 |
|  |  |
|  |  |

1. 有用的工具命令

|  |  |
| --- | --- |
| 工具 | 功能 |
| tar -xf abc.tar.gz -C path | 把压缩文件解压到指定的path路径下 |
| tar zcvf xxxx.tar xxx/ | 把指定文件夹内容打包为指定文件 |
| touch | 作用1：创建空文件  作用2：更改文件修改时间 |
|  |  |
| ln -s source dest | 创建符号链接（最好都用绝对路径） |
|  |  |
|  |  |

1. bash

|  |  |
| --- | --- |
| 命令 | 功能 |
| $(cmd) | 命令替换，如path=$(pwd) |
| (cmd1;cmd2;...cmdn) | 命令组 |
| [ ] | 等同于test |
|  |  |
|  |  |
|  |  |
|  |  |

1. 软件管理

apt-cache search # ------(package 搜索包)

apt-cache show #------(package 获取包的相关信息，如说明、大小、版本等)

apt-get install # ------(package 安装包)

apt-get install # -----(package --reinstall 重新安装包)

apt-get -f install # -----(强制安装, "-f = --fix-missing"当是修复安装吧...)

apt-get remove #-----(package 删除包)

apt-get remove --purge # ------(package 删除包，包括删除配置文件等)

apt-get autoremove --purge # ----(package 删除包及其依赖的软件包+配置文件等（只对6.10有效，强烈推荐）)

apt-get update #------更新源

apt-get upgrade #------更新已安装的包

apt-get dist-upgrade # ---------升级系统

apt-get dselect-upgrade #------使用 dselect 升级

apt-cache depends #-------(package 了解使用依赖)

apt-cache rdepends # ------(package 了解某个具体的依赖,当是查看该包被哪些包依赖吧...)

apt-get build-dep # ------(package 安装相关的编译环境)

apt-get source #------(package 下载该包的源代码)

apt-get clean && apt-get autoclean # --------清理下载文件的存档 && 只清理过时的包

apt-get check #-------检查是否有损坏的依赖

apt-file search filename -----查找filename属于哪个软件包

apt-file list packagename -----列出软件包的内容

apt-file update –更新apt-file的数据库

1. apt
2. 安装

# sudo apt-get install xxxx

# sudo apt-get -f install # -----(强制安装, "-f = --fix-missing"当是修复安装吧...)

1. 更新

apt-get update #------更新源

apt-get upgrade #------更新已安装的包

1. 查询

# apt-cache search # ------(package 搜索包)

# apt-cache show #------(package 获取包的相关信息，如说明、大小、版本等)

1. 卸载

# sudo apt-get remove xxxx （移除式）

# sudo apt-get remove –purge xxxx （清除式，清除配置，--purge表示彻底删除软件和相关文件）

1. dpkg

dpkg --info "软件包名" –列出软件包解包后的包名称.

dpkg -l --列出当前系统中所有的包.可以和参数less一起使用在分屏查看. (类似于rpm -qa)

dpkg -l |grep -i "软件包名" –查看系统中与"软件包名"相关联的包.

dpkg -s 查询已安装的包的详细信息.

dpkg -L 查询系统中已安装的软件包所安装的位置. (类似于rpm -ql)

dpkg -S 查询系统中某个文件属于哪个软件包. (类似于rpm -qf)

dpkg -I 查询deb包的详细信息,在一个软件包下载到本地之后看看用不用安装(看一下呗).

dpkg -i 手动安装软件包(这个命令并不能解决软件包之前的依赖性问题),如果在安装某一个软件包的时候遇到了软件依赖的问题,可以用apt-get -f install在解决信赖性这个问题.

dpkg -r 卸载软件包.不是完全的卸载,它的配置文件还存在.

dpkg -P 全部卸载(但是还是不能解决软件包的依赖性的问题)

dpkg -reconfigure 重新配置

1. 安装

# dpkg -i xxxx.deb

1. 卸载

# dpkg -r xxx （移除式）

# dpkg -P xxxx （清除式，清除配置）

1. Deb包

* 安装deb软件包

命令： dpkg -i xxx.deb

* 删除软件包

命令： dpkg -r xxx.deb

* 连同配置文件一起删除

命令： dpkg -r --purge xxx.deb

* 查看软件包信息

命令： dpkg -info xxx.deb

* 查看文件拷贝详情

命令： dpkg -L xxx.deb

* 查看系统中已安装软件包信息

命令： dpkg -l

* 重新配置软件包

命令： dpkg-reconfigure xxx

1. 系统管理
2. 开机管理
3. 调节亮度

亮度相关的文件在： /sys/class/backlight/intel\_backlight/brightness

sudo vim /etc/rc.local，在 exit 0 之前，加入：

echo 300 > /sys/class/backlight/intel\_backlight/brightness

300为预设亮度

1. 设置蓝牙开关

sudo vim /etc/rc.local

加入：rfkill block bluetooth

1. 设置默认打开程序

设置 -> 系统设置 -> 详细信息 -> 默认应用程序

1. 设置pdf的默认打开程序

右键 -> 属性 -> 打开方式 -> 选择xournal -> 设为默认值

1. 系统文件

|  |  |
| --- | --- |
| 文件 | 功能 |
| /etc/profile | 此文件为系统的每个用户设置环境信息,当用户第一次登录时,该文件被执行.并从/etc/profile.d目录的配置文件中搜集shell的设置. |
| /etc/bashrc | 为每一个运行bash shell的用户执行此文件.当bash shell被打开时,该文件被读取 |
| ~/.bash\_profile | 每个用户都可使用该文件输入专用于自己使用的shell信息,当用户登录时,该文件仅仅执行一次!默认情况下,他设置一些环境变量,执行用户的.bashrc文件. |
| ~/.bashrc | 该文件包含专用于你的bash shell的bash信息,当登录时以及每次打开新的shell时,该该文件被读取. |
| ~/.bash\_logout | 当每次退出系统(退出bash shell)时,执行该文件. |
|  |  |
|  |  |
|  |  |
|  |  |

1. 磁盘管理
2. 查看磁盘容量

df -hl

dd

df -hl 查看磁盘剩余空间

df -h 查看每个根路径的分区大小

du -sh [目录名] 返回该目录的大小

du -sm [文件夹] 返回该文件夹总M数

1. 硬盘复制

可见http://m.blog.csdn.net/article/details?id=12616483

1. 方法一：

要保证两块硬盘的数据完全相同

dd if=/dev/sda of=/dev/sdb bs=1024K

if：输入位置；

of：输出位置，可以是硬盘的盘符，也可以某个文件路径；

bs：缓存大小，合适的缓存能够加快复制速度；

1. 方法二：

1、安装Linux操作系统到U盘中。最简单的办法就是在Windows环境下，使用unetbootin把ubutu桌面版的ISO镜像（如ubuntu-12.04-desktop-i386.iso）写到U盘上。

2、把U盘接到安装有母盘的设备中，并设置从U盘启动。

3、复制MBR到U盘中。

sudo dd if=/dev/sda of=/media/back/mbr.img bs=1024K count=1

说明：使用缓存1M，复制一次，也就是总共复制了硬盘最前面的1M数据，包括了MBR（MBR总共512字节）。

2、复制分区数据到文件中，假设操作系统安装在分区sda1，数据保存在分区sda2。

sudo mount -o rw /dev/sda1 /mnt

sudo tar -cpzvf /media/back/bak.os.tgz-C /mnt . --exclude=./mnt/tmp

sudo umount /mnt

sudo mount -o rw /dev/sda2/mnt

sudo tar -cpzvf /media/back/bak.data.tgz-C /mnt .

sudo umount /mnt

3、把U盘接到需要复制的目标设备上，设置从U盘启动。

4、复制MBR到目标盘上，在目标设备上，硬盘同样是sba。

sudo dd if=/media/back/mbr.img of=/dev/sda bs=1024K count=1

5、格式化目标盘的分区，然后复制分区数据到目标盘上。

sudo mkfs -t ext4 /dev/sda1

sudo mkfs -t ext4 /dev/sda2

sudo mount -o rw /dev/sda1/mnt

sudo tar -xpzvf /media/back/bak.os.tgz  -C/mnt

sudo umount /mnt

sudo mount -o rw /dev/sda2/mnt

sudo tar -xpzvf /media/back/bak.data..tgz  -C/mnt

sudo umount /mnt

此方法的优点在于复制时间短，尤其在硬盘容量大，数据量小的情况下，非常明显。同样是一块500G的硬盘，只使用了3G的数据，普通 5400转的硬盘，以50M/S的速度算，1分钟左右就可以复制完成，就算加上系统切换时间，也比dd 500G硬盘要少得多。缺点在于前期准备工作比较多。另外，推荐把命令写在脚本中，直接执行脚本可以节省不少敲命令的时间。

1. 方法三（最方便）：

使用再生龙

1. 系统安装步骤
2. 分区

/boot: 200 M

swap: 8192 M

/: 102400 M

/home: 剩余

1. 调整工具条

系统设置 -> 外观 -> 右下角“启动图标大小”

1. 把 home 下的文件夹名改为英文

1. 打开终端，在终端中输入命令:

# export LANG=en\_US

# xdg-user-dirs-gtk-update

2. 在弹出的窗口中询问是否将目录转化为英文路径,同意并关闭.

3. 在终端中输入命令:

# export LANG=zh\_CH

4. 重启，选择不替换

1. 更改终端颜色

vi ~/.bashrc 在最后添加：

PS1='${debian\_chroot:+($debian\_chroot)}\[\033[00;32m\]\u @ \h\[\033[00m\]:\[\033[00;34m\]\W\[\033[00m\]\$ '

注意，后面的w表示完整路径，W表示仅当前文件夹

也可以不要 :@host这些重复的东西，如下：

PS1='${debian\_chroot:+($debian\_chroot)}\[\033[00;32m\]\u:\[\033[00;34m\]\w\[\033[00m\]\$ '

1. 设置终端透明

打开终端 →编辑→配置文件→编辑默认的背景

1. 修改sourcelist

要把 /etc/apt/sources.list 替换，然后sudo apt-get update，不然找不到包

1. 装 vim

# sudo apt-get remove vim-common

# sudo apt-get install vim

# sudo apt-get install vim-gnome

1. 启动器图标大小

设置 -> 外观 -> 右下角，启动器图标

1. 安装git

# sudo apt-get update

# sudo apt-get install git

然后配置自己的git账号信息，这样才能进行相关的操作：

# git config --global user.name "markgang"

# git config --global user.email " markgang@126.com"

可以通过下面的命令查看git的配置信息：

# git config –list

显示信息如下：

user.name=Your Name

user.email=youremail@domain.com

这样就表示git的安装告一段落，可以使用git的一些命令了，比如git clone。

1. 安装 atom

# sudo add-apt-repository ppa:webupd8team/atom

# sudo apt-get update

# sudo apt-get install atom

快捷方式：

[Desktop Entry]

Encoding=UTF-8

Name=Atom

Comment=Atom

Exec=/usr/bin/atom

Icon=/usr/share/icons/hicolor/128x128/apps/atom.png

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

1. chrome

下载安装文件后：

# dpkg –i google-chrome-stable\_current\_amd64.deb

如果遇到错误：

dpkg: 处理软件包 google-chrome-stable (--install)时出错：

依赖关系问题 - 仍未被配置

正在处理用于 man-db (2.6.7.1-1ubuntu1) 的触发器 ...

正在处理用于 gnome-menus (3.10.1-0ubuntu2) 的触发器 ...

正在处理用于 desktop-file-utils (0.22-1ubuntu1) 的触发器 ...

正在处理用于 bamfdaemon (0.5.1+14.04.20140409-0ubuntu1) 的触发器 ...

Rebuilding /usr/share/applications/bamf-2.index...

正在处理用于 mime-support (3.54ubuntu1.1) 的触发器 ...

