# Apt命令

sudo apt-cache search opencv

sudo apt-cache search opencv |grep 264

sudo apt-cache search opencv |grep -i x26

sudo apt-cache search 264

sudo apt-get install opencv x264

sudo apt-get install x264

dpkg -L x264

sudo apt-cache search opencv |less

sudo apt-get install libopencv-core2.4

dpkg -L libopencv-core2.4

sudo apt-cache search opencv |grep high

# Apt安装出错的解决方法

ubuntu 安装软件的时候提示“ E: 软件包 openssh-server 需要重新安装，但是我无法找到相应的安装文件 ” 解决方法如下：

su

rm -rf /var/lib/dpkg/info/openssh-server\*

sudo dpkg --remove --force-remove-reinstreq openssh-server

# linux基本命令

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

lsb\_release -a //查看ubuntu版本信息

getconf LONG\_BIT //查看操作系统位数

# grep

grep 'root' /etc/group

grep -n "str" -r ./ //-n是打印行号，-r是在子目录也要查询, -i 搜索的时候可以忽略大小写

grep "NetworkVideo" ./\*

grep '^\(root\|zhang\)' //匹配以root或zhang开头的行

sudo sh -c 'echo "nameservre 192.168.90.2" >> /etc/resolv.conf' //修改DNS: echo和>>是两个命令，都需要sudo权限，故使用sh -c ' '将"echo和>>"合成一个命令

grep --color=auto --exclude=tags --exclude=cscope.\* --exclude=\*.o --exclude=\*.ko bitbake

# find

busybox find /sys -name my\* //在/sys目录下寻找my开头的文件

# tcpdump

tcpdump -i eth0 -nnA 'port 3702'

tcpdump -i eth0 -nnA 'port 80 and src host 192.168.1.231'

tcpdump -i eth0 -nn 'icmp'

-A：以ascii的方式显示数据包，抓取web数据时很有用。

-X：数据包将会以16进制和ascii的方式显示。

-nn：表示以ip和port的方式显示来源主机和目的主机，而不是用主机名和服务。

# iptables

sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8080

//非root用户无法使用1024一下的端口号，可以利用iptables将80端口的数据转到8080端口，非用户绑定8080端口就可以收到80端口的数据了

# Netstat

netstat -gn //查看当前网络添加的组播地址

netstat -apn | grep 21 //搜索占用端口21的进程名及进程端口号

//tcp 0 0 0.0.0.0:21 0.0.0.0:\* LISTEN 1271/busybox

ps -x | grep 1271

kill 1271 //通过进程ID杀死占用端口21的进程

# Sed

sed -i 's/signal/temp/g' /usr/local/bin/test.txt //将文件test.txt中的signal替换成temp

sed -i "s/signal/temp/g" /usr/local/bin/test.txt //将文件test.txt中的signal替换成temp

sed -i s/signal/temp/g `grep signal -rl --include="\*.txt" ./` //在./目录下文件类型为txt的文件中搜索signal字串，并将signal替换成temp

sed: -i 表示操作的是文件， ``括起来的grep命令，表示将grep命令的结果作为操作文件

s///g: g表示一行中有多个相同字串都替换

grep: -l 表示仅列出符合条件的文件名

sed -i s+i2c\_adapter\ =\ \<\&i2c2\>+2c\_adapter\ =\ \<\&i2c2\>+g `grep "i2c\_adapter = <&i2c2>" -rl ./`

# readelf

readelf -a liblive555\_plugin.so

# addr2line

addr2line -e libmedia.so 00057b3f //查看libmedia.so中地址为00057b3f对应的位置

//.so文件位置： /home/yhong/android/out\_703d\_ebox/target/product/gs703d/symbols/system/lib

# objdump

arm-none-linux-gnueabi-objdump -S libmediaplayerservice.so > /home/yhong/dump.txt //反汇编.so文件