在处理时有错误发生：

google-chrome-stable

则可以先修复依赖关系：

# sudo apt-get -f install

然后再

# dpkg –i google-chrome-stable\_current\_amd64.deb

快捷方式：

[Desktop Entry]

Encoding=UTF-8

Name=chrome

Comment=chrome

Exec=/opt/google/chrome/chrome

Icon=/opt/google/chrome/product\_logo\_48.png

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

1. github

* 安装

# sudo apt-get install git

# sudo apt-get install git-doc git-svn git-email git-gui git-gitk

* 配置

# git config --global user.name "MarkGang"

# git config --global user.email ["menggangmark@126.com"](mailto:\"menggangmark@126.com\")

# git config --global credential.helper cache

# git config --global credential.helper 'cache --timeout=3600'

* 创建验证用的公钥

# ssh-keygen -C 'you email address@gmail.com' -t rsa

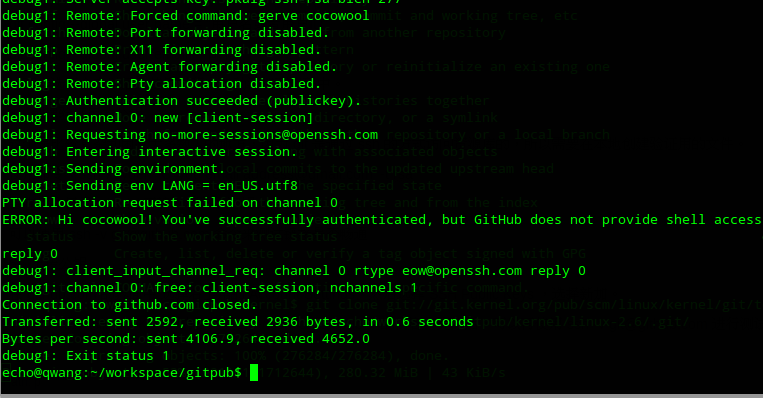
会在用户目录 ~/.ssh/ 下建立相应的密钥文件

可以使用 ssh -v git@github.com 命令来测试链接是否畅通，不通。

* 上传公钥

在 github.com 的界面中，personal settings里面，SSH and GPG keys，选择新加。

Title 可以随便命名，Key 的内容拷贝自 ~/.ssh/id\_rsa.pub 中的内容，完成后，可以再使用 ssh -v git@github.com 进行测试。看到下面的信息表示验证成功。



* 利用git管理本地项目

1、建立仓库

在需要建立项目的文件夹中，使用 git init 进行仓库的建立。完成后，可以看到文件家中多了一个 .git 隐藏目录。

2、添加文件

使用 git add . 来进行初始文件的添加。这里 . 表示将文件夹下所有的文件都添加进去，我们也可以指定文件进行添加。

3、提交文件

使用 git -m commit -m 'comment' 提交，可以将编辑的内容进行提交。

通过 git show 可以看到项目的一些状态。

* 利用git托管项目

创建repository，

* 有用的命令

cd xxx

git init

echo "xxx" > .gitignore

git add .

git commit -m "my message"

git push -u xxx url

git status

|  |  |
| --- | --- |
| 命令 | 作用 |
| git init | 创建一个版本库 |
| git add <modified files> | 每次修改好了后，可以先将修改存入stage(快照/索引)中 |
| git add . | 修改了大量文件则使用下面这个命令批量存入 |
| git commit -m "msg" | 使用commit将快照/索引中的内容提交到版本库中 |
| git commit -a -m "msg" | 也可以将git add与git commit用一个指令完成 |
| git push | 将本地的git档案与github(远程)上的同步 |
| git pull | 将github(远程)的git档案与本地的同步(即更新本地端的repo)，例如,pull指令其实包含了fetch(將变更复制回來)以及merge(合并)操作  git pull git://github.com/tom/test.git |
| git branch | 另外版本控制系統的branch功能也很有意思，若同时修改bug，又要加入新功能，可以fork出一个branch：一个专门修bug，一个专门加入新功能，等到稳定后再merge合并  git branch bug\_fix # 建立branch，名为bug\_fix  git checkout bug\_fix # 切换到bug\_fix  git checkout master #切换到主要的repo  git merge bug\_fix #把bug\_fix这个branch和现在的branch合并  若有remote的branch，想要查看并checkout  git branch -r # 查看远程branch  git checkout -b bug\_fix\_local bug\_fix\_remote #把本地端切换为远程的bug\_fix\_remote branch并命名为bug\_fix\_local |
| git log | 可以查看每次commit的改变 |
| git diff | 可以查看最近一次改变的內容，加上参数可以看其它的改变并互相比较 |
| git show | 可以看某次的变更 |
| git status | 目前工作树的状态 |
| git clone | 复制 |

1. 工具软件

|  |  |  |
| --- | --- | --- |
| 工具 | 功能 | 安装 |
| meld | 文件比较 | # sudo apt-get install meld |
| sysv-rc-conf | 开机启动管理 | # sudo apt-get install sysv-rc-conf |
| xournal | pdf标注 | # sudo apt-get install xournal  **[Desktop Entry]**  **Encoding=UTF-8**  **Name=xournal**  **Comment=xournal**  **Exec=/usr/bin/xournal**  **Icon=/usr/share/xournal/pixmaps/xournal.png**  **Terminal=false**  **StartupNotify=true**  **Type=Application**  **Categories=Application;Development;** |
| nautilus-open-terminal | 当前位置打开终端的工具 | # sudo apt-get install nautilus-open-terminal |
| feh | 图片查看工具 | # sudo apt-get install feh |
| zotera | 文献管理 | 解压，直接用  **[Desktop Entry]**  **Type=Application**  **Name=zotero**  **GenericName=zotero**  **Comment=zotero**  **Exec=/home/mg/mg\_setups/zotera/Zotero\_linux-x86\_64/zotero**  **Icon=/home/mg/mg\_setups/zotera/Zotero\_linux-x86\_64/chrome/icons/default/main-window.ico**  **Terminal=false**  **Categories=Development;zotero;** |
| latex |  | #sudo apt-get install texlive-full  #sudo apt-get install texmaker  #sudo apt-get install latex-cjk-all  **[Desktop Entry]**  **Encoding=UTF-8**  **Name=TexMaker**  **Comment=texmaker**  **Exec=/usr/bin/texmaker**  **Icon=/usr/share/pixmaps/texmaker.png**  **Terminal=false**  **StartupNotify=true**  **Type=Application**  **Categories=Application;Development;** |
|  |  |  |

■■■■ IPython

1. 自定义命令

import time

from IPython.core.magic import (

register\_line\_magic,

register\_cell\_magic,

register\_line\_cell\_magic

)

def \_mg\_exec\_time(code):

begin\_time = time.time()

exec(code)

print "used time:", time.time() - begin\_time

@register\_line\_magic

def mg\_time(line):

\_mg\_exec\_time(line)

@register\_cell\_magic

def mg\_time(line, cell):

\_mg\_exec\_time(cell)

@register\_line\_cell\_magic

def line\_cell\_my\_time(line, cell=None):

if cell is None:

\_mg\_exec\_time(line)

else:

\_mg\_exec\_time(cell)

把这个文件以.py结尾命名保存到~/.config/ipython/profile\_default/startup中，每次启动IPython的时候就会运行这个目录下的文件。重新进入IPython，进行测试：

In [2]: %line\_my\_time a = 1

used time: 0.000108957290649

In [3]: %%cell\_my\_time

...: for i in xrange(10000):

...: pass

...:

used time: 0.00216698646545

**！！但是，目前还不知道在anaconda下应该放入哪个路径。！！**

1. IPython技巧

|  |  |
| --- | --- |
| 快捷键 | 功能 |
| help | 查看帮助 |
| run | 运行源文件(命令行参数调试)，例如  # run xxx.py pa1 pa2 --p3=m2 |
| docstrings | documentation string，指def hello(): 下面的一行，'''或者是""" |
| F9 | 运行选定或当前行，在运行的同时，更改了加载的函数模块，在此运行模块时，将以新内容运行清空 |
| %reset | name space |
| “PEP8” | Style Guide for Python source code known as |
|  | IPython 的符号表达式的显示要在白色背景才能看到 |
|  |  |
|  |  |

1. ipdb

其实就几步：

* 出错后，%debug进入调试，
* 定位出错位置：

u: stack trace up

d: stack trace down

* 查看代码和变量

w: where，显示当前堆栈

l: list，显示前后几行源码

p: print values，或者直接写变量名

* 控制运行

n: next statement

s: step into

r: return

1. 控制图像显示方式

%matplotlib inline：

%matplotlib qt：

■■■■ Spyder

1. 技巧
2. 控制台中，在变量名后输入 ？ 可显示说明
3. 按住 Ctrl 加变量名可跳转到定义

可以设定layout方便布局

使用source toolbar 有回到刚才编辑位置的快捷键

■■■■ VI

1. 移动

|  |  |  |
| --- | --- | --- |
| 范围 | 功能 | 快捷键 |
| **全文** | 全文第一行 | gg |
| 全文最后一行 | G |
| 按行号跳转 | nG |
| 向前翻整页 | Ctrl + f |
| 向后翻整页 | Ctrl + b |
| 向前翻半页 | Ctrl + u |
| 向前翻半页 | Ctrl + d |
| 屏幕第一行 | H |
| 屏幕中间 | M |
| 屏幕最后一行 | L |
| 当前行的第一列 | 0或| |
| 当前行的第n列 | n| |
| **段落** | 开头 | { |
| 结尾 | } |
| **句子** | 开头 | ( |
| 结尾 | ) |
| **行内** | 本行的行首 | shift+^ |
| 本行的行尾 | shift+$ |
| 当前位置的后一个单词的首字母 | shift+W |
| 当前位置的前一个单词的首字母 | shift+b |
| 当前位置的后一个单词的尾字 | shift+e |
| 前一个字开头 | b |
| 前一个字开头（忽略标点） | B |
| 下一个字开头 | w |
| 下一个字开头（忽略标点） | W |
| 下一个字结尾 | e |
| 下一个字开头（忽略标点） | E |
| **其它** | 匹配括号 | % |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. 编辑

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 快捷键 |
| **删除** | 清空文件 | gg dG |
| 删除光标处字符 | x |
| 删除光标前字符 | X |
| 删word | dw |
| 删行 | dG |
|  | 将用户缓冲区内容放到光标位置 | p |
|  | 显示当前文件名 | Ctrl + G  实际上就是 Ctrl + Shift + g |
|  |  |  |

1. 查找与替换

|  |  |  |
| --- | --- | --- |
| 范围 | 功能 | 快捷键 |
| 1 | 向前查找 | / |
| 2 | 向后查找 | ? |
| 3 | 在同一方向重复查找 | n |
| 4 | 在相反方向重复查找 | N |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |

1. 宏

键入"qa"开始把一段宏记录入寄存器变量'a'中，再次再下"q"键,

就结束了宏'a'的录制。当你要重复执行你刚才记录的那些操作时只要使用"@a"命令。共有26个可用的寄存器供你记录宏。

1. 命令

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 快捷键 |
| 1 | 重复上一个动作 | . |
| 2 | 显示行号 | set nu |
| 3 | 取消行号 | set nonu |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |

■■■■ Emacs

1. Desktop

[Desktop Entry]

Encoding=UTF-8

Name=Emacs

Comment=Emacs

Exec=/usr/bin/emacs

Icon=/usr/share/emacs/24.3/etc/images/splash.png

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

C-g ： 取消命令

ESC ESC ESC。这是一个最通用的“离开”命令

要使用帮助（Help）功能，请先输入 C-h，F1，然后再输入一个字符以说明你需要什么帮助。如果你连自己到底需要什么帮助都不知道，那么就输入 C-h ?，Emacs会告诉你它能提供了哪些帮助。如果你按了 C-h 又想反悔，可以用 C-g 取消

C-h c

想得到更多的信息，请把 C-h c 换成 C-h k

C-h f解释一个函数

C-h v 用来显示 Emacs 变量的文档

1. 移动

|  |  |  |
| --- | --- | --- |
|  | 快捷键 | 功能 |
| 字符 | 上一行 | C-p |
| 下一行 | C-n |
| 左一字符 | C-b |
| 右一字符 | C-f |
| 词 | 前一个词 | M-f |
| 后一个词 | M-b |
| 屏 | 下一屏 | C-v |
| 上一屏 | M-v |
|  | 光标居中 | C-l |
| 行 | 行首 | C-a |
| 行末 | C-e |
| 句 | 句首 | M-a |
| 句末 | M-e |
| 文章 | 文章开头 | M- shift+ < |
| 文章末尾 | M- shift+ < |
| 页 | 到页首 | C-x [ |
| 到页尾 | C-x ] |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

C-u, n,（或M-n）， 动作：重复动作n次（可以重复输入多个字符）

1. 文件

|  |  |  |
| --- | --- | --- |
|  | **快捷键** | **功能** |
|  |  |  |
|  |  |
|  |  |
|  |  |

1. 窗格

|  |  |  |
| --- | --- | --- |
|  | **快捷键** | **功能** |
|  | 只保留光标所在窗格 | C-x 1 |
| 将屏幕划分成上下两个窗格 | C-x 2 |
| 将屏幕划分成左右两个窗格 | C-x 3 |
|  |  |
|  |  |  |
|  |  |
|  | 切换光标到另一个窗口 | C-x o |
|  |  |
|  | 增加窗口高度 | C-x ^ |
|  | 减小窗口宽度 | C-x { |
| 增加窗口宽度 | C-x } |
|  |  |  |
|  |  |

C-x C-f 寻找文件。

C-x C-s 保存文件。

C-x s 保存多个缓冲区

C-x C-b 列出缓冲区。

C-x b。切换缓冲区，你必须输入缓冲区的名称

C-x C-c 离开 Emacs。

C-x u 撤销

（“o”指的是“其它（other将光标转移到下方的窗格

C-M-v 滚动下方的窗格

【向上滚动是 C-M-S-v，也就是同时按住 CONTROL、META 和 SHIFT 再按 v

1. 编辑

M-backspace ：删除光标前一个单词

M-d ：删除光标后一个单词

C-k ：删除到行尾的字符（kill）

C-u, n, C-k：删除n行

M-k ：删除到句尾的字符

C-shift+ @ ：标记

，C-w：del选中

C-y ：粘贴所有内容，yanking

M-y ：重复召回前一次

C-x, u （C-shift+ -）：撤销一次

C-x C-f 寻找一个文件

时可以用 C-g 取消

关掉文件备份可以用如下命令：

M-x customize-variable <Return> make-backup-files <Return>

C-x C-b 列出缓冲区

退出 Emacs 用的 C-x C-c

在允许 C-z 的系统中，C-z 会把 Emacs“挂起”，也就是说，它会回到 shell

但不退出 Emacs。在常用的 shell 中，通常可以用“fg”或者“%emacs”命令再

次回到 Emacs 中

C-x 字符扩展。 C-x 之后输入另一个字符或者组合键。

M-x 命令名扩展。M-x 之后输入一个命令名

M-x recover file<Return> 来恢复你的自动保存文件

用 C-h m 可以查看当前主模式的文档

用 C-u C-v 将本行带到屏幕的最上方

M-x auto fill mode<Return> 启动自动折行

1. 搜索

|  |  |  |
| --- | --- | --- |
|  | **快捷键** | **功能** |
|  | C-g | 终止搜索 |
|  | C-s | 是向前搜索 |
| C-r | 是向后搜索 |
|  |  |
|  |  |
|  |  |  |
|  |  |

■■■■ Anaconda

1. 安装
2. 用普通用户安装，不要sudo
3. 默认安装到：/home/mg/anaconda2
4. 可以：

echo ‘export PATH=”~/anaconda2/bin:$PATH”’>>~/.bashrc

source ~/.bashrc

1. 在 /etc/profile 加入：

export PYTHONPATH=/home/mg/mgWorks/caffe-0.9999/python:$PYTHONPATH

要用完整路径

1. 重启
2. 测试

python->import caffe

* protobuf

1. （单独安装的情况，可选）重新安装 protobuf

# make uninstall

# make clean

# ./configure --prefix=/usr

# make

# make check

# sudo make install

1. 进入 python 文件夹：

# python setup.py build (关键)

# python setup.py test

# python setup.py install

如果出现老是找不到 protobuf 的错误，直接

# sudo pip install protobuf

* （建议）Anaconda下安装protobuf

一定要做anaconda环境下装，即

# conda install protobuf，

如果提示You don't appear to have the necessary permissions to install packages，则

# sudo chown -R mg /home/mg/anaconda2/

# sudo chmod -R +x /home/mg/anaconda2/

然后再安装

安装 selective search，把它复制到 anaconda2/lib/python2.7/site-packages下面

* spyder桌面快捷方式

[Desktop Entry]

Encoding=UTF-8

Name=Spyder

Comment=Spyder

Exec=/home/mg/anaconda2/bin/spyder

Icon=/home/mg/anaconda2/share/pixmaps/spyder.png

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

* jupyter 快捷方式

[Desktop Entry]

Encoding=UTF-8

Name=jupyter-notebook

Comment=jupyter-notebook

Exec=/home/mg/anaconda2/bin/jupyter-notebook

Icon=/home/mg/anaconda2/pkgs/notebook-4.1.0-py27\_1/info/recipe/jupyter.png

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

1. 查询软件

anaconda search -t conda package-name=2.4.10

1. 安装软件

conda install theano package-name=0.7.0rc2

1. anaconda的删除

直接删除文件夹，# rm -rf anaconda2/

另外，删除/etc/profile 里面的变量

还要删除 .bashrc 里面的相关内容

1. 包管理

* 安装包：

conda install scipy

* 查看已安装的包：

conda list

* 查看指定环境已安装的包

conda list -n python34

* 查找包信息：

conda search numpy

* 安装：

conda install -n python34 numpy

* 更新包：

conda update -n python numpy

* 删除包：

conda remove -n python numpy

1. 更新

conda update conda

conda update anaconda

conda update python

1. 设置国内镜像

# 添加Anaconda的TUNA镜像

conda config --add channels <https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free>/

# 设置搜索时显示通道地址

conda config --set show\_channel\_urls yes

1. conda cheat

sheet: [http://conda.pydata.org/docs/\_downloads/conda-cheatsheet.pdf](http://conda.pydata.org/docs/_downloads/conda-cheatsheep.pdf)

1. anaconda错误

* 安装错误，权限问题

conda install XXX 出现如下错误：

Error: Missing write permissions in: /home/mg/anaconda2

# sudo chown -R mg /home/mg/anaconda2/

# sudo chmod -R +x /home/mg/anaconda2/

■■■■ Caffe

1. 装 gcc 4.7
2. 安装文件

# sudo apt-get install gcc-4.7 g++-4.7 g++-4.7-multilib gcc-4.7-multilib

1. 系统gcc降级

（其中数字越大，表示优先级越高）：

# sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.8 50

# sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.7 100

# sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 50

# sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.7 100

# sudo update-alternatives --install /usr/bin/cpp cpp-bin /usr/bin/cpp-4.8 50

# sudo update-alternatives --install /usr/bin/cpp cpp-bin /usr/bin/cpp-4.7 100

1. 验证版本

验证gcc-4.7是否安装并成为系统的默认版本：

# gcc -v

1. 安装Matlab

具体过程见 Matlab 有关章节

之后，修改 Caffe 的 Makefile.config

42 MATLAB\_DIR := /usr/local/MATLAB/R2014a

1. Caffe 依赖库

* (推荐)按照 caffe 网站推荐顺序：

# sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libhdf5-serial-dev protobuf-compiler

# sudo apt-get install --no-install-recommends libboost-all-dev

# sudo apt-get install libatlas-base-dev

# sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev

然后，安装 gflags：

# gflags： https://github.com/schuhschuh/gflags/archive/master.zip 下载。解压

# cd gflags-master

# mkdir build && cd build

# export CXXFLAGS="-fPIC" && cmake .. && make VERBOSE=1

# (sudo apt-get install cmake)

# make

# sudo make install

* 或者一次完成：

# sudo apt-get install libatlas-base-dev libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libboost-all-dev libhdf5-serial-dev libgflags-dev libgoogle-glog-dev liblmdb-dev protobuf-compiler

1. CUDA，(未安装，可参考)

# sudo dpkg -i cuda-repo-ubuntu1404-7-5-local\_7.5-18\_amd64.deb

# sudo apt-get update

# sudo apt-get install cuda

# sudo ldconfig /usr/local/cuda-7.5/lib64

1. CUDNN（似乎有问题，目前没法用）

注意于cuda大版本要一致，这里都用大7.5的版本。

解压后：

cd cudnn\_dir

sudo cp lib\* /usr/local/cuda/lib64/

sudo cp cudnn.h /usr/local/cuda/include/

复制文件：

sudo cp include/cudnn.h /usr/local/include/

sudo cp lib64/\* /usr/local/lib/

cd /usr/local/lib/

chmod +r libcudnn.so.\*

复制库：

sudo cp /usr/local/cuda-7.5/lib64/libcudart.so.7.5 /usr/local/lib/libcudart.so.7.5

sudo cp /usr/local/cuda-7.5/lib64/libcublas.so.7.5 /usr/local/lib/libcublas.so.7.5

sudo cp /usr/local/cuda-7.5/lib64/libcurand.so.7.5 /usr/local/lib/libcurand.so.7.5

sudo ldconfig

否则会出现错误：.build\_release/tools/caffe: error while loading shared libraries: libcudart.so.7.5: cannot open shared object file: No such file or directory

出现问题：cudnn\_conv\_layer.cpp:30] Check failed: status == CUDNN\_STATUS\_SUCCESS (6 vs. 0)  CUDNN\_STATUS\_ARCH\_MISMATCH

是因为GPU加速性能不够，**别用了**。

注意：可以不用cuDNN，这样只要在Makefile里不选择就可以。

1. 安装Anaconda

* 直接安装
* （不推荐）单独安装 python 环境

1. 安装依赖软件

# sudo apt-get install python-numpy python-scipy python-matplotlib python-sklearn python-skimage python-h5py python-protobuf python-leveldb python-networkx python-nose python-pandas python-gflags Cython ipython

# sudo apt-get install protobuf-c-compiler protobuf-compiler

1. pycaffe

# cd ~/caffe$ make pycaffe

环境变量

# sudo gedit /etc/profile

末尾添加：

export PYTHONPATH=/path/to/caffe/python:$PYTHONPATH

要用完整路径

1. 重启
2. 测试

python->import caffe

* spyder（会自动安装依赖）

（关键）首先要在 Ubuntu 软件中心卸载 ipython，否则会出现ipython和spyder的启动问题

# sudo apt-get install spyder

不要用 sudo pip install spyder，因为里面没有ipython

快捷方式：

[Desktop Entry]

Encoding=UTF-8

Name=Spyder

Comment=Spyder

Exec=/usr/bin/spyder

Icon=/usr/share/pixmaps/spyder.png

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

* jupyter notebook

（可以不要，不好用）# sudo apt-get install ipython-notebook

# sudo pip install jupyter

[Desktop Entry]

Encoding=UTF-8

Name=jupyter-notebook

Comment=jupyter-notebook

Exec=/usr/local/bin/jupyter-notebook

Icon=/usr/local/lib/python2.7/dist-packages/qtconsole/resources/icon/JupyterConsole.svg

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

* jupyter notebook 快捷方式：
* spyder:

安装依赖文件: # sudo apt-get install python-qt4 python-sphinx

安装: # sudo pip install spyder

升级: # sudo pip install -U spyder

1. 安装protobuf

最好字anaconda下安装

1. 安装 caffe主体
2. 下载

caffe-0.9999( code brew)（针对R-CNN）

目前装到是caffe-master，最新版本

1. 复制Makefile

# cp Makefile.config.example Makefile.config

1. 修改 Makefile.config

8 CPU\_ONLY := 1

42 MATLAB\_DIR := /usr/local/MATLAB/R2014a

68 ANACONDA\_HOME := $(HOME)/anaconda2

69 PYTHON\_INCLUDE := $(ANACONDA\_HOME)/include \

$(ANACONDA\_HOME)/include/python2.7 \

$(ANACONDA\_HOME)/lib/python2.7/site-packages/numpy/core/include \

1. 编译

# make -j 4 all

# make -j 4 matcaffe

# make –j 4 test

# make runtest

1. pycaffe
2. 安装必须的软件：

* easy\_install：

sudo apt-get install python-setuptools python-dev build-essential

* pip：

sudo easy\_install pip

1. 安装 pycaffe

进入caffe-master文件夹，然后：

1. 修改 caffe 的 Makefile.config:

54 PYTHON\_LIB := $(HOME)/mg/anaconda2/lib

68 ANACONDA\_HOME := /home/mg/anaconda2

69 PYTHON\_INCLUDE := $(ANACONDA\_HOME)/include \

70 $(ANACONDA\_HOME)/include/python2.7 \

71 ANACONDA\_HOME)/lib/python2.7/site-packages/numpy/core/include \

87 WITH\_PYTHON\_LAYER := 1

1. 编译

# make pycaffe

# make distribute

1. 执行完后修改~/.bashrc文件，添加

export PYTHONPATH=/home/mg/mg\_works/caffe-master/python:/home/mg/mg\_works/caffe-master/distribute/python:$PYTHONPATH

export LD\_LIBRARY\_PATH=/home/mg/mg\_works/caffe-master/distribute/lib:/home/mg/mg\_works/caffe-master/build/lib:$LD\_LIBRARY\_PATH

注意：有可能出现在python环境下import caffe时出现错误：找不到 libcaffe.so.1.0.0-rc3，则可以通过以上配置解决。关键是 export PYTHONPATH=/home/mg/mg\_works/caffe-master/python，一句。但是，在spyder里面不会有问题，可见spyder里已经加入了相关路径。

1. 测试

# make pytest (一般要重装 protobuf)

1. 可视化网络结构（必须在python 之后）
2. 安装 easy\_install：

# sudo apt-get install python-setuptools python-dev build-essential

1. 安装 pip：

# sudo easy\_install pip

1. 安装GraphViz

# sudo apt-get install GraphViz

1. 安装pydot

# sudo pip install pydot

在anaconda里安装 # conda install pydot

1. 测试：

# python python/draw\_net.py examples/mnist/lenet\_train\_test.prototxt mg-lenet.png --rankdir=BT

1. Caffe 各种的库的下载与安装

目前，可以用自己写大小脚本来创建。

* mnist

复制四个文件到 caffe-0.9999/data/mnist，

修改get\_mnist.sh，

注释掉wget下载内容，然后./data/mnist/get\_mnist.sh。

在caffe-master/examples/mnist下，./create\_mnist.sh

修改 lenet\_solver.prototxt，修改最后一行为使用CPU

在caffe-master/examples/mnist下，./train\_lenet.sh

* Imagenet

复制 caffe\_ilsvrc12.tar.gz 到 caffe/data/ilsvrc12 下，修改get\_ilsvrc\_aux.sh 并执行

把caffe\_reference\_imagenet\_model.caffemodel 和 caffe\_rcnn\_imagenet\_model.caffemodel 复制到 caffe/examples/imagenet 下，并去掉后缀名

* cifar10

复制cifar-10-binary.tar到caffe-0.9999/data/cifar10,

注释掉wget，

./get\_cifar10.sh

* 其他（新版本有）

在 caffe-master 目录下scripts/download\_model\_binary.py <dirname>，其中，<dirname>是 models/ 目录下的目录