//.so文件位置： /home/yhong/android/out\_703d\_ebox/target/product/gs703d/system/lib

# ldd

ldd ./onvifclient //ldd命令查看二进制程序的共享链接库依赖，包括库的名称、起始地址

# nm

nm ./onvifclient //nm命令列出二进制文件中的符号表，包括符号地址、符号类型、符号名等

nm -C onvifclient | grep symbol

# ldconfig

sudo ldconfig -v //查看动态lib库文件加载的详细顺序

# md5sum

md5sum xc6130.ko

# hexdump

hexdump -n 1200 /mnt/media/live555shm\_0

# tar

tar -jcvf bz2/xxx.tar.bz2 xxx --exclude=\*.svn --exclude=\*.git --exclude=\*.repo

tar -xjvf xxx.tar.bz2

tar -zcf bz2/xxx.tar.gz xxx --exclude=\*.svn --exclude=\*.git --exclude=\*.repo

tar -zxf xxx.targz

# wget

使用wget下载单个文件

wget http://www.linuxde.net/testfile.zip

使用参数-O来指定一个文件名

wget -O wordpress.zip http://www.linuxde.net/download.aspx?id=1080

wget限速下载

wget --limit-rate=300k http://www.linuxde.net/testfile.zip

使用wget断点续传

wget -c http://www.linuxde.net/testfile.zip

使用wget后台下载

wget -b http://www.linuxde.net/testfile.zip

tail -f wget-log //察看下载进度

FTP下载

wget ftp-url //匿名ftp下载

wget --ftp-user=USERNAME --ftp-password=PASSWORD url //使用wget用户名和密码认证的ftp下载

# Git cmd

github: yhong061+0714

==============================

git reflog //list git operation and commit id

git branch camera\_0912 b8192b1 //create brance with commit id b8192b1

==============================

https:

echo "# self" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin https://github.com/yhong061/self.git

git push -u origin master

==============================

ssh:

echo "# self" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin git@github.com:yhong061/self.git

git push -u origin master

==============================

git remote add origin https://github.com/yhong061/self.git

git push -u origin master

==============================

git remote add origin git@github.com:yhong061/self.git

git push -u origin master

==============================

==============================

git clone https://github.com/yhong061/sample.git

cd sample

vim git.txt

git add git.txt

git commit -sm "add git.txt"

cat .git/config

git push origin master

==============================

git remote -v //list remote name

==============================

git blame kernel/xc6130\_reg/readme.txt //list file's modify author per lines

==============================

git log --oneline //display log in one line

git log --graph //display log in a graph mode

git log -all //display all branchs log

==============================

git reflog //list HEAD pointer's trajectory

==============================

git add -p readme.txt //separate readme.txt's modification to serial parts

//s(separate) --> y(yes) --> n(no)

==============================

git rebase -i HEAD~2 //compress the last 2 commits to 1

==============================

git stash //stash current work env.

git stash list //list stash

git stash apply //restory stash, default is stash@{0}

git stash apply stash@{2} //restory stash's name stash@{2}

==============================

git fsck --lost-found //list the lost commit

==============================

//clone remote a branch not remote all branchs

git init

git remote add -t BRANCH\_NAME\_HERE -f origin REMOTE\_REPO\_URL\_PATH\_HERE

git checkout BRNACH\_NAME\_HERE

==============================

//save the changed filename to "update.zip" between serial commits

git archive -o ../updated.zip HEAD $(git diff --name-only HEAD^) //save the changed filename between HEAD and HEAD^

git archive -o ../updated.zip NEW\_COMMIT\_ID $(git diff --name-only OLD\_COMMIT\_ID NEW\_COMMIT\_ID) //save the changed file between NEW and OLD commit ID

# Git cmd

git blame arch/arm64/boot/dts/gt7\_ebox.dts

//查看文件的每一行的提交者信息

git blame -L 160,+10 arch/arm64/boot/dts/gt7\_ebox.dts

//查看文件的每一行的提交者信息，指定行

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

repo sync android/device/actions/common/frameworks/video

vim /home/yhong/work/GS700E\_5110\_AR\_160907/android/.repo/manifest.xml

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

patch -p1 < ../video2/0001-Add-monofreq-and-multifreq-PLL-Setting.patch

git status

git add .

git status

git am --continue -s

git commit -as --amend

传统的打patch方式

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

git whatchanged .