1. 安装 R-CNN（已废弃）
2. 创建链接

# ln -sf ~/mgWorks/caffe-0.9999 external/caffe

1. 安装 Selective Search

解压SelectiveSearchCodeIJCV.zip

# mv SelectiveSearchCodeIJCV rcnn/selective\_search/

1. 配置Matlab

打开 Matlab，进入 rcnn，运行startup()

# rcnn\_build()

# key=caffe('get\_init\_key')

1. 复制数据：

进入rcnn/data:

复制r-cnn-release1-data.tgz

在/rcnn/data文件夹下：tar zxvf r-cnn-release1-data.tgz

此时，可以跑了：rcnn\_demo('PASCAL',false)，5分钟左右

之后，# tar zxvf r-cnn-release1-selective-search.tgz

1. 安装 Selective Search

解压selective\_search\_ijcv\_with\_python-master.zip，改名为selective\_search\_ijcv\_with\_python

打开matlab，运行demo

加入 python 的搜索路径，可以直接复制到 /usr/local/lib/python2.7/dist-packages

如果在Anaconda下安装 selective search，则把它复制到 anaconda2/lib/python2.7/site-packages下面

1. 接口学习

* super:

super(type[,object-or-type])

Return the superclass of type.

super不是函数，是一个类名，形如super(B，self)事实上调用了super类的初始化函数，产生了一个super对象

super类的初始化函数并没有做特殊操作，只是简单记录了类类型和具体实例。

super(B,self).func()的调用并不是用于调用当前类的父类的func函数

super要求父类继承object

* OrderedDict

from collections import OrderedDict, Counter

OrderedDict

1. 使用

1.必须在 caffe 的根目录下运行，否则出错

2.绘制网络模型图：

# python python/draw\_net.py examples/mnist/lenet\_train\_test.prototxt lenet.png --rankdir=BT

1. 笔记

Add the module directory to your $PYTHONPATH byexport PYTHONPATH=/path/to/caffe/python:$PYTHONPATH or the like for import caffe

* 见1，blobs,layers and nets:

1. Blob:

Mathematically, a blob is an N-dimensional array stored in a C-contiguous fashion

The conventional blob dimensions for batches of image data are batch number N x channel K x height H x width W. Blob memory is row-major in layout, so the last / rightmost dimension changes fastest.

* N \* K \* H \* W

the value at index (n, k, h, w) is physically located at index:

* ((n \* K + k) \* H + h) \* W + w

a Blob stores two chunks of memories, data and diff.

there are two different ways to access the actual values: the const way and the mutable way

1. Layer:

Each layer type defines three critical computations: setup, forward, and backward

* 具体定义见5，Layer Catalogue

1. Net:

The net is a set of layers connected in a computation graph – a directed acyclic graph (DAG) to be exact

* 见2，Forward and Backward

Net::Forward() and Net::Backward() : carry out the respective passes

Layer::Forward() and Layer::Backward() : compute each step

* 见3，Loss

any layer can be used as a loss by adding a field loss\_weight: <float> to a layer definition for each top blob produced by the layer. Layers with the suffix Loss have an implicit loss\_weight: 1 for the first top blob (and loss\_weight: 0 for any additional tops); other layers have an implicit loss\_weight: 0 for all tops.

any layer able to backpropagate may be given a non-zero loss\_weight

1. Solver：

* 见4，Solver

The responsibilities of learning are divided between the Solver for overseeing the optimization and generating parameter updates and the Net for yielding loss and gradients

Stochastic Gradient Descent (type: "SGD"),

AdaDelta (type: "AdaDelta"),

Adaptive Gradient (type: "AdaGrad"),

Adam (type: "Adam"),

Nesterov’s Accelerated Gradient (type: "Nesterov") and

RMSprop (type: "RMSProp")

本文包含了参数调节的重要原则



learning rate 

momentum 

training, so if you increase , it may be a good idea to decrease accordingly (and vice versa)

Updating Parameters：

Solver::ComputeUpdateValue()，Finally, the Blob::Update method is called on each parameter blob, which performs the final update(subtracting the Blob’s diff from its data).

Snapshotting and Resuming：

Solver::Snapshot() and Solver::SnapshotSolverState()

Solver::Restore() and Solver::RestoreSolverState().

Weights are saved without extension while solver states are saved with .solverstate extension. Both files will have an \_iter\_N suffix for the snapshot iteration number

1. 接口

* caffe.Net is the central interface for loading, configuring, and running models. caffe.Classifier and caffe.Detector provide convenience interfaces for common tasks.
* caffe.SGDSolver exposes the solving interface.
* caffe.io handles input / output with preprocessing and protocol buffers.
* caffe.draw visualizes network architectures.
* Caffe blobs are exposed as numpy ndarrays for ease-of-use and efficiency

1. 源码

caffe-master/python/caffe/net\_spec.py:

■■■■ Faster-rcnn

可参考 《 Faster R-CNN安装笔记，只用CPU.pdf》

1. 下载

# git clone --recursive https://github.com/rbgirshick/py-faster-rcnn.git

1. 修改 setup.py

进入lib目录，修改setup.py，注释掉GPU相关代码

58 # CUDA = locate\_cuda()

90 # self.set\_executable('compiler\_so', CUDA['nvcc'])

125 # Extension('nms.gpu\_nms',

126 # ['nms/nms\_kernel.cu', 'nms/gpu\_nms.pyx'],

127 # library\_dirs=[CUDA['lib64']],

128 # libraries=['cudart'],

129 # language='c++',

130 # runtime\_library\_dirs=[CUDA['lib64']],

131# # this syntax is specific to this build system

132 # # we're only going to use certain compiler args with nvcc and not with

133 # # gcc the implementation of this trick is in customize\_compiler() below

134 # extra\_compile\_args={'gcc': ["-Wno-unused-function"],

135 # 'nvcc': ['-arch=sm\_35',

136 # '--ptxas-options=-v',

137 # '-c',

138 # '--compiler-options',

139 # "'-fPIC'"]},

140 # include\_dirs = [numpy\_include, CUDA['include']]

141 # ),

1. lib 的make

在 py-faster-rcnn/lib 文件夹下，

# make

1. 修改 Makefile

进入 caffe-fast-rcnn 文件夹，修改 Makefile.config

8 CPU\_ONLY := 1

59 MATLAB\_DIR := /usr/local/MATLAB/R2014a/

69 ANACONDA\_HOME := $(HOME)/anaconda2

70 PYTHON\_INCLUDE := $(ANACONDA\_HOME)/include \

71 $(ANACONDA\_HOME)/include/python2.7 \

72 $(ANACONDA\_HOME)/lib/python2.7/site-packages/numpy/core/include \

89 WITH\_PYTHON\_LAYER := 1

1. 整体make

在 py-faster-rcnn/caffe-fast-rcnn 目录下：

# make -j 4

# make pycaffe

1. 安装 easydict

# pip install easydict

1. 安装opencv

# conda install -c https://conda.binstar.org/menpo opencv

可以：# conda install -c https://conda.binstar.org/menpo opencv=2.4.10

1. 修改nms\_wrapper.py

在 py-faster-rcnn/lib/fast\_rcnn 目录下，修改 nms\_wrapper.py：

9 # from nms.gpu\_nms import gpu\_nms

12 def nms(dets, thresh, force\_cpu=True):

1. 准备 model

出现错误：

IOError: /home/mg/mg\_works/py-faster-rcnn/data/faster\_rcnn\_models/VGG16\_faster\_rcnn\_final.caffemodel not found.

Did you run ./data/script/fetch\_faster\_rcnn\_models.sh?

把 faster\_rcnn\_models.tgz 复制到 py-faster-rcnn/data/scripts 文件夹下

# tar zxvf faster\_rcnn\_models.tgz

把文件夹 faster\_rcnn\_models 放到 py-faster-rcnn/data 下。

1. 运行 demo

在 py-faster-rcnn 目录下：

# ./tools/demo.py --cpu

■■■■ scikit-learn

可参考 《 Faster R-CNN安装笔记，只用CPU.pdf》

1. MKL问题

Intel MKL FATAL ERROR: Cannot load libmkl\_avx.so or libmkl\_def.so.

# conda install nomkl numpy scipy scikit-learn numexpr

# conda remove mkl mkl-service

去除对MKL的依赖

1. 随机抓取分配数据集：

from sklearn import cross\_validation

**cross\_validation**.train\_test\_split(X,y,test\_size=0.2,random\_state=42

1. 数据预处理

preprocessing.StandardScaler()

■■■■ Tensorflow

1. 安装

* 成功

1. 二进制文件安装

结论：protobuf要使用pip安装，然后再装tensorflow。

# pip uninstall protobuf

# pip install --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.8.0-cp27-none-linux\_x86\_64.whl

1. 源码安装（太慢，下载不了）

git clone --recurse-submodules https://github.com/tensorflow/tensorflow

* （有问题）安装

1. 纯环境

# sudo pip install --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.8.0-cp27-none-linux\_x86\_64.whl

* Anaconda
* Ubuntu下

Create a conda environment called tensorflow:

# conda create -n tensorflow python=2.7

* Anaconda 的虚拟环境下

Activate the environment and use pip to install TensorFlow inside it. Use the --ignore-installed flag to prevent errors about easy\_install.

# source activate tensorflow

CPU Only：

# pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.8.0-cp27-none-linux\_x86\_64.whl

之后：

# source deactivate

之后，每次使用时：

To use TensorFlow later you will have to activate the conda environment again:

$ source activate tensorflow

(tensorflow)$ # Your prompt should change.

# Run Python programs that use TensorFlow.

...

# When you are done using TensorFlow, deactivate the environment.

(tensorflow)$ source deactivate

* mg推荐：Anaconda下直接安装（总是连接不上）

# conda install -c https://conda.anaconda.org/jjhelmus tensorflow

* mg终极大法

把 tensorflow-0.8.0rc0-cp27-none-linux\_x86\_64.whl 和 protobuf-2.5.0-py2.7.egg 复制到 /home/mg/anaconda2/lib/python2.7/site-packages 目录下

切换到此目录

直接

# (不用sudo)pip install tensorflow-0.8.0rc0-cp27-none-linux\_x86\_64.whl

测试 OK

import tensorflow as tf

hello=tf.constant('mg hello')

sess=tf.Session()

print(sess.run(hello))

a=tf.constant(10)

b=tf.constant(32)

print(sess.run(a+b))

1. saver保存

注意几点：1.保存的环境要完全一样，所以最好采用以下的方式，区分train和test就行。

2.save的时候，最好写明保存的内容，现在发现只有这样才可以restore

3.save的时候需要完整路径名和文件名，restore的时候只要路径

#bTrain = True

bTrain = False

# y = ax

x = tf.placeholder("float")

a = tf.Variable(0.0,"float")

y = tf.mul(a, x)

sess = tf.Session() # 创建会话，计算符号变量表达式

mgSaver = tf.train.Saver({'a':a})

sess.run(tf.initialize\_all\_variables())

if bTrain:

sess.run(a.assign(3.0))

mgSaver.save(sess,'./tmp/mg.tfmodel')

print '====mg:saved!'

else:

ckpt = tf.train.get\_checkpoint\_state('./tmp')

if ckpt:

mgSaver.restore(sess,ckpt.model\_checkpoint\_path)

print '====mg:loaded!'

for i in range(10):

print "%d:%f" % (i,sess.run(y, feed\_dict={x: i}))

else:

print 'not load!'

1. 语法
2. ops.run()，tensor.eval()
3. TensorBoard
4. 关于histogram

显示百分位数

* 把0.5版本装在anaconda的虚拟环境下

# conda create -n mg-tb python=2.7

# source activate mg-tb

进入 /home/mg/anaconda2/envs/mg-tb/lib/python2.7/site-packages，复制tensorflow-0.5.0-cp27-none-linux\_x86\_64.whl

终端里，进入此目录

# pip install tensorflow-0.5.0-cp27-none-linux\_x86\_64.whl

然后切换到其他路径查看就可以了。

使用完毕关闭虚拟环境：

# source deactivate

* tensorboard显示空白的问题解决

其实不是问题，是响应慢。安装 chrome会快些。

* ImportError: cannot import name symbol\_database：

安装高版本protobuf

# pip uninstall protobuf

# pip uninstall tensorflow

# brew uninstall protobuf

# pip install https://storage.googleapis.com/tensorflow/mac/tensorflow-0.5.0-py2-none-any.whl

* merge summary的错误

InvalidArgumentError: You must feed a value for placeholder tensor 'Placeholder\_1' with dtype float

牛逼回复：

From your error message, it looks like you are using IPython. One pitfall when using IPython to build a TensorFlow model is that functions like tf.merge\_all\_summaries() will remember every summary created in the current session, including cells that failed with an error. This is a result of TensorFlow using a default graph to collect all of the operations, summaries, etc. that are created in a process, unless you specify the graph explicitly. I suspect that your call to tf.merge\_all\_summaries() is returning more than the three histogram summaries that you created in your code, and the older ones will have a dependency on a previously created placeholder.

There are two main ways that you could fix that. The simplest is to explicitly merge the summaries, rather than using tf.merge\_all\_summaries():

weights\_summary = tf.histogram\_summary("weights", W)

biases\_summary = tf.histogram\_summary("biases", b)

y\_summary = tf.histogram\_summary("y", y)

merged = tf.merge\_summary([weights\_summary, biases\_summary, y\_summary])

An alternative would be to set an explicit default graph before constructing your model. This is awkward if you want to split your model across multiple IPython cells, but should also work:

# Sets a new default graph, and stores it in `g`.

with tf.Graph().as\_default() as g:

x = tf.placeholder(tf.float32, [None, 784])

# ...

# Summaries are added to `g`.

\_ = tf.histogram\_summary("weights", W)

\_ = tf.histogram\_summary("biases", b)

\_ = tf.histogram\_summary("y", y)

# ...

# `merged` contains only summaries from `g`.

merged = tf.merge\_all\_summaries()

# ...

**关键是：用ipython的时候，不能使用cell的模式，要一次运行。**

* 关于出现图叠加的问题

猜测可能由于：当函数被多次调用时，默认的图被从上一次调用时留下的未使用的OPS或者Variables挤满，

猜测解决方法：

重新创建图，每次都使用单独的图：

# graph = tf.Graph()

# with graph.as\_default():

# ……..

1. TDB

* 安装

# pip install tfdebugger

或者进入安装文件夹，使用python setup.py install的方式，但是这样无法进行函数的有效跳转，可以直接把egg包里面的tdb的文件夹复制到/home/mg/anaconda2/lib/python2.7/site-packages

jupyter的支持，运行以下文本：

import notebook.nbextensions

import urllib

import zipfile

#SOURCE\_URL = 'https://github.com/ericjang/tdb/releases/download/tdb\_ext\_v0.1/tdb\_ext.zip'

#urllib.urlretrieve(SOURCE\_URL, 'tdb\_ext.zip')

with zipfile.ZipFile('tdb\_ext.zip', "r") as z:

z.extractall("")

notebook.nbextensions.install\_nbextension('tdb\_ext',user=True)

* 关于plot为空

使用sleep()的方法不能有效解决

目前有效方法：plt.ginput(timeout=0.1)

■■■■ Theano

1. 安装
2. anaconda里面安装

conda install theano

■■■■ Python

1. 环境交互
2. 只执行一句语句

python –c “command”

1. 运行shell命令

多种方法，推荐的有：

* 方法一：

import subprocess

subprocess.call([“ls”,”-l”])

或subprocess.call(“ls -l”)

subprocess.call(“ls -l”,shell=True)表示最终在shell中运行

* 方法二：

import commands

commands.getoutput(“ls -l”)

还有getstatus和getstatusoutput两种选择

* 方法三：

os.system(“ls -l”)

1. 查看内建命令

dir(\_\_builtin\_\_)

1. 查看相关内容

s = ‘abc’

dir(s)

help(s.replace)

1. 产生通配符列表

>>>import glob

>>>glob.glob(’\*.py’)

[‘prime.py’,’random.py’,’bbc.py’]

1. 退出脚本

sys.exit()

1. sleep

import time

time.sleep()

1. IO

* str()

返回人类可读的表示形式

* repr()

生成可由解释器读取的表现形式

1. 命令行参数

import argparse

parser = argparse.ArgumentParser(description='mg Test ArgumentParser')

parser.add\_argument('def\_p0',help='mg p0') # 位置参数

parser.add\_argument('p1',help='mg p1') # 位置参数

parser.add\_argument('--p2',help='mg p2') # 可选参数

parser.add\_argument('p3',default='def p3',help='mg p3') # 位置参数

args = parser.parse\_args()

print 'p1:',args.p1

print 'p2:',args.p2

print 'p3:',args.p3

调用：

* IPython里：

>>>run test\_parser.py t1 t2 t3

p1: t2

p2: None

p3: t3

* 终端里：

# python test\_parser.py t1 t2 t3

1. 编译与生成
2. 生成单个pyc文件

直接在idle中

>>>import py\_compile

>>>py\_compile.compile(r'H:/game/test.py')

1. 批量生成pyc文件

>>>import compileall

>>>compileall.compile\_dir(r'H:/game')

1. 预定义名称

|  |  |
| --- | --- |
| **名称** | **功能** |
| \_\_name\_\_ | 文件名 |
| \_\_all\_\_ | 在\_\_init\_\_.py中定义应该被导入的模块名称列表，如：  \_\_all\_\_=[“echo”,”sound”,”reverse”] |
|  |  |
|  |  |
|  |  |

1. 路径
2. 执行代码的文件路径

os.path.split(os.path.realpath(\_\_file\_\_))[0]

1. 被初始执行的脚本的所在目录

sys.path[0]或sys.argv[0]

1. 起始执行目录

os.getcwd()

1. 导入
2. 调用包里的函数

import sys

sys.path.append('/home/mg/mg\_codes/tf')

import mg\_open\_tensorboard

mg\_open\_tensorboard.open\_tb('tf\_logs')

1. 相同目录导入

直接import b(b.py)

1. 不同目录导入

先sys.path.append('c:\xxxx\b.py') 再import b

建议路径写为：

sys.path.append('c:\\xxx\\b.py') 或sys.path.append('c:/xxxx/b.py')

1. 重新导入

import imp;

imp.reload(mname)

1. 包内导入

绝对导入:

from a.b.c import echo

相对导入:

from . import echo

from .. import formats

from ..filters import equalizer

1. 得到当前目录

os.getcwd()

1. 文件
2. 保证文件夹存在

import os

if not os.path.exists(\_dir):

os.makedirs(\_dir)

1. 模块
2. 自定义模块（用于import）

一般来说，我们会将自己写的python模块与python自带的模块分开存放以达到便于维护的目的。Python 运行环境在查找库文件时是对 sys.path 列表进行遍历，如果我们想在运行环境中注册新的类库，主要有以下三种方法：

（1）在sys.path列表中添加新的路径。

（2）设置PYTHONPATH环境变量。

（3）将库文件复制到sys.path列表中的目录里（如site-packages目录）。

其实，最简单的办法是用 .pth 文件来实现。在Python/Lib/site-package目录下创建一个MyModule.pth文件，其中内容为自定义的模块所在的路径。

1. 安装模块

# pip install 模块名

1. 关于使用自定义函数

首先，在/home/mg/anaconda2/lib/python2.7/site-packages下面（**必须在此**）加入.pth文件，里面写入自定义文件夹路径，不加引号，如：

/mnt/hgfs/mg\_sharedFolder/codes/\_mg\_py\_tools

之后，在~/.bashrc里面加入一个自定义变量，指向自定义文件夹，如：

MG=/mnt/hgfs/mg\_sharedFolder/codes/\_mg\_py\_tools

每次使用时，可以python $MG/…

1. 模块引入时是执行的，因此可以看到print的结果
2. 重新引入模块

reload

from imp import reload

reload(xxx)

1. exec

从交互提示模式启动文件二不必导入以及随后的重载。每次都运行文件的最新版本。相当于把文件内容复制过来，不用考虑重载。

但问题是，可能存在变量名被覆盖的情况。

在2.6 里面有个等效的函数:execfile(‘module.py’)

>>> x = 999

>>> exec(open(‘module.py’).read())

...do sth ...

>>> x

‘spam!’

1. 重要模块

|  |  |
| --- | --- |
| 名称 | 功能 |
| os | 操作系统接口，如system(),getcwd(),chdir() |
| shutil | 日常文件和目录管理，如copyfile(),move() |
| glob | 文件通配符，如glob(), |
| sys | 命令行参数，如sys.argv(), sys.stderr.write(), |
| re | 字符串模式匹配，如findall(), sub() |
| math | 数学，如cos(), log(), |
| random | 随即操作，如choice(), sample(), random(), |
| urllib | 互联网，如urlopen() |
| smtplib | 邮件，如SMTP() |
| datetime | 日期时间，如today(), strftime(), |
| zlib | 数据压缩，如compress(), decompress(), crc32() |
| timeit | 时间性能评价，如Timer(‘a,b=b,a’).timeit() |
| reprlib | 格式化输出，如repr(), |
| pprint | 精细输出 |
| textwrap | 折行，如print(textwrap.fill(doc, width=40)) |
| logging | log文件,把日志发送到 sys.stderr，如logging.debug(‘xx’), info(xxxx), warning(), error(), critical() |
|  |  |
|  |  |

1. 字符串
2. 字符串格式两种：

* >>> '{0},eggs,and {1}'.format('aaa','bbb')

'aaa,eggs,and bbb'

* >>> '%s,eggs,and %s' %('aa','bb')

'aa,eggs,and bb'

1. 原始字符串常量

使用 r 开头

1. raw字符串无法添加最后一个反斜杠

解决方法：

使用分片：r'1\nb\tc\\'[:-1]

手动添加：r'1\nb\tc'+'\\'

1. 松散型

加上re.VERBOSE

re.search(pattern, 'M', re.VERBOSE)

1. 加速

先用list把字符串打散，处理后在用join连接

>>> s='spammy'

>>> L=list(s)

>>> L

['s', 'p', 'a', 'm', 'm', 'y']

>>> L[3]='x'

>>> L

['s', 'p', 'a', 'x', 'm', 'y']

>>> L[1:3] = 'y'

>>> L

['s', 'y', 'x', 'm', 'y']

>>> s=''.join(L)

>>> s

'syxmy'

1. 字符编码

* 到ASCII码

>>> ord('s')

* 根据ASCII码得到字符

>>> chr(115)

's'

1. 路径

|  |  |  |
| --- | --- | --- |
| 函数 | 功能 | 例 |
| os.path.**basename**() | 查询路径中包含的文件名 |  |
| os.path.**dirname**() | 查询路径中包含的目录 |  |
| os.path.**split**() | 将路径分割成文件名和目录两个部分，放在一个表中返回 | >>> (filepath, filename) =os.path.split(  "c:\\music\\ap\\mahadeva.mp3")  >>> filepath  'c:\\music\\ap'  >>> filename  'mahadeva.mp3' |
| os.path.**join**() | 使用目录名和文件名构成一个路径字符串 |  |
| os.path.**commonprefix**() | 查询多个路径的共同部分 | p\_list = [path, path2]  print(os.path  .commonprefix(p\_list)) |
| os.path.**splitext**() | 分割文件名和后缀名 | >>> (shortname, extension) = os.path.splitext(filename)  >>> shortname  'mahadeva'  >>> extension  '.mp3' |
| os.path.**exists**() | 查询文件是否存在 |  |
| os.path.**getsize**() | 查询文件大小 |  |
| os.path.**getatime**() | 查询文件上一次读取的时间 |  |
| os.path.**getmtime**() | 查询文件上一次修改的时间 |  |
| os.path.**isfile**() | 路径是否指向常规文件 |  |
| os.path.**isdir**() | 路径是否指向目录文件 |  |
| glob.**glob**() | 列出所有符合该表达式的文件 | >>>import glob  >>>print(  glob.glob('/home/vamei/\*')) |
| os.**mkdir**() |  |  |
| os.**rmdir**() |  |  |
| os.**listdir**() | 相当于$ls命令 |  |
| os.**remove**() |  |  |
| os.**rename**(src,dst) |  |  |
| os.**chmod**(path,mode) | 相当于$chmod命令 |  |
| os.**chown**(path,uid,gid) | 改变path所指向文件的拥有者和拥有组。相当于$chown命令 |  |
| os.**stat**() | 查看path所指向文件的附加信息，相当于$ls -l命令 |  |
| os.**symlink**(src,dst) | 为文件dst创建软链接，src为软链接文件的路径。相当于$ln -s命令 |  |
| os.**getcwd**() | 得到当前目录 |  |
| shutil.**copy**(src,dst) | 复制文件，相当于$cp |  |
| shutil.**move**(src,dst) | 移动文件，相当于$mv |  |
|  |  |  |

1. 遍历
2. 遍历字典

>>>knight={‘gallahd’:’the pure’,’robin’:’the brave’}

>>>for k,v in knight.items():

… print(k,v)

1. 遍历序列

>>>for i,v in enumerate([‘tic’,’tac’,’toe’]):

… print(i,v)

1. 遍历多个序列

>>>for q,a in zip(lista,listb):

… print(‘{0},{1}’.format(q,a))

1. 逆序遍历

>>>for i in reversed(range(1,10,2)):

… print(i)

1. 按顺序遍历序列

>>>basket=[‘apple’,’orange’,’apple’,’pear’]

>>>for f in sorted(set(basket)):

… print(f)

1. 遍历文件夹及文件

* 方法一：os.path.walk()：

import os

s = os.sep #根据unix或win，s为\或/

root = "d:" + s + "ll" + s #要遍历的目录

def func(args,dire,fis): #回调函数的定义

for f in fis:

fname = os.path.splitext(f) #分割文件名为名字和扩展名的二元组

new = fname[0] + 'b' + fname[1] #改名字

os.rename(os.path.join(dire,f),os.path.join(dire,new)) #重命名

os.path.walk(root,func,()) #遍历

这种方法在使用时有个问题，不能递归遍历下一层（这点我还不确定

* 方法二：os.walk()

os.walk(top, topdown=True, onerror=None, followlinks=False)