查看commitID涉及到的文件修改

git format-patch -1

生成commitID对应的patch, 1代表生成倒数第一个commitID的patch, 2代表生成倒数第一个和倒数第二个commitID的patch

git am –abort

放弃掉以前的am信息，这样才可以进行一次全新的am

git am ../video2/0001-Add-monofreq-and-multifreq-PLL-Setting.patch

可以一次合并一个文件，或者一个目录下所有的patch

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

git cherry-pick 8168fb32d64fff496f1b6db7063e8d1bd4a5147c

//将其他分支的指定commit-id分支同步到当前分支

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

593 git status

594 git commit -asm "add driver xc6130"

595 git status

596 git log

597 git branch

598 git checkout dev

599 git log

600 git show -p

601 rm test

602 git status

603 git rm test

604 git status

605 git commit -asm "revert to top tree "

606 git log

607 git pull

608 git branch

609 git checkout yhong

610 git log

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

git push gl5209 yhong1:review/TB\_ipc\_ov2718

531 git branch

532 git status

533 git diff

534 git reset --hard

535 git status

536 git fetch gl5209

537 git branch -a | grep ov2718

538 git checkout -b ov2718remotes/gl5209/review/TB\_ipc\_ov2718

539 git checkout -b ov2718 remotes/gl5209/review/TB\_ipc\_ov2718

540 git branch

541 git branch -D ov2718remotes/gl5209/review/TB\_ipc\_ov2718

542 git branch

543 git log

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

639 git branch

640 git status

641 git commit -as

642 git log

643 git branch

644 cat .git/config

645 git branch

646 git push gl5209 yhong:review/TB\_IPC\_yhong

647 git branch -a | grep yhong

648 git branch -a

649 git branch -a | grep yhong

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

595 git checkout -b ipc remotes/gl5206/sh/GS700C\_5110\_IPC\_20160825

596 git branch

597 git checkout yhong

598 git push gl5206 yhong:review/TB\_ipc\_camera

599 git log

600 git checkout ipc

601 git cherry-pick f4d9b3104974382ff172288c669bfb81cd73ab23

602 git log

603 git pull

604 git push gl5206 ipc:sh/GS700C\_5110\_IPC\_20160825

=====================================================================

git fetch gl5209

git rebase

=====================================================================

HEAD: 表示当前版本

HEAD^: 表示上一个版本

HEAD^^: 表示上上一个版本

HEAD~100: 表示当前版本网上100个版本

=====================================================================

git add:

git add <path> //添加 created or modified 文件，不包括 deleted 文件

git add -u <path> //添加 modified or deleted 文件，不包括 created 文件

git add -A <path>

git add -i //查看文件状态

git rm path/file //删除通过add命令提交到暂存区的文件

=====================================================================

git branch:

git checkout branch //检出分支

git checkout -b branch //创建分支并检出分支

git branch -d dev //删除dev分支

关联远程分支：

git checkout -b dev; //创建本地分支dev

git pull origin dev //关联远程分支origin和本地分支dev，dev和origin将保持一致

git push origin dev:tmp //将本地分支dev的commit内容推送到远程仓库origin的tmp分支

git push origin :tmp //删除远程仓库origin的tmp分支

git fetch

git branch --set-upstream dev gl5206/sh/TB\_multistream

//--set-upstream: 跟踪远程分支

//本地分支名：dev

//远程分支名：remotes/gl5206/sh/TB\_multistream

//最终结果：在 .git/config 文件中增加关联信息

[branch "dev"]

remote = gl5206

merge = refs/heads/sh/TB\_multistream

=====================================================================

git checkout/reset:

git reset --hard commit-id //回退到指定版本号的地方

git reset HEAD file //取消暂存区的修改，并将暂存区的修改返回到工作区

git checkout -- filename //取消工作区的修改

git checkout branch -- filename //用branch所指向的提交中的filename替换暂存区和工作区的文件

git checkout . //用暂存区的所有文件覆盖本地文件

git checkout -- . //用暂存区的所有文件覆盖本地文件

git checkout //汇总当前目录及子目录的文件在工作区，暂存区，HEAD的差异

git checkout HEAD //汇总当前目录及子目录的文件在工作区，暂存区，HEAD的差异

=====================================================================

git diff:

git diff readme.txt //比较本地文件跟git仓库的不同

git diff HEAD -- readme.txt //比较工作区文件跟git仓库(本地分支)的不同

=====================================================================

git merge:

get merge dev //将dev分支合并到当前分支，假定当前处于master分支

get merge --no-ff -m "no-ff" dev //将dev分支合并到当前分支，并为dev分支创建commit-id

--no-ff表示不是用Fast forward方式合并分支

fast forward方式合并只是移动了HEAD指针，并没有做内容拷贝

可通过git log查看分支历史

=====================================================================

git init //创建空的版本库

git add readme.txt //将代码修改提交到暂存区

git commit -m "add readme.txt" //把暂存区的修改文件提交到git仓库

git status //查看git仓库修改状态

git diff readme.txt //比较本地文件跟git仓库的不同

git diff HEAD -- readme.txt //比较工作区文件跟git仓库的不同

git log //查看git仓库的修改记录

git log --pretty=oneline //查看git仓库的修改记录, 在一行显示

git reset --hard HEAD^ //获取上一个版本

HEAD: 表示当前版本

HEAD^: 表示上一个版本

HEAD^^: 表示上上一个版本

HEAD~100: 表示当前版本网上100个版本

git reset --hard commit-id //回退到指定版本号的地方

git reset HEAD file //取消暂存区的修改，并将暂存区的修改返回到工作区

git checkout -- readme.txt //取消工作区的修改

rm test.txt;git rm test.txt;git commit -m "remove test.txt"

//删除工作区文件，删除暂存区文件，删除git仓库文件

rm test.txt;git rm test.txt;git checkout -- test.txt

//文件误删：删除工作区文件，删除暂存区文件，恢复删除的文件

//创建远程库

登陆GitHub网站

点击"Create a new repo",创建远程库

"Repository name"中输入"learngit"

git remote add origin git@github.com:michaelliao/learngit.git //关联本地git仓库到远程

origin: 远程库名

michaelliao: GitHub的账户名,自己创建

git push -u origin master //将本地库内容推送到远程仓库, 第一次推送

git push origin master //将本地库内容推送到远程仓库

git clone git@github.com:michaelliao/gitskills.git //克隆远程库，gitskill为远程库名

git brance dev //创建dev分支

git checkout -b dev //创建dev分支，并切换到dev分支

git branch //查看所有分支，当前分支前用"\*"表示

git checkout dev //切换到dev分支， 即将HEAD指针指向dev

git checkout master //切换到主分支， 即将HEAD指针指向master

get merge dev //将dev分支合并到当前分支，假定当前处于master分支

get merge --no-ff -m "no-ff" dev //将dev分支合并到当前分支，并为dev分支创建commit-id

--no-ff表示不是用Fast forward方式合并分支

fast forward方式合并只是移动了HEAD指针，并没有做内容拷贝

可通过git log查看分支历史

git branch -d dev //删除dev分支

//解决冲突

//修改feature1分支上的readme.txt文件

git checkout -b feature1

修改readme.txt文件

git add readme.txt

git commint -m "AND simple"

//修改master分支上的readme.txt文件

git checkout master

修改readme.txt文件

git add readme.txt

git commint -m "& simple"