一般只使用第一个参数。（topdown指明遍历的顺序）该方法对于每个目录返回一个三元组，(dirpath, dirnames, filenames)。第一个是路径，第二个是路径下面的目录，第三个是路径下面的非目录（对于windows来说也就是文件）：

import os

s = os.sep

root = "d:" + s + "ll" + s

for rt, dirs, files in os.walk(root):

for f in files:

fname = os.path.splitext(f)

new = fname[0] + 'b' + fname[1]

os.rename(os.path.join(rt,f),os.path.join(rt,new))

这种方式可以递归遍历所有的文件

* 方法三：listdir

import os

s = os.sep

root = "d:" + s + "ll" + s

for i in os.listdir(root):

if os.path.isfile(os.path.join(root,i)):

print i

这里需要注意的是，其中的i是目录或文件名，不是完整的路径，在使用时要结合os.path.join()方法还原完整路径。

* 另外，还可以使用os.system(cmd)来调用shell里面的相关命令对文件进行处理，很好很强大。

1. 语法
2. 关键字

|  |  |  |
| --- | --- | --- |
| **语法** | **功能** | **例** |
| **None** | 占位符 | >>> x = None  >>> print(x)  None |
| **and** | 所有值都为true，那么and返回最后一个值；某个为false，则返回第一个false的值。 |  |
| **or** | 如果有一个为true，则返回该值；如果都为false，返回最后一个false。 |  |
| **and or** |  | processFunc = collapse and (lambda s: " ".join(s.split())) or (lambda s: s)  processFunc 现在是一个函数，但它到底是哪一个函数要取决于 collapse 变量。  如果 collapse 为真，processFunc(string) 将压缩空白；  否则 processFunc(string) 将返回未改变的参数。 |
| **pass** | 什么都不做 |  |
| **sys.getrefcount()** | 查看引用计数 | >>> import sys  >>> sys.getrefcount(1)  3068 |
| **del** | 明确地清除某变量，删除list的元素 | del a[5]  del m |
| **\** | 换行 |  |
| **;** | 一行多个语句 | print  >>> a=1,b=2  >>> print '%i,%f'%(a,b)  1,2.00000 |
| **//** | 整除，返回不大于结果的最大整数，为了区分/的浮点除法，要加入 | from \_\_future\_\_ import division |
| **is** | 判断两个实例是否是同一个，即是否是对同一个对象引用 | >>>a=1  >>>b=1  >>>id(a) == id(b)  True  >>>a is b  True |
| **isinstance** | 是一个实例 | >>>a=(1,2)  >>>print isinstance(a,tuple)  True  >>>b=’abc’  >>>print isinstance(a,**basestring**)  True |
| **xrange** |  |  |
|  |  |  |
|  |  |  |

1. 函数

* 在Python中，函数也是一种对象。实际上，任何一个有\_\_call\_\_()特殊方法的对象都被当作是函数。

class SampleMore(object):

def \_\_call\_\_(self, a):

return a + 5

add = SampleMore() # A function object

print(add(2)) # Call function

map(add, [2, 4, 5]) # Pass around function object

* return后面没有返回值时，将返回None
* 默认值和关键字传递是两码事
* 包裹传递：

def func(\*name): #

print name

def func(\*\*dict): # dict

print dict

数量未知，当成一个

* 传参顺序：位置 -> 关键字 -> 包裹位置 -> 包裹关键字

1. 部分内部函数：

|  |  |  |
| --- | --- | --- |
| 语法 | 功能 | 例 |
| **None** | 占位符 | >>> x = None  >>> print(x)  None |
| **lambda表达式** | 生成一个函数对象 | func = lambda x,y: x + y  print func(3,4)  def test(f, a, b):  print 'test'  print f(a, b)  test((lambda x,y: x\*\*2 + y), 6, 9) |
| **enumerate()** | 返回索引和元素 |  |
| **zip()** | 每次循环时，从各个序列分别从左到右取出一个元素，合并成一个tuple | >>> ta=[1,2,3]  >>> tb=[9,8,7]  >>> zipped=zip(ta,tb)  >>> zipped  Out[36]: [(1, 9), (2, 8), (3, 7)]  >>> zip(\*zipped)  Out[45]: [(1, 2, 3), (9, 8, 7)] |
| **map()** | 有两个参数，一个是函数对象，一个是参数表。map()的功能是将函数对象依次作用于表的每一个元素 | re = map((lambda x: x+3),[1,3,5,6])  可以有多个参数：  re = map( (lambda x,y: x+y), [1,2,3], [6,7,9]) |
| **filter()** | 与map类似，函数返回True才保留 | def func(a):  if a > 100:  return True  else:  return False  print filter(func,[10,56,101,500])  out: [101, 500] |
| **reduce()** | 累进地将函数作用于各个参数 | print reduce( (lambda x,y: x+y), [1,2,5,7,9])  相当于(((1+2)+5)+7)+9 |
|  |  |  |
|  |  |  |

1. 类
2. 类属性和对象属性

class a(object):

attr = ‘abcd’ # 类属性

self.attr = ‘abcd’# 对象属性

1. 特性

特性是特殊的属性，是即时生成属性的方法。

property()最多可以加载四个参数。前三个参数为函数，分别用于处理查询特性、修改特性、删除特性。最后一个参数为特性的文档，可以为一个字符串，起说明作用。

class num(object):

def \_\_init\_\_(self, value):

self.value = value

def getNeg(self):

return -self.value

def setNeg(self, value):

self.value = -value

def delNeg(self):

print("value also deleted")

del self.value

neg = property(getNeg, setNeg, delNeg, "I'm negative")

x = num(1.1)

print(x.neg)

x.neg = -22

print(x.value)

print(num.neg.\_\_doc\_\_)

del x.neg

Out：-1.1

22

I'm negative

value also deleted

1. \_\_dict\_\_

对象的属性储存在对象的\_\_dict\_\_属性中。\_\_dict\_\_为一个词典，键为属性名，对应的值为属性本身。

下面两种属性修改方法等效：

summer.\_\_dict\_\_['age'] = 3

print(summer.\_\_dict\_\_['age'])

summer.age = 5

print(summer.age)

如果只有一个对象，而不知道它的类以及其他信息的时候，我们可以利用\_\_class\_\_属性找到对象的类，然后调用类的\_\_base\_\_属性来查询父类

class bird(object):

feather = True

class chicken(bird):

fly = False

def \_\_init\_\_(self, age):

self.age = age

summer = chicken(2)

>>>summer.\_\_dict\_\_

Out[79]: {'age': 2}

>>>summer.\_\_class\_\_

Out[80]: \_\_main\_\_.chicken

>>>summer.\_\_class\_\_.\_\_base\_\_

Out[81]: \_\_main\_\_.bird

\_\_dict\_\_分层存储属性。每一层的\_\_dict\_\_只存储该层新增的属性

1. \_\_getattr\_\_

当我们查询一个属性时，如果通过\_\_dict\_\_方法无法找到该属性，那么Python会调用对象的\_\_getattr\_\_方法，来即时生成该属。

Python中还有一个\_\_getattribute\_\_特殊方法，用于查询任意属性。\_\_getattr\_\_只能用来查询不在\_\_dict\_\_系统中的属性

1. 内置类属性

\_\_add\_\_() : +

\_\_sub\_\_() : -

属性相关：

\_\_setattr\_\_

\_\_getattr\_\_:

hasattr():

delattr():

isinstance():

issubclass():

实现类似于序列和映射的类，可进行下标操作：

\_\_setitem\_\_

\_\_getitem\_\_

\_\_delitem\_\_

\_\_len\_\_

1. 上下文管理器

with open("new.txt", "w") as f:

print(f.closed)

f.write("Hello World!")

print(f.closed)

当我们使用上下文管理器的语法时，我们实际上要求Python在进入程序块之前调用对象的\_\_enter\_\_()方法，在结束程序块的时候调用\_\_exit\_\_()方法。对于文件对象f来说，它定义了\_\_enter\_\_()和\_\_exit\_\_()方法(可以通过dir(f)看到)。在f的\_\_exit\_\_()方法中，有self.close()语句。

1. 任何定义了\_\_enter\_\_()和\_\_exit\_\_()方法的对象都可以用于上下文管理器

class VOW(object):

def \_\_init\_\_(self, text):

self.text = text

def \_\_enter\_\_(self):

self.text = "Added prefix: " + self.text # add prefix

return self # note: return an object

def \_\_exit\_\_(self,exc\_type,exc\_value,traceback):

self.text = self.text + " subfix" # add suffix

with VOW("I'm fine") as myvow:

print(myvow.text)

print(myvow.text)

Out: Added prefix: I'm fine

Added prefix: I'm fine subfix

1. 闭包（closure）

所谓的闭包是一个包含有环境变量取值的函数对象。环境变量取值被保存在函数对象的\_\_closure\_\_属性中。

def line\_conf():

b = 15

def line(x):

return 2\*x+b

return line # return a function object

b = 5

my\_line = line\_conf()

print(my\_line.\_\_closure\_\_)

print(my\_line.\_\_closure\_\_[0].cell\_contents)

Out： (<cell at 0x7f51a0d7b088: int object at 0xb78008>,)

15

例：

def line\_conf(a, b):

def line(x):

return a\*x + b

return line

line1 = line\_conf(1, 1)

line2 = line\_conf(4, 5)

print(line1(5), line2(5))

Out：(6, 25)

函数line与环境变量a,b构成闭包。

利用闭包，我们实际上创建了泛函。

1. 装饰器（decorator）
2. 函数装饰器

相当于func = decorator(func)，然后在func(params)

1. 含参装饰器（注意层数）

def pre\_str(pre=''):

# old decorator

def decorator(F):

def new\_F(a, b):

print(pre + "input", a, b)

return F(a, b)

return new\_F

return decorator

# get square sum

@pre\_str('^\_^')

def square\_sum(a, b):

return a\*\*2 + b\*\*2

# get square diff

@pre\_str('T\_T')

def square\_diff(a, b):

return a\*\*2 - b\*\*2

print(square\_sum(3, 4))

print(square\_diff(3, 4))

Out：

('^\_^input', 3, 4)

25

('T\_Tinput', 3, 4)

-7

1. 类装饰器

def decorator(aClass):

class newClass:

def \_\_init\_\_(self, age):

self.total\_display = 0

self.wrapped = aClass(age)

def display(self):

self.total\_display += 1

print("total display", self.total\_display)

self.wrapped.display()

return newClass

@decorator

class Bird:

def \_\_init\_\_(self, age):

self.age = age

def display(self):

print("My age is",self.age)

eagleLord = Bird(5)

for i in range(3):

eagleLord.display()

Out：

('total display', 1)

('My age is', 5)

('total display', 2)

('My age is', 5)

('total display', 3)

('My age is', 5)

1. 正则表达式

* import re
* 函数

|  |  |
| --- | --- |
| 语法 | 功能 |
| **m = re.search(pattern, string)** | 搜索整个字符串，直到发现符合的子字符串  对于返回的m, 我们使用**m.group()**来调用结果 |
| **m = re.match(pattern, string)** | 从头开始检查字符串是否符合正则表达式。必须从字符串的第一个字符开始就相符 |
| **str = re.sub(pattern, replacement, string)** | 在string中利用正则变换pattern进行搜索，对于搜索到的字符串，用另一字符串replacement替换。返回替换后的字符串 |
| **re.split()** | 根据正则表达式分割字符串， 将分割后的所有子字符串放在一个表(list)中返回 |
| **re.findall()** | 根据正则表达式搜索字符串，将所有符合的子字符串放在一给表(list)中返回 |
| **re.compile()** |  |
|  |  |
|  |  |
|  |  |

* 语法

|  |  |  |
| --- | --- | --- |
|  | 语法 | 功能 |
| 单个字符 | . | 任意的一个字符 |
| a|b | 字符a或字符b |
| [afg] | a或者f或者g的一个字符 |
| [0-4] | 0-4范围内的一个字符 |
| [a-f] | a-f范围内的一个字符 |
| [^m] | 不是m的一个字符 |
| \s | 一个空格 |
| \S | 一个非空格 |
| \d | [0-9] |
| \D | [^0-9] |
| \w | [0-9a-zA-Z] |
| \W | [^0-9a-zA-Z] |
| 重复 | \* | 重复 >=0 次 |
| + | 重复 >=1 次 |
| ? | 重复 0或者1 次 |
| {m} | 重复m次。比如说 a{4}相当于aaaa，再比如说[1-3]{2}相当于[1-3][1-3] |
| {m, n} | 重复m到n次。比如说a{2, 5}表示a重复2到5次。小于m次的重复，或者大于n次的重复都不符合条件。 |
| 位置 | ^ | 字符串的起始位置 |
| $ | 字符串的结尾位置 |
|  |  |  |

1. 生成器(generator)

当生成器遇到一个yield时，会暂停运行生成器，返回yield后面的值。当再次调用生成器的时候，会从刚才暂停的地方继续运行，直到下一个yield。

def gen():

a=100

yield a

a = a\*8

yield a

yield 2000

for i in gen():

print i

结果为：

100

800

2000

1. 日期与时间

* import time
* 获取时间

print(time.time()) # wall clock time, unit: second

print(time.clock()) # processor clock time, unit: second

time.sleep(10) # sleep for 10 seconds

st = time.gmtime() # 返回struct\_time格式的UTC时间

st = time.localtime() # 返回struct\_time格式的当地时间, 当地时区根据系统环境决定。

s = time.mktime(st) # 将struct\_time格式转换成wall clock time

* 时间间隔：

import datetime

t = datetime.datetime(2012,9,3,21,30) # 是2012年9月3日21时30分

t\_next = datetime.datetime(2012,9,5,23,30)

delta1 = datetime.timedelta(seconds = 600)

delta2 = datetime.timedelta(weeks = 3)

print(t + delta1)

print(t + delta2)

print(t\_next - t)

* 两个datetime对象还可以进行比较

1. 错误
2. FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.

在比较None时，用is和is not，不要用 == 例如：

不要写成 if a == None 或 a != None，而应该用：

if val **is** None 或val **is not** None

1. 子进程

主要用于shell方式的操作。

通过使用subprocess包，我们可以运行外部程序。这极大的拓展了Python的功能。

1. 导入

import subprocess

1. 建立

* subprocess.call()

父进程等待子进程完成

* subprocess.check\_call()

父进程等待子进程完成

* subprocess.check\_output()

父进程等待子进程完成

import subprocess

out = subprocess.call("ls -l", shell=True)

out = subprocess.call("cd ..", shell=True)

* popen

Popen对象创建后，主程序不会自动等待子进程完成。我们必须调用对象的wait()方法，父进程才会等待

import subprocess

child = subprocess.Popen(["ping","-

c","5","www.google.com"])

child.wait() # 也可以不等待

print("parent process")

1. 子进程

child.poll() # 检查子进程状态

child.kill() # 终止子进程

child.send\_signal() # 向子进程发送信号

child.terminate() # 终止子进程

child.pid 子进程的PID存储

1. 文本流

child.stdin

child.stdout

child.stderr

1. 管道

利用subprocess.PIPE将多个子进程的输入和输出连接在一起，构成管道(pipe):

import subprocess

child1 = subprocess.Popen(["ls","-l"],stdout=subprocess.PIPE)

child2 = subprocess.Popen(["wc"],stdin=child1.stdout,stdout=subprocess.PIPE)

out = child2.communicate()

print(out)

1. 子进程输入

以利用communicate()方法来使用PIPE给子进程输入

import subprocess

child = subprocess.Popen(["cat"], stdin=subprocess.PIPE)

child.communicate("vamei")

1. 多进程

其用法与threading包中的同名类一致。所以，multiprocessing的很大一部份与threading使用同一套API，只不过换到了多进程的情境。

应优先考虑Pipe和Queue，避免使用Lock/Event/Semaphore/Condition等同步方式 (因为它们占据的不是用户进程的资源)。

1. 创建

import multiprocessing as mul

def f(x):

return x\*\*2

pool = mul.Pool(5)

rel = pool.map(f,[1,2,3,4,5,6,7,8,9,10])

print(rel)

我们创建了一个容许5个进程的进程池 (Process Pool) 。Pool运行的每个进程都执行f()函数。我们利用map()方法，将f()函数作用到表的每个元素上。

1. 共享内存

import multiprocessing

def f(n, a):

n.value = 3.14

a[0] = 5

num = multiprocessing.Value('d', 0.0)

arr = multiprocessing.Array('i', range(10))

p = multiprocessing.Process(target=f, args=(num, arr))

p.start()

p.join()

print num.value

print arr[:]

1. python编写的软件的安装

setup.py其实是python工具包distutils的配置文件，setuptools就是基于distutils来做的。egg文件实际上是个zip压缩文件

1. 安装setuptools
2. 自动：

$ sudo apt-get install python-setuptools

1. 手动：

$ wget http://peak.telecommunity.com/dist/ez\_setup.py

$ sudo python ez\_setup.py

$ sudo python ez\_setup.py -U setuptools

1. 或者：

$ wget http://pypi.python.org/packages/2.6/s/setuptools/setuptools-0.6c11-py2.6.egg

$ sudo sh setuptools-0.6c11-py2.6.egg

1. 制作

$ mkdir egg-demo

$ cd egg-demo

$ touch setup.py

$ ls

setup.py

1. 填写setup.py

#!/usr/bin/env python

#-\*- coding:utf-8 -\*-

from setuptools import setup,find\_packages

setup(

name = "demo",

version = "0.1.0",

packages = find\_packages(),

zip\_safe = False,

discription = "mg egg test demo",

long\_description = "mg first egg demo",

author = "mg",

author\_email = "mg@mg.com",

license = "GPL",

keywords = ("mg","Test","egg"),

platform = "Independant",

url = "",

)

1. 生成

$ python setup.py bdist\_egg

可以查看发生了什么：

$ ls -F

build/ dist/ setup.py UNKNOWN.egg-info/

可以看到多了三个文件夹。而在dist文件夹下，有一个egg文件：UNKNOWN-0.0.0-

py2.6.egg。 产蛋成功！

egg文件：

$ file dist/UNKNOWN-0.0.0-py2.6.egg

dist/UNKNOWN-0.0.0-py2.6.egg: Zip archive data, at least v2.0 to extract

1. 查看egg文件内容

$ unzip -l dist/demo-0.1.0-py2.6.egg

Archive: dist/demo-0.1.0-py2.6.egg

Length Date Time Name

--------- ---------- ----- ----

121 2010-12-06 17:30 demo/\_\_init\_\_.py

344 2010-12-06 17:46 demo/\_\_init\_\_.pyc

137 2010-12-06 17:46 EGG-INFO/SOURCES.txt

5 2010-12-06 17:46 EGG-INFO/top\_level.txt

1 2010-12-06 17:46 EGG-INFO/zip-safe

1 2010-12-06 17:46 EGG-INFO/dependency\_links.txt

227 2010-12-06 17:46 EGG-INFO/PKG-INFO

--------- -------

836 7 files

1. 安装

$ sudo python setup.py install

1. 导入使用

$ python -c "from demo import test;test()"

Hello, I'm amoblin.

1. 卸载

以python2.6版本为例，egg文件一般安装在/usr/local/lib/python2.6/distpackages/目录下，该目录下还有一个easy-install.pth文件，用于存放安装的egg信息。:

$ cd /usr/local/lib/python2.6/dist-packages

$ cat easy-install.pth|grep demo

./demo-0.1.0-py2.6.egg

$ ls -F|grep demo

demo-0.1.0-py2.6.egg/

卸载egg文件很简单，首先将包含此egg的行从easy-install.pth中删除，然后删除egg文件夹即可。

1. 单元测试uinittest

* 使用方法

unittest的基本使用方法：

1.import unittest

2.定义一个继承自unittest.TestCase的测试用例类

3.定义setUp和tearDown，在每个测试用例前后做一些辅助工作。

4.定义测试用例，名字以test开头。

5.一个测试用例应该只测试一个方面，测试目的和测试内容应很明确。主要是调用assertEqual、assertRaises等断言方法判断程序执行结果和预期值是否相符。

6.调用unittest.main()启动测试

7.如果测试未通过，会输出相应的错误提示。如果测试全部通过则不显示任何东西，这时可以添加-v参数显示详细信息。

两种测试方式：

1.unittest.main()来启动单元测试的测试模块；

2.添加到testsuite集合中再加载所有的被测试对象，而testsuit里存放的就是单元测试的用例。

* 命令行使用方法

# python -m unittest test\_pure\_tf

Anaconda里面要这样:

# !/home/mg/anaconda2/bin/python -m unittest test\_pure\_tf

* 例

mgCode.py

def sum( x, y):

return x+y

def sub( x, y):

return x-y

test\_mgCode.py

import unittest

import mgCode

class mytest(unittest.TestCase):

##初始化工作

def setUp(self):

pass

#退出清理工作

def tearDown(self):

pass

#具体的测试用例，一定要以test开头

def test\_sum(self):

self.assertEqual(mgCode.sum(1, 2), 2, 'test sum fail')

def test\_sub(self):

self.assertEqual(mgCode.sub(2, 1), 1, 'test sub fail')

if \_\_name\_\_ =='\_\_main\_\_':

unittest.main()

mgClass.py

class mgClass:

def \_\_init\_\_(self):

pass

def sum(self, x, y):

return x+y

def sub(self, x, y):

return x-y

test\_mgClass.py

import unittest

import mgClass

class mytest(unittest.TestCase):

##初始化工作

def setUp(self):

self.tclass = mgClass.mgClass() ##实例化了被测试模块中的类

#退出清理工作

def tearDown(self):

pass

#具体的测试用例，一定要以test开头

def testsum(self):

self.assertEqual(self.tclass.sum(1, 2), 3)

if \_\_name\_\_ =='\_\_main\_\_':

unittest.main()

1. 杂项

* 引用计数

需要注意的是，当使用某个引用作为参数，传递给getrefcount()时，参数实际上创建了一个临时的引用。因此，getrefcount()所得到的结果，会比期望的多1。

from sys import getrefcount

a = [1, 2, 3]

print(getrefcount(a))

out：2

* 垃圾回收

当Python运行时，会记录其中分配对象(object allocation)和取消分配对象(object deallocation)的次数。当两者的差值高于某个阈值时，垃圾回收才会启动。

* 可以通过gc模块的get\_threshold()方法，查看该阈值:

import gc

print(gc.get\_threshold())

Out：(700, 10, 10)

我们也可以手动启动垃圾回收，即使用gc.collect()

* 分代回收

Python将所有的对象分为0，1，2三代。所有的新建对象都是0代对象。当某一代对象经历过垃圾回收，依然存活，那么它就被归入下一代对象。垃圾回收启动时，一定会扫描所有的0代对象。如果0代经过一定次数垃圾回收，那么就启动对0代和1代的扫描清理。当1代也经历了一定次数的垃圾回收后，那么会启动对0，1，2，即对所有对象进行扫描。

get\_threshold()返回的(700, 10, 10)返回的两个10。也就是说，每10次0代垃圾回收，会配合1次1代的垃圾回收；而每10次1代的垃圾回收，才会有1次的2代垃圾回收。同样可以用set\_threshold()来调整。

import gc

gc.set\_threshold(700, 10, 5)

■■■■ Numpy

1. 语法

* len

numpy里面，len(a),得到a的第0维的个数，

1. 技巧
2. 由空矩阵逐渐生成二维矩阵

* 方法一：（每次都要重新复制，比较慢）

**首先，生成空矩阵：**

a = np.empty(shape=[0,2])

或 a = np.zeros((0,2))

或 x = [] ; x = np.array(x) 可以得到 array([], dtype=float64)

t = np.array([1,2])

**之后，依次叠加：**np.vstack((a,t))

X = np.append(X, [[i, j]], axis=0)

* 方法二：先创建，再逐个添加

1. 数组组合

np.hstack((只接受一个参数)), np.r\_[]

np.vstack((只接受一个参数)), np.v\_[]

concatenate((a,b,c), axis= x)

1. 数组分割

np.hsplit

np.vsplit

np.dsplit

np.split

1. 数组元素查找

* np.argwhere(a)：根据条件筛选，得到索引，索引shape与a相同

>>> a = np.array([1, 3, 4, 1, 6, 1, 7, 1])

>>> np.**argwhere**(a==1)

… array([[0],

[3],

[5],

[7]])

>>> a=np.reshape(a,[2,4])

… array([[1, 3, 4, 1],

[6, 1, 7, 1]])

>>> np.**argwhere**( a==1 )

… array([[0, 0],

[0, 3],

[1, 1],

[1, 3]])

* np.where(condition, [x, y])：根据条件在两个矩阵中筛选元素

给出 condition 按condition筛选，true选x元素，false选y元素；

没有给出condition 则找出不为0的

>>> np.**where**([[True, False],

[True, True]],

... [[1, 2], [3, 4]],

... [[9, 8], [7, 6]])

array([[1, 8],

[3, 4]])

>>> x = np.array( [[6,2,9], [1,5,2]] )

>>> np.**where**( x>5 )

… (array([0, 0]), array([0, 2]))

>>> x[ np.**where**( x>5 ) ]

… array([6, 9])

>>> np.**where**(x < 5, x, -1)

… array( [[-1, 2, -1],

[ 1, -1, 2]] )

* np.extract(codition,arr)：返回符合条件的元素
* np.argmin() np.argmax()：数组中，最小最大值的索引

1. 数组排序

np.msort(a)列排序

np.sort(a)行排序

np.argsort(a)排序后返回下标

■■■■ Matplotlib

1. 语法

gca ：得到当前axes

gcf ：得到当前figure

1. 技巧
2. 散点图

# plt.scatter(cens[:50,0],cens[:50,1],c='red')

# plt.scatter(cens[50:100,0],cens[50:100,1],c='green')

# plt.scatter(cens[100:,0],cens[100:,1],c='blue')

# plt.show()

■■■■ PIL

1. 技巧
2. 保存图像

img = np.array(255.0\*datas[i,:],dtype='uint8') # 类型转换

img = img.reshape(28,28)

img = Image.fromarray(img).convert('RGB') # 注意转为RGB

img.save(svpath)

1. 问题解决
2. IOError: cannot write mode P as JPEG

You need to convert the image to RGB mode.