git merge feature1 //产生冲突

git status //查看冲突的文件

直接查看工作区中的readme.txt，并修改冲突

//文件中<<<<<<< HEAD, =========, >>>>>>> feature1标记了冲突的内容，直接修改即可

git add readme.txt //提交修改冲突后的文件

git log --graph --pretty=oneline --abbrev-commit //查看分支的合并情况

git brance -d feature1 //删除feature1分支

//分支储藏: 临时插进来的任务,如bug的修改

git stash //将当前分支修改内容储藏起来

git checkout master //切换到master分支

git checkout -b issue //创建新的分支issue

修改issue分支工作区的readme.txt //修改bug

git add readme.txt //提交修改内容

git commit -m "fix bug 101" //提交修改内容

git checkout master //切换到master分支

git merge --no-ff -m "merged bug fix 101" issue //合并issue分支到主分支

git brance -d issue //删除issue分支

git checkout dev //切换到dev分支

git status //显示dev分支的修改记录： 无

git stash list //查看分支的储藏情况

git stash apply //恢复stash的内容，但stash内容不删除

git stash apply stash@{0} //恢复指定的stash的内容，针对多次stash

git stash pop //恢复stash的内容，stash内容删除

git stash drop //删除stash的内容

//远程库冲突

//远程库名: origin

git remote //查看远程库的信息

git remote -v //查看远程库的详细信息

git push origin master //将master分支上所有本地信息提交到远程库

git push origin dev //将dev分支上所有本地信息提交到远程库

//第一位修改者

git clone git@github.com:michaelliao/learngit.git //克隆远程库learngit

git branch //本地库将生成master分支

git checkout -b dev origin/dev //创建远程origin的dev分支到本地

修改dev分支文件： hello.py

git add hello.py

git commit -m "add coding: /usr/bin/env"

git push origin dev

//第二位修改者

git clone git@github.com:michaelliao/learngit.git //克隆远程库learngit

git branch //本地库将生成master分支

git checkout -b dev origin/dev //创建远程origin的dev分支到本地

修改dev分支文件: hello.py

git add hello.py

git commit -m "add coding: utf-8 "

git push origin dev //与第一位修改者产生冲突

//修改远程冲突

git brance --set-upstream dev origin/dev //建立本地dev与远程origin/dev的链接

//--set-upstream解决"git push"报错：no tracking information

git pull //将最新的提交从origin/dev上抓取下来

本地修改冲突的文件

git add hello.py

git commit -m "merge & fix hello.py"

git push origin dev

git tag V1.0 //给当前分支打上标签

git tag V1.0 commit-id //在commit-id处打上标签

git tag //查看当前分支标签

git show V1.0 //查看标签信息

git tag -a v0.1 -m "version 0.1 released" commit-id //在commit-id处打上标签，并添加信息

git tag -d V1.0 //删除V1.0的标签

git push origin v1.0 //给远程库origin打标签

git push origin --tags //将本地所有的标签推送给远程库

git push origin :refs/tags/v1.0 //删除远程库上的标签

# Vi Cmd

**dd** 删除整行

**nG** 跳到指定行

**yy** 拷贝一行

**p** 粘贴

**u** 取消上一次操作

**U** 取消所有操作

c**trl+r** 对撤销的撤销

**:e!** 放弃更改，然后相当于重新打开

**ma** 标记当前行为'a'

y`a 复制当前行至标记`a的位置

d`a 删除当前行至标记`a的位置

**%** 移动到匹配的括号上去，如{}, [], ()

**`** 移动到上一次的修改行

**v%** 将光标放到{处，执行v%就会选中{}中的内容

**fx** 向右跳转到字符x处

**Fx** 向左............

**tx** 向右跳转到字符x前

**Tx** 向左.............

**#** 到与当前单词相同的上一个单词

**\*** ..................下一个....

**:set hlsearch** 高亮显示查找结果

**:set nohlsearch** 取消高亮

**nG**：光标移至第n行首

**``**  回到原来行

**zz** 将当前行放置屏幕中间

**~** 更改字母的大小写

**.**  重复上一个命令

## 打开文件：

**vi filename :**打开或新建文件，并将光标置于第一行首

**vi +n filename** ：打开文件，并将光标置于第n行首

**vi + filename** ：打开文件，并将光标置于最后一行首

**vi +/pattern filename**：打开文件，并将光标置于第一个与pattern匹配的串处

## 光标移动：

**k、j、h、l**——上、下、左、右光标移动命令

**%**是从大括号的开始移动到大括号的结束位置

**:**后边加行号就是跳到这一行

**gg** 移动到文件第一行

**shift + g**移动到文件末尾

**)** ：光标移至句尾

**(** ：光标移至句首

**}**：光标移至段落开头

**{**：光标移至段落结尾

nG：光标移至第n行首

`` 回到原来行

n+：光标下移n行

n-：光标上移n行

n$：光标移至第n行尾

0：（注意是数字零）光标移至当前行首

^:移动光标到行首；

$:移动光标到行尾；

shift -移动到光标所在行行首

shift +移动到光标所在行下一行首

H ：光标移至屏幕顶行

M ：光标移至屏幕中间行

L ：光标移至屏幕最后行

ctrl-b:类似于键盘上的"PgUp"(b为backword)

ctrl-f：类似于键盘上的"PgDn"(f为forword)

w:光标移动到下一个单词的词首；注：对于中文，连续的多个汉字作为一个word。

2w:重复执行w操作2次；

e:光标移动到下一个单词的词尾；

5e:重复执行e操作5次；

b：向前移动光标，移动到前一个单词的词首；

m+单词（单词为a-z或者A-Z）

ma, 设定当前行标记为'a', 输入`a, 则会返回到标记为a的地方

mb, 设定当前行标记为'b', 输入`b, 则会返回到标记为b的地方

:delmarks 删除指定标记

:marks 列出所有的标记

' 跳转前的光标位置

" 最后编辑的光标位置

[ 最后修改的开始位置

] 最后修改的结束位置

## 删除命令：

ndw/ndW：删除光标处开始及其后的n-1个字

do：删至行首

d$：删至行尾

ndd：删除当前行及其后n-1行

x/X：删除一个字符，x删除光标后的，而X删除光标前的

**J**:删除换行符，使下一行并过来

**nJ:** 连接后面的n行

## 搜索及替换命令

:[range]s/{pattern}/{string}/[flag]

[range]通常是两个整数，表示作用域起始行和结束行的行号，中间以一个逗号隔开。如："4,8"表示第4行到第8行（包括第4行和第8行）。

百分号'%'表示文档的所有行。

省略不写则代表光标所在行，

"$"代表文本的最后一行

{pattern}和{string}都支持正则表达式。

{pattern}为空串，则使用上一条替换命令中的 {pattern}作为模式串pattern

{pattern}和{string}中不能包含分隔符，否则会产生歧义。为此，要使用反斜杠'\'进行转义。

如:"%s/<br>/<br />/g"必须写成":%s/<br>/<br \/>/g"

如果这两个表达式中的斜杠非常多，比如"file:///usr/share/man/man1/vim.1.gz"，那么可以使用其他字符作为分隔符（总是将作为替换命令代表的's'后的第一个字符作为分隔符）

如:%s+<br>+<br />+g ('+'加号代表分隔符)

[flag]有多种，"g"表示替换[range]范围内每行中的所有匹配模式，而不写"g"时，只对每行第一次匹配出现的{pattern}进行替换。

"c"表示每次替换前要求确认，

"i"表示不区分大小写，

/pattern：从光标开始处向文件尾搜索pattern

?pattern：从光标开始处向文件首搜索pattern

n：在同一方向重复上一次搜索命令

N：在反方向上重复上一次搜索命令

:s/str1/str2/ 用字符串 str2 替换行中首次出现的字符串 str1

:s/str1/str2/g 用字符串 str2 替换行中所有出现的字符串 str1

:.,$ s/str1/str2/g 用字符串 str2 替换正文当前行到末尾所有出现的字符串 str1

:1,$ s/str1/str2/g 用字符串 str2 替换正文中所有出现的字符串 str1

:g/str1/s//str2/g 功能同上

g 放在命令末尾，表示对搜索字符串的每次出现进行替换；

不加 g，表示只对搜索字符串的首次出现进行替换；

g 放在命令开头，表示对正文中所有包含搜索字符串的行进行替换操作。

## 搜索关键字高亮：支持记录3种搜索高亮

：match Search /main/ //搜索关键字“main”, 并按Search定义的字体颜色显示, 字体颜色详见hi命令

: 2macth Search /main1/

: 3match Search /main2/

## 同时显示多个文件：

:split

:vsplit

## 1.文件间切换

Ctrl+6—下一个文件

:bn—下一个文件

:bp—上一个文件

对于用(v)split在多个窗格中打开的文件，这种方法只会在当前窗格中切换不同的文件。

## 2.在窗格间切换的方法

Ctrl+w+方向键——切换到前／下／上／后一个窗格

Ctrl+w+h/j/k/l ——同上

Ctrl+ww——依次向后切换到下一个窗格中

## 打开指定函数：

vim -t main

:ts //如果有多个main的话，执行这个命令会列出所有main函数入口

## 复制指定字符：

在命令行下按"v"进入可视模式，然后按"H"或"L"移动光标，选中要复制的字符，

然后按一次"y"。。。这样就复制下来了，如果要剪切就按“d“

## vim命令：

set ignorecase

set noignorecase

## 折叠代码

zf进行折叠,

zo打开折叠,也可以方向键向右打开折叠,

zc 关闭折叠(只要在被折叠的块中的任一个语句就行)

缩进代码

# Ctags

git clone <https://github.com/SublimeText/CTags>

or

sudo apt-get install exuberant-ctags

sudo apt-get install ctags

$ cd /home/aepc/clxu/vlc-2.1.4

$ ctags -R //在/home/aepc/clxu/vlc-2.1.4目录下会生成一个 tags 文件, -R表示递归子文件夹

$ vim /home/aepc/clxu/vlc-2.1.4/bin/vlc.c

:set tags=/home/aepc/clxu/vlc-2.1.4/tags //该命令将tags文件加入到vim中来

ctrl+], :tselect 列出光标所在函数的所有定义处

Ctrl+] 直接跳转到第一个 func tag 处

g] 不跳转，列出有多少个 func tags 可选

g Ctrl+] 一般同 g]，但如果 func tag 只有一个，则直接跳转过去

ctrl+T, 这样又跳回到函数被调用的地方

ctrl+N, 自动补全

# cscope

sudo apt-get install cscope

$ cd /home/aepc/clxu/vlc-2.1.4

$ cscope -Rbkq //在/home/aepc/clxu/vlc-2.1.4目录下会生成cscope.out, cscope.in.out, cscope.po.out

//cscope.out是基本的符号索引，后两个文件是使用”-q”选项生成的，可以加快cscope的索引速度。

-R: 在生成索引文件时，搜索子目录树中的代码

-b: 只生成索引文件，不进入cscope的界面

-k: 在生成索引文件时，不搜索/usr/include目录

-q: 生成cscope.in.out和cscope.po.out文件，加快cscope的索引速度

//cscope在产生索引文件时，只搜索类型为C, lex和yacc的文件(后缀名为.c, .h, .l, .y)，C++的文件根本没有生成索引。不过按照手册上的说明，cscope支持c++和Java语言的文件。

c++:

find . -name "\*.h" -o -name "\*.c"-o -name "\*.cc" > cscope.files

cscope -bkq -i cscope.files

ctags -R

:cs add ./cscope.out

:cs find c soap\_init1 //find caller

:cs find c soap\_init1 //find definition

# Taglist

1. 从http://www.vim.org/scripts/script.php?script\_id=273下载安装包.

2. 进入~/.vim目录，将Taglist安装包解压，解压后会在~/.vim目录中生成几个新子目录，如plugin和doc（安装其它插件时，可能还会新建autoload等其它目录）。

3. 进入~/.vim/doc目录，在Vim下运行"helptags ."命令。此步骤是将doc下的帮助文档加入到Vim的帮助主题中，这样我们就可以通过在Vim中运行“help taglist.txt”查看taglist帮助。

4. 打开配置文件~/.vimrc，加入以下几行：

let Tlist\_Show\_One\_File=1 "不同时显示多个文件的tag，只显示当前文件的

let Tlist\_Exit\_OnlyWindow=1 "如果taglist窗口是最后一个窗口，则退出vim

let Tlist\_Ctags\_Cmd="/usr/bin/ctags" "将taglist与ctags关联

基本功能使用方法

在Vim命令行下运行":Tlist"就可以打开Taglist窗口，再次运行":Tlist"则关闭。

左右窗口切换Ctrl+ww

在taglist窗口中，可以使用下面的快捷键：

<CR> 跳到光标下tag所定义的位置，用鼠标双击此tag功能也一样

o 在一个新打开的窗口中显示光标下tag

<Space> 显示光标下tag的原型定义

u 更新taglist窗口中的tag

s 更改排序方式，在按名字排序和按出现顺序排序间切换

x taglist窗口放大和缩小，方便查看较长的tag

+ 打开一个折叠，同zo

- 将tag折叠起来，同zc

\* 打开所有的折叠，同zR

= 将所有tag折叠起来，同zM

[[ 跳到前一个文件

]] 跳到后一个文件

q 关闭taglist窗口

<F1> 显示帮助

# End