Image.open('old.jpeg').convert('RGB').save('new.jpeg')

■■■■ Pandas

1. 读没有列标题的数据

把header设置为None：

df\_test = pd.read\_csv(file\_test, header = None, sep =' ')

1. 修改列标题

重新定义全部列标题

df.columns = ['pathname','label']

1. 删除一行

df = df.drop(n)

1. 增加内容相同的一列

df[‘new’] = value

1. 两列字符串合并变为新的一列

df[‘new’] = df.col1 + df.col2

■■■■ JAVA

1. 安装 jdk

* sudo mkdir /opt/java
* sudo tar zxvf jdk-8u102-linux-x64.tar.gz -C /opt/java/
* sudo vim /etc/profile，在最后，加入：

export JAVA\_HOME=/opt/java/jdk1.8.0\_102

export JRE\_HOME=${JAVA\_HOME}/jre

export CLASSPATH=.:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

export PATH=${JAVA\_HOME}/bin:$PATH

* 然后，source /etc/profile 使之生效
* 验证，java -version

1. 安装 eclipse

* tar zxvf ...tar.gz后，把文件copy到 /opt 下面后，
* vim ~/Desktop/eclipse.desktop

[Desktop Entry]

Encoding=UTF-8

Name=Eclipse Platform

Comment=Eclipse IDE

Exec=/opt/java/eclipse/eclipse

Icon=/opt/java/eclipse/icon.xpm

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

**关键：进入eclipse目录，**

# sudo mkdir jre

# cd jre

# sudo ln -s /opt/java/jdk1.8.0\_77/bin/

否则，会出现：

A Java Runtime Environment (JRE) or Java Development Kit (JDK) must be available in order to run Eclipse. No Java virtual machine was found after searching the following locations:

/opt/java/eclipse/jre/bin/java

java in your current PATH

■■■■ Matlab

1. Matlab的安装
2. 挂载

# mount -o loop /media/…/MATHWORKS\_R2014A.iso /media/matlab

1. 安装

# sudo ./install

1. SN

找crack里面的sn: 12345-67890-12345-67890

1. 破解

选择license\_405329\_R2014a.lic

复制/Crack/Linux/libmwservices.so到：/usr/local/MATLAB/R2014a/bin/glnxa64/

1. 桌面创建启动器

# vi ~/Desktop/Matlab.desktop

# chmode +x Matlab.desktop

内容为：

[Desktop Entry]

Type=Application

Name=Matlab

GenericName=Matlab 2014a

Comment=Matlab:The Language of Technical Computing

Exec=sh /usr/local/MATLAB/R2014a/bin/matlab -desktop

Icon=/usr/local/MATLAB/R2014a/toolbox/nnet/nnresource/icons/matlab.png

Terminal=false

Categories=Development;Matlab;

1. Matlab的技巧

运行 shell 命令

! + 命令内容

■■■■ Latex

1. 安装
2. texlive-full,2013

#sudo apt-get install texlive-full

#sudo apt-get install latex-beamer

#sudo apt-get install texmaker

#sudo apt-get install latex-cjk-all

#sudo apt-get install fontforge

（倒着卸载）

1. texlive2016,CD
2. 安装perl-tk

为了使用图形化安装界面，需要安装perl的tk组件：

# sudo apt-get install perl-tk

1. 安装

直接选择使用磁盘映像器挂载打，mount也可以，cd 进入后，运行

./install­tl ­gui(加gui是图形安装的意思，强烈推荐）

1. 配置环境变量

vi ~/.bashrc后，加入

PATH=/usr/local/texlive/2016/bin/x86\_64-linux:$PATH; export PATH

MANPATH=/usr/local/texlive/2016/texmf-dist/doc/man:$MANPATH; export MANPATH

INFOPATH=/usr/local/texlive/2016/texmf-dist/doc/info:$INFOPATH; export INFOPATH

然后，source ~/.bashrc

1. 配置字体

* 将texlive字体配置文件复制到系统内：

sudo cp /usr/local/texlive/2016/texmf-var/fonts/conf/texlive-fontconfig.conf /etc/fonts/conf.d/09-texlive.conf

刷新缓存

sudo fc-cache -fsv

1. 安装字体

#sudo apt-get install latex-cjk-all

1. 测试tex安装成功与否

# tex -version

1. 安装texmaker

如果卸载出现问题：正试图覆盖 /usr/share/texmaker/texmaker\_se.qm，它同时被包含于软件包 texmaker-data 4.1-1，dpkg-deb：错误：子进程 粘贴 被信号(断开的管道) 终止了

则直接 # sudo apt-get remove texmaker-data

# sudo dpkg -i texmaker\_ubuntu\_14.04\_4.5\_amd64.deb

注意：如果出现无法跳转的情况，请用pdflatex编译

[Desktop Entry]

Encoding=UTF-8

Name=TexMaker

Comment=texmaker

Exec=/usr/bin/texmaker

Icon=/usr/share/pixmaps/texmaker.png

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

1. texstudio

<http://texstudio.sourceforge.net/>

注意：如果出现无法跳转的情况，build & view选项采用，Compile&View，其中，compile请用pdflatex编译

另：如果出现提示文字无法显示，可更改默认字体

[Desktop Entry]

Encoding=UTF-8

Name=texstudio

Comment=texstudio

Exec=/usr/bin/texstudio

Icon=/usr/share/texstudio/texstudio48x48.png

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

1. （可选）安装lyx

* 安装依赖：sudo apt­get install zlib1g­dev qt4­designer
* 解压文件后，进入 文件根目录，执行以下命令

./configure

make

sudo make install

* 命令行输入lyx启动了

1. 如何确认CJK安装包的版本

sudo apt-get install apt-show-versions

apt-show-versions | grep -i cjk

1. 快捷方式文件

[Desktop Entry]

Encoding=UTF-8

Name=TexMaker

Comment=texmaker

Exec=/usr/bin/texmaker

Icon=/usr/share/pixmaps/texmaker.png

Terminal=false

StartupNotify=true

Type=Application

Categories=Application;Development;

语言包（待测试）

提示ctexart.cls not found，应该是语言包没有装：

# sudo apt-get install texlive-lang-cjk

1. 安装字体(未成功)

安装方法，终端输入：

$ sudo thunar /usr/share/fonts/truetype

待turetype文件夹打开，创建一个英文名文件夹，将字体ttf文件移进

文件夹后关闭

$ sudo fc-cache

命令刷新字体缓存并重启X-Window.

字体配置：

/usr/share/texlive/texmf-dist/tex/latex/ctex/fontset/ctex-xecjk-winfonts.def

1. zotero与zotero-better-bitex

使用better-bitex

1. 下载 .xpi 文件

<https://github.com/retorquere/zotero-better-bibtex/releases/latest>

目前是 zotero-better-bibtex-1.6.75.xpi

1. 安装 zotero-better-bibtex

在zotero的工具 -> 插件 -> extensions -> (右上的工具按钮) -> Install add-on from file -> 选择 .xpi 文件 -> 重启zotero

1. 配置

打开zotero -> 编辑 -> 首选项 -> 导出 -> 默认输出格式 -> Better BibTex Quick Copy

打开zotero -> 编辑 -> 首选项-> Better BibTex -> Citation keys -> Quick copy/drag-and-drop citations -> QuickCopy format -> Latex

1. 文章类型

* 短篇，10页以内，无目录

\documentclass{article}

* 中长篇，10页以上，有目录

\documentclass{book}

report

beamer

1. 命令
2. 命令的类型

* \+若干英文，
* \+单个字符，

需要空格可以使用{}或者\ ，

1. 命令参数

* 必要参数

{参数1}{参数2}....，参数顺序不能换

* 可选参数

[参数1][参数2]

或者[参数1，参数2，....]或者[参数1=选项，参数2=选项，...]

1. 命令的作用范围

* 声明形式，将作用于命令后的所有内容

如，粗体\bfseries

* 参数形式，只作用于命令所带的参数

如，\textsl{Asia}，Asia均为斜体，\textsl Asia，则只有A为斜体

* 组合形式，

如，{\textsl Asia}

1. 自定义命令

* 通常

\newcommand{新命令}[参数数量][默认值]{定义内容}

* 带星号的，则不能换段或空行

\newcommand\*{新命令}[参数数量][默认值]{定义内容}

* 防止同名命令冲突

\providecommand{新命令}[参数数量][默认值]{定义内容}

1. 字体

* 花体

\mathcal{字母}

或者：\usepackage{mathrsfs}

\mathscr{字母}

1. 编号

\begin{itemize}

\item{xxx}

\item{xxxx}

\end{itemize}

1. 抄录
2. 数学公式
3. 单行公式
4. 行内

* $ ... $
* \( ... \)
* \begin{math} ... \end{math}

1. 行间（单行）

* $$ ... $$，可使用\eqno{(1)}指定公式编号
* \[ ... \]
* \begin{displaymath} ... \end{displaymath}
* 指定编号

\begin{align}

xxxx \tag{xxx}

\end{align}

1. 多行公式

* 无编号

\begin{align\*}

xxxx

\end{align\*}

* 指定编号

\begin{align\*}

xxxx \tag{xxx}

\end{align\*}

& 为对齐，\\换行

1. 公式中插入文字

$\text{减数}-\text{被减数}=\text{差}$

1. 粗体，符号的字体

与文字的类似，如粗体\mathbf{}

或者，\usepackage{bm}，然后 \bm x

1. 矩阵

* 使用环境编写

matrix、pmatrix、bmatrix、Bmatrix、vmatrix或者Vmatrix

\begin{matrix}

0 & 1 \\

1 & 0

\end{matrix}

* 使用array环境来输入矩阵

\begin{equation} %开始数学环境

\left( %左括号

\begin{array}{ccc} %该矩阵一共3列，每一列都居中放置

a11 & a12 & a13\\ %第一行元素

a21 & a22 & a23\\ %第二行元素

\end{array}

\right) %右括号

\end{equation}

1. 表格
2. 用法

\begin{tabular}{|l|c|}括号里面的竖线表示表格中的竖线

\hline 横线

& \\

\hline

\end{tabular}

1. 多行

\usepackage{makecell}

需要多行的项：\makecell{xxx \\ xxx \\}

1. 插图
2. 工具sam2p

#sudo apt-get install sam2p

使用 sam2p xxx.png xxxx.eps

1. 例

\usepackage{graphicx}

\includegraphics[scale=0.5]{weight\_definition.eps}

1. 画图

使用tikz：

|  |  |
| --- | --- |
| 功能 | 命令 |
| 包 | \usepackage{tikz} |
| 行内 | \tikz xxxx |
| 行间 | \begin{tikzpicture}  \end{tikzpicture} |
| 划线 | \draw (0,0) -- (2,2) |
|  |  |
|  |  |
| 设置 | [设置]{文字内容}  [color=red!70,pos=0.25,above,sloped]{Hello} |
| 标注 | node[]{} |
| 预定义属性 | [L1Node/.style={circle, draw=blue!50,fill=blue@20,very thick,minimum size=10mm}]  node[L1Node] (n1) at (0,0){test node} |
| 循环 | \foreach \x in {1,...,5}  \node[N1Node] (w1\_\x) at (2\*\x){$\int\_\x$} |
|  |  |

1. 多文件

\input 直接把文件内容插入\input所在位置，不做多余处理。插入不开启新页。在被引入文件的末尾可以加\endinput显示地结束文件读入，之后可以加注释性文字而不用加注释号。

\include 插入开启新页

\includeonly 插入多文件时，可以一个一个文件编译，提高效率，

1. 目录

* \tabelofcontents
* 跳转功能：

再加上\usepackage{hyperref}

* 去红框：

\usepackage[colorlinks]{hyperref}

改颜色\usepackage[colorlinks,linkcolor=red,anchorcolor=blue,citecolor=green]{hyperref}

注意：要编译两次才能生成。

1. 参考文献
2. 直接使用bibliography

参考文献命令格式：

\begin{thebibliography}{编号样本}

\end{thebibliography}

应放在document环境中，正文后面，\end{document}之前；

每条参考文献格式为：

\bibitem{编号}

在正文中引用参考文献的命令为：

\cite{编号}

1. 使用bibtex
2. 特殊输入

|  |  |  |
| --- | --- | --- |
| 内容 | 输入 | 例 |
| 破折号 | --- |  |
| 负号 | \textnormal{xxx} |  |
| 英文省略号 | \dots或\ldots |  |
| 空格 | {}或\ （空格） |  |
| 换行  (正常行文用的少，一般字诗歌、表格或公式中) | \\ |  |
| \linebreak |  |
| 产生于参数长度一样的空格 | \phantom{参数}  有 \hphantom 和 \vphantom |  |
| 原样输出，\verb后面两个同样字符里面的内容 | \verb |  |
| 同\verb但，可以使空格显示可见 | \verb\* |  |
|  | \begin{verbatim}  \end{verbatim} |  |
| 分段 | 空行 |  |
| 首行缩进 | \usepackage{indentfirst}  不缩进的 在段落前加 \noindent |  |
| 段落加框 | \usepackage{framed}  \begin{framed}  xxxx  \end{framed} |  |
| 上划线 | \overline |  |
| 下划线 | \underline |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |