Basic Knowledge

Yan Zhao

Contents

0.1 Linux basic

- There is a good article to introduce Linux distribution. **The name is "RedHat vs Debian : Administrative Point of View".** You can google and read it. In one word. RedHat is stable, commercial server with small amount of packages. Fedora and CentOS are based on it. Fedora is cutting edge implementation(For test). Another big group is Debian, It's also stable with much more packages. Ubuntu is based on Debian.

- Ubuntu is mostly for desktop, and Mint is sleek and quick version of Ubuntu, CentOS is for server, It's more stable. **If you use virtual machine, recommend Mint, if you want to install linux directly on a computer, you can use Ubuntu system.**, because it's more friend toward beginner and PC.

- There are two Interface Framework, Qt and GTK. KDE is based on QT, and Gnome and Cinnamon(Mint) is based on GTK.

- You can use `$ uname ↵` to check basic information about your computer, detail can be seen uname –help. The processor type (or name) refers for what architecture has been made the processor. The hardware machine name must be compatible with the processor type, in other words, must be the same type as the processor type. And finally, the hardware platform refers to the whole instructions that the hardware uses to process and which it needn't be a higher version than the processor type. i686=x86_64
- `$ lscpu ↵` will tell you how many cores do you have. Socket is physics CPU, "cores per socket" is number of cores.

0.2 Shell

0.2.1 Shell type

- There are two basic shell, one is bash and the other is tcsh. They can be change by calling `$ bash ↵` or `$ tcsh ↵`. You can use echo $0 to see which shell you are using right now.
- By now, a new shell is zsh, It support more features.
- You can use `$ chsh ↵`, -l will list all available shell, and -s follow new shell full path name, such as `$ chsh -s /bin/bash ↵`.
- Every time when you open a new terminal, It will read .bashrc or .rshrc. It depends on what shell you are using, that is why you need to use `$ echo $0 ↵` to know which shell you are using. It will use export(bash) or setenv(csh) to load all envoriment variable.
- In tcsh, you can use bindkey -e and bindkey -v to change to emacs mode and vi mode.In bash, you can use set -o vi or set -o emacs.
- Once in vim mode, It's important to distinguish insert mode and command mode. For bash(version 4.3 or 4.4) you can use "set show-mode-in-prompt on" in .inputrc file. For zsh, you also can implement it. But in tcsh, there isn't any good way to show message on prompt. Just accept it.

0.2.2 wild character and quote

- Single quotes preserves the literal value of each character. Double quotes preserves the literal value of all characters within the quotes, with the exception of '$', 'backtick', '\', and, when history expansion is enabled, '!'.

```
echo '$PATH `pwd` '
echo "$PATH `pwd` \$PATH"
```

- For shell, There are "*, ?, [ ] and {}" wild character. **Any time you input these four wild characters, shell will interpret it according to all the files names.**
- {} will not match file name, echo a{.h,.c} just output a.h and a.c **Don't use space inside {}** .
- For example echo [a-d]c.
  1. shell see [ ], it will expand according to file names. If it doesn't match, it will print No match. If there are bc file or dc file. [a-d]c will be expanded to "bc dc".
  2. Then shell invoke echo command.
- For {}, You use `$ echo {*.txt,*.c} ↵`. **No space after comma.** It will expanded to a.txt b.c if there is a.txt and b.c in your directory.
- If you don't want shell to interpret wild character, use \
- A good example of wild character is `$ grep -exclude=a\*.h 'a*b' *.h ↵`. You should know three * meaning in previous command.

0.2.3 Environment variable

- When bash is invoked as an interactive login shell, or as a non-interactive shell with the –login option, it first reads and executes commands from the file /etc/profile, if that file exists. Then/etc/profile.d/*.sh. Then it looks for /.bash_profile, /.bash_login, and /.profile, in that order, and reads and executes commands from the first one that exists and is readable. The –noprofile option may be used when the shell is started to inhibit this behavior.
- When an interactive shell that is not a login shell is started, bash reads and executes commands from /etc/bash.bashrc and /.bashrc, if these files exist. This may be inhibited by using the –norc option. The –rcfile file option will force bash to read and execute commands from file instead of /etc/bash.bashrc and /.bashrc.

- For tcsh, read .cshrc file, for zsh, read .zshrc file.
- This goes pretty far back in the Unix history. rc stands for "run commands", and makes sense actually.
- echo $0 will show "bash", it's not login shell. If it shows "-bash", it is login shell.
- Why we need variable, just like macro in C language. With variable, we can configure and customize an application outside. And we can customize a variable to affect a lot of applications. Such as $PATH.
- For different shell, you have different syntax in shell script. For example, for bash shell you can use YanVar=123, There is no space between =.You also can use unset command "unset YanVar". Or use check command: `$ echo $YanVar ↵`. Different shell have different syntax, tcsh muse use set. **Pay attention, in shell, there is no variable, just macro, so expansion is key idea to understand it's behaviour. Such as expansion order**.
- There are two kinds of variables: environment and user defined.
- `$ env ↵` shows all the environment variable. Such as $HOME, $PATH, $LANG,$EDITOR which can specify you default editor in your system.`$ set ↵` list all the local environment variables and user defined variable , that is more than env command. For example, the $PS1.
- `$ getconf ↵` can get some system variable, such as `$ getconf ARG_MAX ↵`, you can use xargs -n 50 to make command satisfy the ARG_MAX
- In mint, maybe in your home directory, there is no .bashrc file, so you need to create one and add export PATH=$PATH:/home/yan/openuh-install/bin then exit the current terminal and restart a new one. Then use echo $PATH to see if the directory has been added.
- In previous command, Linux use ":" but windows use ";" why windows use different?
- If you already in terminal, you can use "export" or "setenv" command to add environment variable. If these commands are in a sh file, you have to use `$ source a.sh ↵` to run. Because, if no source command, it will open a child terminal, so after a.sh finish, child terminal will disappear, and all the environment variable you just set in the child terminal will disappear too.

- If There is only one environment variable need to be set, you can run setenv or export in your current shell directly, then you can run a command, this command will be run in the child process, but child process can access parent environment variable, it's OK
- `$ export DEPART=Sale ↵` and `$ DEPART=Sale ; export DEPART ↵` they means the same.
- `$ setenv ↵` is csh command. In bash, you can use `$ export ↵` directly. **csh doesn't use =, bashrc doesn't use backslash.**

  ```
  in .cshrc
  setenv PATH $PATH\:/users/yzhao4/python-3.23/bin

  in .bashrc
  export PATH=$PATH:/storage/yzhao/binexport
  ```

- If There are a few environment variables need to be set, You'd better put them into a script file. **But you have to use source a.sh** to make sure a.sh running in the current process, not a child process.
- Two shells don't share history and environment variables.

0.2.4 pipe

- pipe command can be called "filter", It will accept input from STDIN, perform operations, then send it to STDOUT. Such as **grep, uniq, sort, fmt, pr, head, cut, tee, tr, join, paste, expand, split, tr, awk, sed, less**
- You can't judge if a command is filter by running it. "cat" will hang up to wait for you from STDIN, but less will show error message.
- xargs is also a filter, you can use this way `$ find . -name "*.c" | xargs rm -rf ↵` a.c and b.c will be rm command arguments.
- Sometimes, a command need a file name as input or output, but you don't want to give a filename, It usually used in pipe commands, such as `$ gzip -dc a.tar.gz | tar -xvf - ↵`. It just like `$ gzip -dc a.tar.gz | tar -xvf /dev/stdin ↵`
- "-" has nothing with shell, It's only available in tar command. Most of time, It's only used in tar.

0.2.5 shell script

Basic

- The first line of script is `#!/bin/bash`. This is a special clue given to the shell indicating what program is used to interpret the script. Other scripting languages such as perl, awk, tcl, Tk, and python can also use this mechanism.

- After you finished script file, use `$ chmod +x your-script-name ↵`.

- echo -e "aaa\nbbb" will output two lines.

- `$ !! ↵` run the previous command, `$ !$ ↵` is the previous argument. `$ $? ↵` is the last bash command result, If it's 0, means that everything is OK. Detail can be found in google "Become a Command Line Ninja With These Time-Saving Shortcuts"

- `$ command1;command2 ↵`. To run two command with one command line. `$ command1 && command2 ↵` command2 is executed if, and only if, command1 returns an exit status of zero. `$ command1 || command2 ↵` command2 is executed if and only if command1 returns a non-zero exit status.

Variable

- In script file, $0 is script name, and $1, $2.. are arguments to the scripts.

- print the contains of variable (HOME) and not the HOME. You must use $ followed by variable name to print variable. `echo $HOME` print the variable contents. And `echo HOME` just print "HOME" string. It's a little confused for a C programmer, but you need to be used to it.

- In most cases, $var and ${var} are the same. The braces are only needed to resolve ambiguity in expressions: such as echo ${var}bar

- $(command) is same as ‘command.

0.2.6 Terminal tips

- Motion shortcut keys are list below: Ctrl+p or up down arrow key can go to the previous command, this is very useful.

shortcut	function
Ctrl+f,b	forward, backward character
alt+f,b	forward, backward word
Ctrl+a,e	move start,end
Ctrl+p,n	previous, next line
Ctrl+r	search command history

- delete shortcut keys: You can use "hw" to delete previous character or word. "dd" use to delete forward character or word, "uk" to sentence beginning and end. In Vim, the keyboard shortcut is not the same. **They are bash edit keyshortcut.**

shortcut	fucntion
Ctrl+l	clear the screen
ctrl+k,u	delete to begin/end
alt+d ctrl+w	delete forward/backward word
Ctrl+d,h	delete forward/backward a character

- You can remember "eu" "ak". They both delete the whole line, The first letter is move command. And the second letter is edit command.

- `$ bind -p` ↵ will list all the shortcut. This is bash command. In tcsh shell, you can use `$ bindkey -v` ↵ to make your commad line edit function compatible with VI. Also use esc to switch command mode and insert mode. For example, by press Esc, then, you can use fx command to jump x character in the command line. It's very convince for edit long command line.

- About keyboard shortcut, I have good idea, that is to use left Ctrl and Alt together, because you can use your thumb to press Alt and use palm to press Ctrl_L,(Even in my three laptops, I also can press Ctrl_L easily by palm). So a shortcut can be defined below:

$$\left\{\begin{array}{ll} \text{move} & \left\{\begin{array}{l}\text{other: Ctrl\_L}\\ \text{emacs: Alt\_L}\end{array}\right. \\ \text{select} & \left\{\begin{array}{l}\text{other: Ctrl\_L+Shift\_L}\\ \text{emacs: Alt\_L+Shift\_L}\end{array}\right.\end{array}\right. + \left\{\begin{array}{c}\text{left character: J}\\ \text{right character: L}\\ \text{upward: I}\\ \text{downward: k}\\ \text{left word: U}\\ \text{right word: O}\\ \text{begin line: H}\\ \text{end line: ;}\end{array}\right.$$

Delete command is below:

$$\text{delete}\begin{cases}\text{left character: Backspace}\\ \text{right charcter: Ctrl\_L+N}\\ \text{left word: Ctrl\_L+Backspace}\\ \text{right word: Ctrl\_L+M}\\ \text{line: Ctrl\_L+P}\end{cases}$$

Question 1: why always left Ctrl?
Answer: Now, if you are smart enough, you can found that there is rules inside. All the commands is left Ctrl add right hand character, becuase left Ctrl can be pressed by left palm and right hand is more flexible than left hand when you click the different character.

Question 2: why other use Ctrl and Emacs use Alt.
Answer: In common applications, Alt has been assign to trigger menu item, such as Alt+F will trigger File menu, so, I must use Ctrl. In Emacs, on the contrary, Ctrl has been used to trigger some common commands, so I use Alt key(and Alt is used not often as Ctrl).

Question 3: How can I export my custom shortcut to other computers
Answer: There are two kind of shortcut one is kate and other is kile, they store in `.kde/share/apps/katepart/katepartui.rc` and `.kde/share/apps/kile/kileui.rc` you can copy them and cover them in your computer. If version is different, Maybe it's a little difficult. But you can just do it within the application, it don't need very long time.

By now, these customized shortcuts haven't been used in practical use. Anyway, you can use arrowkey, it don't need too much memory. But it is a good suggestion. You can learn how to define a customized shortcut. If you need to do a lot texting job, they are very useful.

- Ctrl+Alt+F1…F6 switch terminal. Ctrl+Alt+F7 return back to GUI. When F7 doesn't work, you can try F8.

0.2.7 Time

- For Linux file time: there are three time stamps: atime (access time), it is when the file was last read. ctime is the inode change time, while mtime is the file modification time. mtime changes when you write to the file. It is the age of the data in the file. **Whenever atime or mtime changes, so does ctime, except you use touch command** But ctime changes a few extra times. For example, it will change if you change the owner or the permissions on the file.

- timestamp will be used in many linux commands, ls -l will show modification time. and you can use `$ stat fileName ↵` to see all the three time. The can be used in find command.

- atime sometimes will not updated by visiting a file. Atime updates are by far the biggest IO performance deficiency that Linux has today. So sometimes it's disable when mount in option in /etc/fstab

- `$ touch ↵` can change time of a file or you can use it to produce an empty file. `$ touch -a existFile ↵` change access time and ctime. `$ touch -t existFile ↵` change modify time and ctime. `$ touch -c existFile ↵` change a,c and m time.

- `$ touch -t YYMMDDHHmm ↵` will set mtime and atime to the date you want and it sets ctime to **NOW**. You have complete control over mtime, but the system stays in control of ctime. So mtime is a little bit like the date on a letter while ctime is like the postmark on the envelope. System use ctime to do backup job. An example can be found in my evernote book mark.

0.3 File and Dir

0.3.1 Basic

- A hard link points to the file by **inode**. A symbolic link points to the file by **filename**.

- There is no "real" hard link name; All hard links are equally valid names for the file. You can use `$ ls -l ↵`. The first number after the file mode is the link count(this count is represent hard link number). For symbolic link, It just point to a filename, If origin file name changed, symbolic link will not be valid. symbolic link is very flexible, It can be linked to a dir or it can be linked to different file system. But hard link has many restriction.

- `$ find -L / -samefile path/to/foo.txt ↵`find all files links to foo.txt

- Absolute directory must begin with root directory /.

- Linux doesn't use extension name to specify file type, you should use `$ file fileName ↵` to judge it.

- When you use ls -l, the first character stands for different kinds: -:file, d:Dir, l:link file, b:interface of device. So you can use `$ ls -al | grep ^d ↵` to show all the directories.

- `$ ls -d */ ↵` will list only directory without all files in it. if you want to see all files in it. omit -d

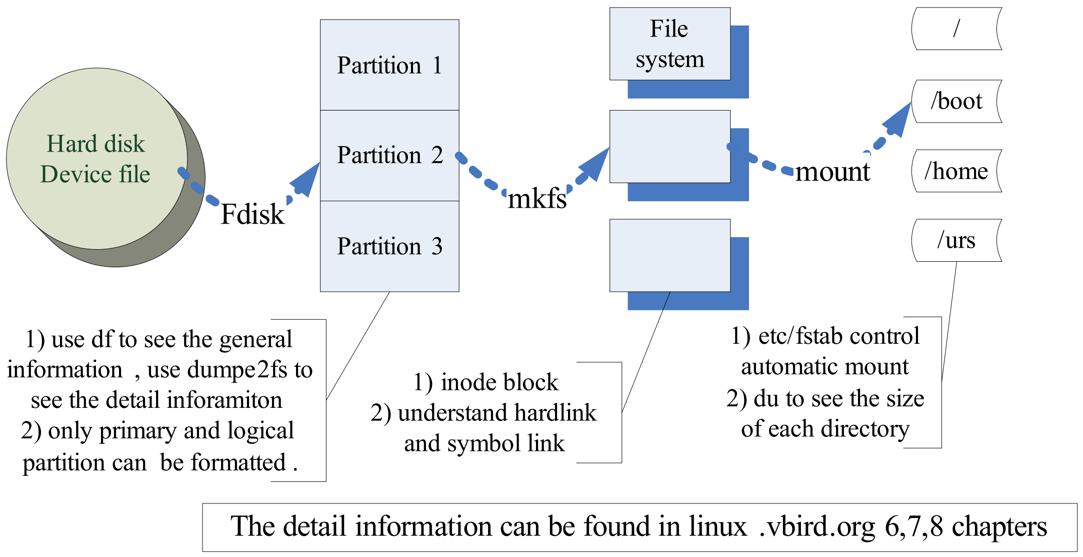
- /usr stands for UNIX Software Resource, isn't users directory. It associate with software. /users includes all the users name, don't confuse them. FHS recommend linux developer install their application into the different dirs inside of /usr: such as /usr/bin and /usr/lib. Don't build their separately directory. For example, when you install codeblock, you can see codeblock exe file in /usr/bin. When you use `$ ldd codeblock ↵`. you can see it use a lot of lib in /usr/lib. There is no codeblock directory which includes everything. It's different with Window system.

- /var include all cache, log, mail which are increased when you system is running. so it will increase with time.

- The Dir Structure:

/bin	includes important comands: mv, mkdir, chmod, cat, chown and date
/etc	Main configuration file: /etc/init.d/ /etc/fstab
/home	home directory
/opt	just like /usr/local
/usr	/usr/local install some your own software, /usr install OS software. /usr/bin, /usr/include(c++ language), /usr/lib(c++ language), /usr/src
/tmp	any body can access or write it.
/srv	web service(www and ftp) access data
/sbin	root command
/proc and /sys	virtual file system, they are stored in memory. proc and kernerl information
/dev	device file
/lib	lib used when you start linux. /lib/modules has kernel modules

0.3.2 partition and mount point


Hard disk
Device file
Fdisk
Partition 1
Partition 2
Partition 3
mkfs
File system
mount
/
/boot
/home
/urs
1) use df to see the general information , use dumpe2fs to see the detail inforamiton
2) only primary and logical partition can be formatted .
1) inode block
2) understand hardlink and symbol link
1) etc/fstab control automatic mount
2) du to see the size of each directory
The detail information can be found in linux .vbird.org 6,7,8 chapters

- A good tool software in linux system is GParted, which is used for creating, deleting, resizing, moving, checking, and copying disk partitions and their file systems. You also can use df command to check your system **partitions**. You can create 1) partitions by GParted 2) mkdir a dir 3) then mount them by adding below text in /etc/fstab file. `/dev/sda3 /home/yan/llvm ext4 defaults 0 0`
- **/etc, /bin, /dev, /lib and /sbin can NOT be put in different partition with root /.**

0.3.3 Permission or file mode

- `$ ls -al ↵` will list all the files ownership and permission. Basically, all the system support `$ ll ↵` command directly.
- There are three different groups: owner, Group, and Other. In each group, there are three different permissions: r, w, x;
- For File and Dir, "rwx" has different meaning. x for Dir means you can come into this dir. For a Dir, if you only have r permission, you only can `$ ls -l Dir ↵`, You can't cp a file from it unless you have x permission. So x permission is very important for a Dir
- `$ chown chgrp ↵` are very useful when you copy files to other peoples. `$ chgrp -R grpName dirName ↵`, you must make sure grpName in

/etc/group file. or it will report error. `$ chown ↵` command need to make sure owner name in /etc/passwd file.

- When you need to use chgrp or chown? When you copy a file to other people.
- `$ chmod ↵` command follow[ugoa][+-=][rwx], for example, `$ chmod u+x file ↵` or `$ chmod go=r file ↵`. There are four letters: u(owner), g(group), o(other) and a(all).**u represent owner, You need to remember that specially**. ("=") means "set the permissions exactly like this.
- SUID or GUID, In simple words users will get file ownerâĂŹs permissions as well as owner UID and GID when executing a file/program/-command. An good example is passwd command, It's owned by root, When you launch it, you have root permission to modify /etc/shade file. That is all!

0.3.4 Commands about File

- `$ du -h ↵` command displays the sizes in kilobytes of all files in the specified directory. df command displays the amount of unused space left in your disk system. `$ du -sh * ↵` will show your every directory size. `$ du -sh * | sort -h ↵` will sort all directory size. Don't use sort -n here, it will put 1.1G ahead 1.2M.
- `$ pwd ↵` return where you are
- `$ cd - ↵` will return the previous path
- `$ diff a.cpp b.cpp ↵` will show line-by-line differences.
- when you run a command, sometimes you need to `$ ./a.out ↵`. "./" means current directory.
- When you use cp, you need pay attention to -a options, it related to maintains permission and owners.
- You can use `$ od ↵` to see the binary file, It reminds me the UltraEdit
- `$ rmdir ↵` only can be used to delete an empty dir. You can use rm -r to delete a directory with files in it.

- `$ tail, head, cat ,tac, less ↵` less will stop, to ask you to continue. Maybe less is better than cat. I often copy-and-paste text from the web into a file like this (command prompt shown):

```
$ cat > filename
<Cmd-V>
<Ctrl-D>
```

- `$ which -a command ↵` will help you find command's name and in all $PATH directory. `$ type ↵` can tell you if a command is bash build in command. Such as cd command. They are used to know more about your commands
- `$ file ↵` is to determined the kind of all files.(executable, text, or data file).
- `$ locate ↵` can help you find a file very quickly, because it just search in an index database. You can use `$ updatedb ↵` to update this database. `$ whereis ↵` just look for binary(-b option) or source(-s option) files. They just used to search binary and source files.

0.3.5 Find command

- `$ find . *.txt ↵` and `$ find . "*.txt" ↵` are different. In the first example, shell will receive *.a and expand it to a.txt b.txt c.txt.... In the second example, find command receive *.a. So the first find command, If in your current directory, there is a.txt. find will not look all *.txt recursively.
- `$ find . "ab" ↵` will not find "abc" file. You need to use "ab*" to get it. A better expression is "*ab*" will find more files.
- In the previous example, you also can use `$ find . \*.txt ↵`. Use \ to stop shell to expand it.
- `$ find . -name "tex*" ↵` can help you find all tex* file from current directory recursively. It's a very powerful command, without -name, it will search all the files. **Don't forget double or single quote around tex*. It will give tex* directly to find command. if you don't do that, shell will expand it by itself. And it will not search recursively.**

- `$ find . -iname "Abc*"` ↵ will ignore letter case.

- You can find according to **name, type ,size, owner and time.** You also can use logic or operator. Below are some useful examples.

  1. `$ find ./dir -maxdepth 1 -type d -iname "man*"` ↵ -type can be b(block), c(characer special file) d(directory), p(pipe), l(symbolic link) s(socket), or f(plain file).
  2. `$ find ./dir !(-not) -iname "man"` ↵ use ! or(-not). ! or(-not) means to exclude "man"
  3. `$ find ./dir -name '*.php' -o -size +20M` ↵ Default it AND, if you want to use OR, use -o.
  4. `$ find $HOME -ctime -2 -name "*.cpp"` ↵ -ctime(-cmin) +n|-n|n: Find files that were changed more than n (+n), less than n (-n), or exactly n days ago. cmin is minutes. About ctime and mtime, can be seen previous explanation.
  5. `$ find . -size -50M -size +20M` ↵ c:bytes, k:kbyte, M:Mbyte G: b:512-byte blocks. Find all files smaller than 50M and bigger than 20M.
  6. `$ find . -user yan -group UH -perm 644` ↵, see -user, -group and -perm.
  7. for symbolic link, you can see -H, -P, -L options in find command man page. default is -P, means that Never follow symbolic links.
  8. Sometimes, find will output "Permisson denied". You can use below command to filter these message.

     ```
     in bash:
     find / -name art  2>&1 | grep -v "Permission denied"
     in tcsh:
     find / -name art |& grep -v ".."
     ```

0.3.6 Grep command

- `$ grep -r -w -i "match" file1 file2` ↵.**Put files name in the end. That is basic pattern. If it's a directory, add -r after grep.** Such as, `$ grep -r 'word' .` ↵

- -n will give match line number. Once you know line number, you can vim +num file to open it.

- -v inverse result. -l will suppress your output. It will only output file name. -l will help you to deal with file with some pipe command, such as copy or rm files with some match words in them.

- A, B, C use to print line around match. `$ grep -B 2 -A 1 'computer' file ↵`

- `$ grep -include=\*.{c,h} -rnw -e "pattern" /path ↵` -w stands match the whole word. -l (letter L) can be added to have just the file name. This will only search through the files which have .c or .h extensions. Similarly a sample use of –exclude:

- –include usage will recursive search all the sub directory. You can use *.txt, but it only works on current directory.

- export GREP_OPTIONS='–color=always' will give grep command color output.

- `$ echo i* ↵` or `$ ls i* ↵`, first, shell will deal with * symbol first. Look for all filename which begin with letter i,**It will not replace * with all file names and add i in front of it**. If it fail, it will print out "No match"

- `$ echo -include=* ↵`, just like previous example, it will fail to look for filename begin with –incl.. So it will print out No match. If you use `$ echo -include=\* ↵`, it will print literal string out.

- `$ echo -inclue=\*.{a,b} ↵` will output –include=*.a and –include=*.b

- From all previous examples, if you use `--include="*.{c,h}"`. It will prevent shell to expand * and big bracket. It's not right. For new version grep, You don't need double quote. –include will prevent shell expand. But for old grep, it doesn't work. I recommend to use `--include=\*.{h,c}` to escape * symbol. It work on both new and old version.

- `$ grep 'word' *.txt ↵` will just look for txt file in current directory. `$ grep -r -include=.txt 'word' . ↵` will look for all .txt file recursively.

- -c will only print match number, -C3 will print context 3 line information .

- Since you usually type regular expressions within shell commands, it is good practice to enclose the regular expression in single quotes (') to stop the shell from expanding it.

- grep support regex, You need to use -E option to support extended regex. `$ grep -E 'abc{1,2}d' file ↵`. That's very power feature. Here, I just list a few basic usage.

- -e and -E follow regular expression. -e is basic version. In basic regular expressions the meta-characters `?, +, {, |, (,)` lose their special meaning; Instead use the backslashed versions `\?, \+, \{, \|, \(, and \)`. **I recommend to use -E option.**

- basic regex syntax:

sytax	example	description
^ (Caret)	'^smug'	'smug' at the start of a line
$	'smug$'	match expression at the end of a line
\ (Back Slash)	'123\$' Just looke for '123$' in file.	turn off the special meaning of the next character, as in ^ and $ {.
[ ] (Brackets)	'[0-9][0-9]' pairs of numeric digits	match any one of the enclosed characters, Use Hyphen "-" for a range, as in [0-9]. [^] means excludes
. (Period)	'^.$' lines with exactly one character	match a single character of any value, except end of line.
* (Asterisk), ? , +		match zero or more*, + (pluse) is one or more, ?(question mark) is zero or one occurrence.
{x,y}, {x} {x,}		{x,y}match x to y occurrences of the preceding; or {x} x occurrence; or {x,} x or more occurrences.

- When you mean match number, `* equal ? and +`

- '[abc]+', '(abc)+' and 'abc+' are different. (abc) means it's a group, and should be considered as whole. abc+ means that + just used on letter c, So it will match ab or abc.

- '<.+>' will math "<car>...</car>". Default it use greedy match. If you want to use lazy match, use '<.+?>'. When you use '<.+?>'. You should use `$ grep -P '<.+?>'` ↵ -P means that use perl language standard. Because only perl support lazy search.

- Usually, grep just match one line. If you want match multi-lines. You can use `$ grep -Pnzo 'BLOCK(\n|.)*?END_BLOCK` ↵. Pay attention it use lazy match method. and . doesn't means newline, you need to use '(\n|.)*?'

- By placing part of a regular expression inside round brackets or parentheses, you can group that part of the regular expression together. This allows you to apply a **quantifier** to the entire group or to restrict **alternation** to part of the regex. Only parentheses can be used for grouping. Square brackets define a character class, and curly braces are used by a quantifier with specific limits.

- Differences between grep and find:

  1. "grep" put file names or directory name in the end, "find" just use directory after the find command.
  2. "find" don't need -r, it find files recursively automatically, grep must use -r if you follow a dir
  3. grep will partial match, find will not partial match unless you use *.
  4. `$ grep -r -inlucde=\*.{a,b} -E 'abc*' .` ↵ The first * mush precede a escape symbol, The second * is regular expression, stand for match zero or more.
  5. find command pattern: **find dirname expre1 [or] expre2**, expre1 is -name '*.txt' expre2 is -ctime -2

0.4 User

- `$ cat /etc/passwd` ↵ will tell you all the users and which shell they are using.

- Don't log in root, you can use `$ sudo` ↵ follow your command

- `$ whoami` ↵ tell you account information.

0.5 processes

- CTRL-C aborts the app, CTRL-Z **suspend** app and put it to background, you can use bg command to re-continue this job on background. CTRL-D is EOF. **C is cancel, D is end.**

- `$ ps -l ↵` and `$ ps aux ↵` are two common processes check command. ps aux will product a lot content, so we often use `$ ps aux | grep user ↵`. In status column, S is sleep, T is stop, and R is run. + symbol means it's in foregound.

- `$ ./hello & ↵` will put hello to background. Another useful commands are `$ fg bg, job ↵`. **They just belongs to this bash**. You can use ONE bash shell to do multi tasks. They are very helpful when you are login sever with ssh command. At this time, you only have one terminal window. So multi-task is import for you. If you work on local workstation. You can open another terminal windows easily, These commands are not very helpful for you.

- View all the background jobs using jobs command, If you have multiple background ground jobs, and would want to bring a certain job to the foreground, `$ fg %2 ↵` will bring the job#2 and `$ kill %2 ↵` will kill it.

- Ctrl-Z will suspend process. If you want to continue, you need to 1) use jobs to know its number, 2) use bg %num to restart it in background. 3)If you want to bring it back to foreground, use fg %num.

- `$ nice commandname & ↵` means that you run command friendly with other(not occupy all resources) and run it at background.

- `$ htop ↵` will show processes dynamically.

- `$ kill -s singal PIDnumber ↵` will send signal to processes with PIDnumber. **This command has some confusion with its name.**

- `$ kill -l  ↵` will list all available signal, The default signal is TERM which allows the program being killed to catch it and do some cleanup before exiting. A program can ignore it, Specifying -9 or KILL as the signal does not allow the program to catch it, do any cleanup or ignore it.**It should only be used as a last resort.**

- `$ ps -f -forest ↵` will show all the processes in hierarchy. Just like pstree.

- I launch a background process, either by appending "&" to the command line or by stopping it with CTRL-Z and resuming it in background with "bg". Then I log out. We were quite sure it should have been killed by a SIGHUP, but this didn't happen; upon logging in again, the process was happily running and pstree showed it was "adopted" by init. But then, if it is, what's the nohup command's purpose? Below is answer: For BASH, this depends on the huponexit shell option, which can be viewed and/or set using the built-in shopt command.

0.6 Application

0.6.1 Internet

- You can google "where is my IP address" to get you external IP, or use `$ ipconfig ↵` to know your internal IP. `$ host ↵` can know IP or host name from each other. Another Interesting tool is `$ netstat ↵` can tell you what connections are there in your computer. and `$ traceroute ↵` can trace the path in connection. They are some useful Internet connection tool.

0.6.2 VNC server

- It can help you to get remote linux desktop, It's very helpful for a windows programmer to use GUI in the linux host.

- vncserver -help will list all the command optoins.

- You need to install VNC server in the linux with su priviliage, then run vncserver on the linux machine. Last on you windows machine, you can use putty and vncviewer to access the remote desk top. Detail can be googled, there are some reference pages available.

0.6.3 clipboard

- There are two kinds of clips: 1) normal clip, Ctrl+C copy and Ctrl+V paste. 2)Xclip, shift+mouse left button select+copy. Mouse middle button to paste. Most linux system support both. But they are two different clipboards. If you use Ctrl+C command, You can't use mouse middle button to paste it.

- `$ xclip -o ↵` will output content in xclip. Not normal clip.

- You just try xclip in shell to see if it's installed. if it's not installed, you can use `$ sudo apt-get install xclip ↵`. If you use Mac, in VMware fusion, you need to configure the Linux mouse setting. default is command+primaryButton to simulate middle button. xclip is very useful in linux, it support copy from a vim and paste to another vim.

0.6.4 Installing

- There are two kinds of installing method in linux, one is install from source code, The other is install binary package.

- By now, There are three main install methods from source file, one is make and the other is cmake, For cmake, you should use "Out of source" build. Another is use autotool, but this method is a little obsolete, don't study and use it any more. For some simple project, you can use make file directly.

- If you install from source code, you need to download tarball, then read INSTALL or README(option), then run config to produce makefile, last run make and make install.

```
./configure --prefix="$HOME" --build=x86_64-unknown-linux-gnu
make 2>&1 | tee make.log (bashrc)
make |& tee make.log (csh)
make install
```

- As root, you should put install a application under /usr/local. If you don't have su privilege, use `$ -prefix ↵`. It will compile source first, when you make install.

- You should not specify app name in prefix directory. Linux will put all execute binary to $HOME/bin, and lib to $HOME/lib or $HOME/lib64. If It's development package, it will put header file to $HOME/include, and document file in $HOME/share. So just use `--prefix=$HOME`.Then add $HOME/bin to your path in .bashrc file.

- ./Configure use Makefile.in to produce Makefile. They are a set of automatic tools. You can see them in c++ web directory, but they are a little complicated, Kdevelop also use them. Just know them. Below is an example to install Perl.

- If there is no configure execute in your directory, you can use autoreconf to generate it. Basically, The first thing to do is read INSTALL and README two documents.

- install some perl programme, If you want to install in your own directory, you can add PREFIX. That will assure you have permission on it

  ```
  perl Makefile.PL PREFIX=\storage\yzhao
  make
  make test
  make install
  ```

- "tee" command is used to store and view (both at the same time) the output of any other command.

- Before you install package, you can use `$ md5sum ↵` or `$ sha1sum ↵` on the package to get fingerprint, then compare your fingerprint with official one on the website to check the files validation.

- There are two main binary installing method RPM+YUM(online update) and dpkg+APT(apt-get). CentOS uses the first, and Ubuntu uses the second. Detail can be found in Vbird linux book.

- You can download the .deb files and use 'dpkg-deb -x' to extract them underneath your home directory. You will then have a lot of "fun" setting the PATH, LD_LIBRARY_PATH, and other variables. The more complex the program or app you're installing the more fun you'll be up for :) So this is your last resort if you don't have su permission.

- Recently, more and more programme has been put into github. Such as xclip, So you can first google some application name. Such as xclip, google it and found git webaddree https://github.com/astrand/xclip. Then git clone gitaddress on you Download directory. After that, you can use configure and make....

- Some application has pre-compiled portable binary package, such as double commander. On it's homepage, you can find file doublecmd-0.7.8.gtk2.x86_64.tar.xz. You need to know two things, 1) You GUI is gtk or qt? 2) You computer is 32 or 64, then download right package for you.

0.6.5 Double commander

- About how to install, see previous installing section.
- You can google solarized theme double commander to change it to dark theme, although it's not as dark as your terminal windows, but it's much better than white background.
- It support file compare function. It is in File/Compare by contents. You can configure it to your favorite external tool, Configuration/Options/Tools. such as meld.
- You can use tabs in Double commander, It can help you manage more directories at the same time. You also can use Ctrl+tab to switch tab. Notice, it's not Alt+tab.
- By now, I use gvim as Double commander's default editor, and meld as file comparison tool.
- command used key shortcut.

key	**action**
ctrl+p	Place path in command combo box
ctrl+enter	Append selected item to the command combo box
shift+f2	switch to command combo box
ctrl shift + x	copy file Name
ctrl shift + c	copy dir+file
ctrl+L	calculate dir size
ctrl+r	refresh
alt+enter	show property
F9	terminal
alt+f7(Commands)	search in files(grep)
ctrl+s	search file name
F2	rename
ctrl+H	dir history
ctrl+command+right	show dir right
ctrl+home	home directory
ctrl+Pageup	parent directory

0.6.6 Other

- **First, adjust system font, It will make all menu and window font bigger, Second, for specific application, such as terminal**

, double commander and chrome, You can change it's font by preference.

- There is Dark reader extension for chrome browser. **There are three applications: 1) terminal(vim)+solairized theme 2) double commander+solairized theme 3) Chrome browser+Darker reader**. Most of time, They are quite enough for my development career.

- To make less show Chinese, `$ export LESS=-isMrf ↵` I don't know what it means?

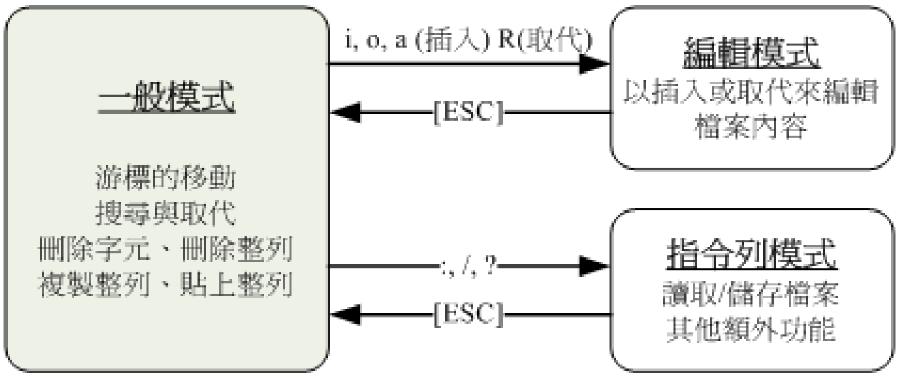
- usually, virtual box think right Ctrl as default host key, it's not convenient in linux, because most of move command need right Ctrl, so you need change it. In window, run VBoxManage.exe setextradata global GUI/Input/HostKey 165 can change it to right Alt. Here, I need to explian, the 165, it's virtual keycode defined by microsoft. you can find detail in google. Now I change it to Win_L, value is 91.

- There are AltGr key to input multi-language character, but I don't need it by now, according to my laptop layout, I need to change it to Alt, so I can use move command shortcut. and define win_menu to Ctrl. I finish it as follow:
  1) use `$ xev ↵` get keycode, AltGr is 108 and win_menu is 135
  2) create your own .Xmodmap and write keycode 108 = Alt_L
  3) in .bashrc, add some statements
  xmodmap -e "add Contrlo = Menu" (this statement is very important)
  xmodmap -e "keycode 133 = Control_R"

Chapter 1

Vim

1.1 basic



1.1.1 Basic knowledge

Configure a new computer

- You should do below steps if you want to install vim on a new computer:

  1. Install new version vim in ubuntu 16.04. In this way, you vim will suport "+, *" two registers.

     ```
     sudo add-apt-repository ppa:jonathonf/vim
     sudo apt update
     sudo apt install vim
     sudo apt-get install vim-gtk
     ```

  2. Run `$ git clone https://github.com/VundleVim/Vundle.vim.git /.vim/bundle/Vundle.vim ↵`

3. Go to my github, download my own vimrc and change it to .vimrc. If you use unbuntu, you can download ubuntu_virmc file and name it to .vimrc
4. go to my github and download vim/ycm_extra_conf.py and rename to .vim/.ycm_extra_conf.py. YCM will use it.
5. go to my github and download vim. Then copy three child directories "after", "plugin" and "macros" to .vim directory.
6. Open a vim and then run :PluginInstall
7. Go to YCM homepage to see how to intall YCM. In a new version Linux, this can be done very easy.
8. Google how to swap Esc and Cap key. Some OS support swap ESC and Captive, **I recommend this method.** It's more reliable and just get used to it.
9. sudo apt-get install xclip.
10. sudo apt-get install exuberant-ctags
11. sudo apt install cscope

Basic rules

- **Rule1: Don't use arrow key any more. At first, you always want to use arrow key to move to right position in insert mode. That is not correct. Any time you want to move, just go back to normal mode, then use "hjkl" and other motion command.**I have disabled arrow key in my .vimrc file.
- **Rule2: Once you go to normal mode, you will have a lot of commands which you combines together to finish your task: 1)basic move, 2)fast move 3)delete+(motion,text objects) 4)visual,copy or move. 5)specific edit(r, surrounds commands) After all necessary task,6)you can use i(I),a(A) o(O) to return back insert mode at right position.**
- **Rule3: Most of time, you should be in normal mode, such as after finish one line, return normal mode and press o go into insert mode again.**
- **Rule4: c command is more useful than d command, 1)it will go into insertmode directly, 2)and all the action after that can be repeated by . command**

- **Rule5: Any motion support forward and backward directions. Know (ge b * e w) position. Jump short with f or long or special character, such as comma or stop with \(fF).**
- **Rule6: Learn to use dot command more, detail can be seen below dot section.**
- **Rule7: Learn to use delete command in insert mode more, such as <C-h,w,u>, del , <c-x>s(Spelling suggestions) and <c-t,d>(Indent)**
- **Rule8: :command also support tat auto-complete.**
- **Rule9: Any time you click same key too many time, stop and google to see if you have better command or smart command, practice until you forget it.**
- **Rule10: But don't overdo it. Don't just focus on trick and plugin. If you can't find solution easily, just give up and focuse on your work, not tool**
- **Rule11: I made some mapping in insertmode, but they are just for temporary action to avoid "Esc+one action+i". You should not think that is mainstream operation, just supplement methods. For example, if you want move cursor a lot, first go to normal mode, then use necessary commands,then come back to insert mode. That is better than <Alt+hjkl> in inser mode. If you just want to move one time, please use<Alt+h>.**

Basic knowledges

- **Basic style is three terminals: the first one is for .tex, the second one is for .cpp, and the last one is for make and other**
- Esc is not in good position. You can use <C-[> to exit insert mode. You also can map jj to Esc. You also can map <Cap> to Esc. By now, my configuration support these three methods. **Sometimes, <Cap> will not work, at this time, you should run :!xmodmap -e 'clear lock' command, it will fix this problem.**
- In terminal windows, in edit menu, you can see preference. You can "disable All menu access keys", In this way, you can use alt+f or alt+h key in your vim.

- By now, I configure cursor in different shape in insert and normal mode. By adding two command in my vimrc file. For different version termnial, there are different solutions, just google it when you get your new computer.

  ```
  au VimEnter,InsertLeave * set nocul
  au InsertEnter,InsertChange * set cul
  ```

- `$ sudo apt-get install vim-gnome ↵`, then use `$ vim -g ↵` will support mouse. With mouse, you can click a position to move there. you also can drag mouse to select a block of text. It's easier to use it.

- :h s will give you detail information about each command. `:h i_CTRL-W` will show a command "CTRL-W" represent in insert mode. "i_" is insert mode. "v_" is visual mode. And if you don't use any prefix, it represents all the command in the normal mode. There are two things worth mentioning: 1) Most commands in insert mode is combination key, such as CTRL-?. 2)In the help topic, you can see a lot of other optional insert commands around CTRL-W. You can extend your knowledge by learning other insert commands.

- `$ vim -v ↵` will open read-only files. You also can use less, but less support doesn't motion command.

- `:%!xxd` and `:%!xxd -r` will go and exit the binary mode.

- `vim -version` will show you a lot if useful information. It will tell you if it support some mode or patch. For example, if you see +quickfix, It means that your vim support quickfix mode. If it shows -quickfix, maybe you need to recompile the whole vim.

- .vimrc and .gvimrc are two important configure files, gvim will read .vimrc first, and then read .gvimrc files. You can use `if has('gui_running')` to just configure for gvim, not vim in terminal.

- If you run vim in a terminal windows, terminal windows font and size will affect vim. If you run gvim, You should configure vim font by add to .vimrc files. Detail can be seen in my vimrc file in github.

- set LANGUAGE="en_US.UTF-8" in /etc/rc.conf and LANG="en_US.UTF-8" in /etc/locale.conf, then logged out and logged back in and it worked. My terminal displays unicode properly now. If don't have root right, set these two varaibles in your .cshrc or .bashrc file.

- gvim's color is sharper than vim in terminal. Maybe it use more colors than terminal.
- g and z don't use as any command in normal mode, so they are prefix as combination command, such as zt,zb, gj, gk, gg. :help z and :help g will show you all the commands that sit behind these prefixes.
- `$ vim -u NONE ↵` will launch vim without load .vimrc. It will help you if you have some troubles in your .vimrc file or you want to do some experiments.
- location list window command: open and close are "lop"and "lcl". For quickfix windows, "cw" and "ccl".
- Control-operations: C-A, C-X. It will add or subtract some number in normal mode. You need to press number first, then press <C-A>.

Dot

- Dot "." is an interesting thing in vim:
  1. In normal mode,dot represent repeat last edit command.
  2. In register. dot represent last insert content. If you use dot, it will remember a command before you go into the insert mode. If you use ".p, It just paste your insert content at current position.
  3. In Mark, dot present you last insert position, you can use 'dot to jump back, different with gi. gi will go into insert mode.
- Below command will be remembered by dot command.
  1. insertion : a, A, i, I, o, O
  2. Text changes involving registers: c, C, d, D, p, gp, P, gP, s, S,x, X.
  3. other text changes: J, gJ, r, gr, R, gR, gU, gu, gw, gq, g?, , g , <, >, =
  4. Equivalent of these operations in visual mode.
- in "practical vim" there are three good examples to illustrate dot command. the first one is tip2, use a add semicolon at the end of each sentence.the second one is tip 3, use s to add two space around +. the third one is tip 5, use cw to replace word. **they all follow the same routine, one keystroke for motion, and one keystroke for repeat**

- . just remember the last command in normal mode, so daw is better then two edit command bdw, you can repeat it by dot command later.
- Don't small count if you can repeat. Use count when it matter. Pratical Vim tip 11 give detail explanation.

1.2 Text Block

1.2.1 text object

- Common text objects in vim.

```
aw,iw a word (with white space), innerword
aW,iW a WORD (with white space), inner WORD
as,is a sentence (with white space), inner sentence
ap,ip a paragraph (with white space),inner paragraph

ab,ib a () block (with parenthesis),inner () block
aB,iB a {} block (with braces) ,inner {} block
a[,i[ a [] block (with []), inner [] block

a",i" a double quoted string (with quotes),inner
a',i'a single quoted string (with quotes)
a`,i`a string in backticks (with backticks)

at,it a <tag> </tag> block (with tags), inner<tag.
a<,i<a <> block (with <>),inner <> block
```

- There are "a,i" two big categories. There are "w,s,p", "b,B [", and "double quote, single quote, backticks". In addtion, There is extra "t,<", Just remember 11 group. You can use them in the "d,c,y,v"commands. Especially you can use "v" command to see if scope of these text objects in your text.
- For text object, **delete around, change inside**. Detail can be found in practical vim tip 52.
- **VimTex** plugin will add new text object, such as vimtex add "e" and "c" two text objects.
- **Surrord** plugin add another control scope, such as "s" beside "a" and "i". **Just remember a", i" and s"**

type	text object	surround
",’,‘	(a,i)(",’,‘)	f"(jump), cs"(, ds"
`(,{,[`	(a,i)(b,B,[)	%(jump) ds(,cs("
<p> head </p>	(a,i)(t,<)	%(matchit), dst, cst<h1>, 1) phh(Emmet) 2)select,S,<p>(Sround) 3)select,<C-y>,p(Emmet)
/begin ../end	(a,i)(e,c)	%(matchit), dse, dsc
w,s,p	(a,i)(w,W,s,p)	Add surrounds: csw", csW", ysiw" yss".

1.3 Basic command

1.3.1 Motion command

Move in insert mode

- **Move in insert mode is not mainstream operation in VIM.**

- Move, you should be able to move in both insert mode and normal mode. For some simple move, you should not leave insert mode, that means you can save some time.You need to use `inoremap` to map to alt key, see next item.

- Some consideration about mapping<A-?> in insert mode: (By now, I didn’t use it very often, They are history legacy.)

  1. All <A-*> need to be use `inoremap <A-t> <C-o>gg` command to map to a command in normal mode. In table, I just keep <A-*>, detail mapping can be seen in my .vimrc file.
  2. Don’t change any command in normal mode, when you use laptop or other computer, when your mapping doesn’t work well, you can return to normal mode to use these original commands in normal mode
  3. when you are mapping, you can refer the normal mode, such as <A-w> is move to next word, just like w command in normal mode.
  4. If you just want to have ONE action move or other operation, you don’t need to leave insert mode, so mapping the move used command in insert mode, by now, I mapping some move and editing command in insert mode. Detail can be seen in below table.

5. By now, you can use Alt-char map some commands, and you also can use two captive letter which don't use very often in our language. such as UU RR etc.
6. By now, on some computers, if you map alt-(char), It doesn't work very well. Sometimes, when you press it,it will go into the normal mode. I don't have time to investigate right now.

Move in normal mode

- Some basic motion commands:

move position	insert mode	normal mode
b/e of document	<A-f>(begin)	1G(or gg) , G
b/m/e of screen	<C-o>...	H, M, L
pre/next para	<C-o>...	{ }
pre/next sentence	<C-o>..	()
begin, end of line	<A-i>and <A-a>	(0 or ^) and ($ or g_)
next, previous word(begin)	<A-w>, <A-b>,<A-e>	w, b
next, previous word(end)	<C-o>..	e, ge
match parenthes	<C-o>...	%
match next character	<A-q>, <A-z>	fx, Fx, tx, Tx
mark and return	<A-m> <A-n>	ma, then ‘a.
return to previous jump position	<C-o>...	<C-o>
search	<C-o>...	/ and ?
previous and next word in cursor	<C-o>..	#, *
All /first word in cursor(For C/C++)		[*I*, [*i*
will go to next line in wrapped mode.		gj and gk

- **For left and right two words, and beginning and ending position use w(W) b(B) e(E) ge(gE) command to jump**
- **From 2 to 4 words, you can use f and F command to follow a letter. Prefer to use less common character, such as x,j. Avoid aeiou.**

- **More than 4 words, you need to use easyMotion plugin. It also support w(W) b(B) and f command. It will list all options, but you need click more keyboard.**
- j k 0 ˆ $ can add g in front of letter to distinguish real line and display lines.
- "f F t T" will not go over multiline. It just search within one line. If you need to multiline search, you can use "/ ?". Or use Easymotion Plugin.
- h,j,k,l, w(W), e(E), b(B), ˆ, $,fx,tx, (,), { ..., gg. All these move command you can add number before them. For example, `4f,` will jump to the fourth ,(comma) directly.
- fx command, then ; and , will repeat forward and backward command, Just like n and N in / and ? Command.
- reuse the last search // and last search in sub s//aaaa/gc
- jump lists basic:
  1. Jump commands are "G","/", "?", "n", "N","%", "(", ")", "[[", "]]", "", "", ":s", ":tag"(When you have tagbar, you don't need run :tag manually), "L", "M", "H"
  2. "w b e j k h l" are not jump command, they will not recored in jump list. "f, F, t, T" are not jump command either. **With in one line isn't jump command, You can think it in this way although it's not 100% accurately.**
  3. When you use easymotion, jump more than one line, it's regarded as jump.
  4. All jump command will record in jump list; :ju will show jump list. Then you can use <C-o> and <C-i> to navigate the jump list.
  5. gi will return last position you exit insert mode.
  6. <C-o> will return back last jump positio, <C-ˆ> will switch back to previous file.
  7. Double single quote and double backtick will toggle current and previous jump. Double backtick will come back exact position. Double single quote will come back previous line position. **If you want to jump to one place, then jump back, these two**

commands are very useful. You don't need use m to mark the original position and use ‘ to jump to mark position.

Mark

- marks lists basic:

  1. :marks command will show you all the marks list.
  2. Basic command is ma and 'a.
  3. Low case letter just mark position in current file. Captive letter can mark a position in different file, when you jump to this position, you don't need to open this file manually.
  4. There are single quote command, backtick command. Single quote will jump to exact mark, backtick will jump to mark line. They need follow a mark name.
  5. \`. will return your last edited position. \`" will return exit position of current buffer. And \`0 will return exit position of file(when vim exit, That is to say, The last file you are working when you exit your vim last time.)
  6. Double single quote command will help you to jump back origin position, but still in normal mode, **gi command will return back to the position where you exit insert mode, and return back to insert mode. It's very useful command.**

Search

- search command in VIM:

  1. `$ :lv word % ↵` will search all word in current file. It will show all the result in position list.
  2. position list is associated with current window, and quickfix list is global.
  3. You can use :lne :lw :lpr to manipuate position list windows. You can use :cne :cw :cpr to manipuate quickfix windows.
  4. If you want to find all key words in the current file, you can use :lv. If you want to find all words in the multi files in one project, such as a C++ project, you can use \vv command, It's vim plugin. It will show all your result in quickfix window.

5. You can use # and * command, you don't need to input word which you want to search.
6. If you use / or ?, you can use <C-r> <C-w> to paste current word to command windows. or you can input manually.
7. **Press F4 to toggle highlight research result.**

- regex in search command:
  1. vimgrep use // to enclose pattern, but when you use / it's a little different, /abc/e search abc and put cursor to end.
  2. patten example 1) escape html 2) search duplicate word.
  3. \v magic; \c case incentive
  4. There are two subtopics, One is regex syntax is different between grep, python and vim. The detail can be see below link. https://remram44.github.io/regex-cheatsheet/regex.html#syntax-basics And a good regex example can be found here: http://www.rexegg.com/regex-quickstart.html
- Three methods search: 1)"/,?" 2):lv 3) easygrep plugin
  1. / search current buffer, it support regex too, but it will not output result to position list or error list.
  2. :vim(grep) and :lv(imgrep) difference lies in write to location list or quickfix list.
  3. :lv need follow % to represent the current file, or *.c which means all c file. Or you ca use ***.c which means recursive. But a better way is using easygrep plugin. You only need \vv command.
  4. \vv is just syntax sugar of :vimgrep. You don't need input word manually, and it will open quickfix window directly. The shortcoming is you can't edit regex manually.
  5. For :lv,You can input complex regex manually, So :lv is more flexible than \vv. \vv is more convenient than :lv.
- :marks :ju :reg :change are three useful commands.

1.3.2 Edit command

- Below are some edit commands:

delete	insert mode	normal mode
delete previous character	backspace,<C-h>	hx(move then x) or X
delete and replace current character	<A-x>	x , s , r{char}
delete word to end	<A-c>	(d,c)e or w
delete word to begin	<C-w>	(d,c)b or ge
delete current word	<A-b>,then <A-c>	daw or daW
delete line to end	<A-v>	D or C
delete line to begin	<C-u>	d^ or c^
delete current line	<A-d>	dd S cc
replace in current line	<C-o>..	:s/old/new/g(c)
replace in whole file	<C-o>..	:%s/old/new/g(c)
indent and unindent	<C-t> <C-d>	» and «
Join next line	<C-o>..	J
swap line with next	<C-o>..	ddp
undo and redo	<A-g>	u and . (ctrl+r)

- tricks about edit command:

  1. Edit commands "d,c,y", **d keep in the normal mode. c go to insert mode. y keep contents.**
  2. **d,c,y can all follow EasyMotion command.**
  3. C delete to the end of line, and go into insert mode. D is the same, but stay in the normal mode. S delete the whole line, and then go into the insert mode. X delete previous character. **remember(s,x), (C,D) and (S,dd).**
  4. d c y > < = g gu gU are operator, They can follow motion, and operator+conquer.
  5. All previous operator can follow motion, it give edit command a "operator scope". For example, "d2w" will delete next two words. "d3j" will delete next 3 lines. "ct;" will delete every thing until ";", it's very useful for C++ language.
  6. At the same time, all operator also can follow text object, such as "aw" and "iw". A list of text objects can be see below. You can use `di"` to delete all the contents inside of a pair of double quotes. You also can use `da"` to delete all the contents plus a pair of double quotes.There are 11 groups common used text objects which I introduced in previous section.

7. 3dd and 2dw & support number+command means how many times you will perform this command.
8. <C-h> <C-w> <C-u> and del, these four commands can delete in insert mode. Learn to use <C-h> more, It's easier than type backspace.
9. <C-r> and <C-r>= paste in the insert mode.
10. After select, use c to delete and into insert mode.
11. x stay in normal mode, s go into insert mode, r need to follow a letter and stay in normal mode too.
12. **c is same as d, and s is same as x, but they will go to insert mode, you should use them more, to avoid use i, a later.**
13. command v also can follow motion and text objects. It can give me a tip. If you are not sure what you will delete, you can use v command to highlight, if highlight is not what you expect, you can Esc, and adjust you motion or text object, until you get what you expect.
14. ggyG will select all contents in a file. such as Ctrl+A and Ctrl+C
15. "ddp" swap lines. In fact, it is two commands, "dd" and "p". If you understand it. you can know "xp" is swap character. And "dawwP" is swaping word.
16. "J" can be use to delete empty line below.
17. **When you use delete to a character at long distance, You can use d*backslash*[fF] to give exact position, Or you can use df, than use dot command to repeat it.**

1.3.3 Scroll

- scroll command, scroll window, but cursor will stay in the same position.

scroll one page	<A-t> and <A-y>	ctrl+b and ctrl+f
scroll one line	<A-u> and <A-r>	ctrl+e and ctrl+y
move cursor to m/t/b	<C-o>...	zz zt zb

- **zz command use more often when you want see more.**
- **Don't use j,k command to see more content(turn page, or scroll).**

1.3.4 Window

- Control windows commands:

move	insert mode	normal mode
switch window	<C-c>	Ctrl + w{h,j,k,l}
split window	<C-o>,then :sp	Ctrl + ws
close a window		Ctrl + wq or <C-x>
close all other		<C-w> o
split window vertically		:vsp
window resize		<C-w> [count]+,-
window vertical resize		<C-w> [count]<,>

- <C-w>,o is more useful when you want to close other windows, You don't need switch to another windows.

1.3.5 Buffer

- Manage buffer:

Action	insert mode	normal mode
edit a file in a new buffer		:e file
next, previous buffer	RR	:bn :bp , <C-ˆ>
go to number buffer	<C-o>..	:b num
delete a buffer		:bd
list all open buffers		:ls
open a file in a new buffer and split window	<C-o>..	:sp file or :vsp file

- Closing windows is different with deleting buffer. When you use :bd command. It will close the windows too. If you want to just close buffer. You should use :bn to switch to another buffer, then use :ls to know the buffe number which you want to close, in the end. use :bd num close to close it background without close current windows.

- use :ls! command will show all the buffers. Such as Nerdtree buffer, VimtexToc buffer. To reload buffer into certain windows.

- `:bd` will close current buffer, you can follow number to close specific buffer. You also can use `:bd+space+tab` will list all the avaible buffers.

- Any time you want to switch buffer or open new buffer, save current buffer first.
- ":e" will open buffer in current window. If you want to open in a different window, you can use "sp file" or "vsp file".
- Open another buffer in splite window, `:sp | b1` Pay attention, it's different with :sp file-name.
- Usefule commands:
  1. Open at current windows, 1):e file 2) <C-n> seletct file and enter.
  2. Open at split windows, 1):sp file 2) <C-n> select file and i
  3. Open at vsplit windows 1):vsp file 2) <C-n> slect file and s
  4. For CtrlP plugin, <C-p>, then <C-o>, you can select which window to open.

1.3.6 Visual

- Basic commands are below:

v, V, Ctrl-v	being select
d, y	copy and delete
<,>	indent left,right
V, yy	select, copy the whole line
gv	reselect previous select
o	toggle free end of selection

1. Using v to exit select mode is faster than Esc.
2. o command change free end, make you can select text more freely.
3. **Almost all the motion commands can be used in visual mode.**
4. **You also can use EasyMotion command in visual mode.**
5. <,> will exit select mode, gv will select it again.
6. . can repeat visual edit command.But it act on the previous selected region.
7. **prefer operator+motion than visual, because it can repeat.**

1.3.7 Register

- You can check all register by :reg command
- gP is useful when you want to paste multi-line text several times. See "Practical VIM tip 62". Captive P and gP can be used in this contidtion. When you use "g", The curser is near orignal one. When no "g", the cursor is near copied region. That's all.
- :reg will show your list of register."0 will always have the content of the latest yank, and the others will have last 9 deleted text, being "1 the newest, and "9 the oldest.
- " will change when y and delete, and 0 will save y. and 1 9 save large block delete.
- Default register is ", so p command is equal ""p .
- **". save all your content that you just insert.** So if you want to repeat some line, first, you can insert, then use ".p . Then in the end, use . to repeat previous command.
- "ay will copy some thing to "a" register, "ap will paste it.
- "Ay will appnd something to "a" register.
- Unnamed register, yank register, named register(a-z) expression register, only read register "% # . : /" five registers.
- paste from register, include character-wise and line-wise
- In Vim, there are three clips:
  1. **Basically, there are three different method to paste, 1) vim register 2) normal clip, 3) xclip**
  2. normal clip, You only can use mouse to select(It will add line number into your selection), then in insert mode, right-click mouse to popup menu, select paste; Or shift+insert. **Don't use Ctrl+C and Ctrl+v, and must do it in insert mode, or all you content will be input as command in the normal mode, It's dangerous**
  3. xclip, Use mouse middle-button to past in Vim.**Must in insert mode, and It's not good method.** When you use xclip, you should click left mouse first, then click shift key. When you finish

select, it will go to xclip, then you can use F7 to paste in VIM in normal mode, it will keep format.

4. vim own clip, use y and p command. **Must in normal mode**
5. The contents of a register can be pasted while in insert mode: type Ctrl-r then a character that identifies the register. For example, Ctrl-r then " pastes from the default register
6. Use `$ vim -version ↵` to see if vim has been compiled with clipboard, If there is + symbol before it. You can use "+y and "+p to copy and past selection to system clipboard. If not, Just use mouse middle-button.
7. Sometimes, when you copy source code from browsers, then use shift+insert to paste it, The format will be messed. You should use shift+mouse to select and copy content to xclip. Then in your vim, 1) move your cursor to insert position, 2) in normal mode run `:r!xclip -o<CR>`, it will paste according to right format. Don't use normal clip and xclip in insert mode.
8. Add key map to .vimrc, detail can be found in .vimrc file.By now, I map it to F6 and F7.

1.3.8 Format

- format command : Other reformat tricks can be found in next section "Programmer tips"

=, =%	format, format{..}
==	format current line
15G=25G	will reformat from 15 to 25 lines

1.3.9 Ins-completion

- Basic help information can be seen in `:h ins-completion`.
- In order to reduce number of options, you should input two or three letters firstly.
- Ctrl-p, Ctrl-n will finish word previous and next part. It's useful for programming. Usually, Ctrl-p will pop up previous options, they are very useful in programming because you don't need remember previous long variable name.
- Ctrl-x, Ctrl-l will finish previous line. It will list all previous lines first.

- Ctrl-x, Ctrl-f will finish file name. Ctrl-x, Ctrl-n will insert key words.
- Ctrl-x, Ctrl-n will list all the words in the current file, by now, this functions is not very useful for me.
- Ctrl-n,p will navigate all the items in the pop-up menu and accept it. Up and down just navigate. And Ctrl-e close pop-up menu, and Ctrl-y accept the current item.
- Ctrl-x, Ctrl-k will finish word by dictionary.
- Ctrl-x, Ctrl-i insert keywords in current and included file, Ctrl-] will insert tags, You need produce a tag file. Ctrl-d will insert definitions or macros. They are very useful for C/C++ and python programming.
- :set spell , then [s, then , zg it's right word, zu is not right word. z= will list all the option words. :set nospell to close it.
- In insert mode, <C-x>s will jump back to the first error word, and pop up spell correction windows.

1.3.10 Ex command

- **tab also can help you finish when you type :command.**
- ex command strike far and wide
- For ex command, ".", "$", "%" , "'<" and "'>" represent different [range], "." is current line, and "$" is the last line of the file. and "%" is the whole file.
- **:%normal i**// while file, perform a normal mode command, Here is insert // .
- @: repeat last ex command.
- :shell will open shell window, you can run some shell command, then input exit to close shell window.
- q: will open command-line window, :quit will close it. Sometime, when you want to use :q, but you misstype q:, You will go to the command history windows, At this time, you only can use :quit to quit this windows, or press enter on an empty line.

1.4 Plug in

1.4.1 Vundle

- About vundle usage, I have added a bookmark to evernote. First you need run "git clone https://github.com/VundleVim/Vundle.vim.git /.vim/bundle/Vundle.vim", then just need to edit .vimrc file.
- When you have vundle installed, you can edit .vimrc to install others plugin. By now, I have backup my .vimrc file. When you transfer to other computer, you can use it directly.
- You can google "vim awesome". It introduce a lot of vim plugins. By now, what I installed can be found in .vimrc file. You can download my .vimrc from my github new_doc repositor.

1.4.2 Appearence

Solarized

- vim-colors-solarized, it can be only used in vim in terminal, It only support 16 color. You need to go to vim "color scheme test c website" (add it evernote) and select you favorite scheme. and then download it. put it in .vim/bundle/vim-colors-solarized/colors directory. Then you can modify your .vimrc.

  ```
  if has('gui_running')
  colorscheme blacklight
  else
  colorscheme solarized
  endif
  ```

- When you use solarized plug in(theme), you have to change the terminal. For some new version gnome terminal, you can see terminal->preferences->profiles->color tab, there is solarized dark option in "text and backgroud color", and solarized option in "build in scheme in palette", just select it.
- When you have old terminal, There are two options, for some old gnome terminal, there are no import color option, you can download Anthony25/gnome-terminal-colors-solarized in a temp directory, and then come to this directory and run .install. You need to first add an

solarized profile, then the application will ask if which profile you like to overwrite, select solarized profile, you can keep you original default unmodified. Then you can select your solarized profile, the color will be set properly.

- If some keyword highlight background color is not correct, you need add `set t_Co=16` to your .vimrc.
- For mac, you can download tomislav/osx-terminal.app-colors-solarized(I haven't tried it yet). Or you can download git:altercation/solarized. There is directory `osx-terminal.app-colors-solarized/`, you can import *.termnial file in your mac terminal preference color. That is all.

vim-airline

- You can go to "https://github.com/powerline/fonts" download font package. Then extract it. Go into the directory, then run ./install.py. All the font will be installed in HOME/.local/share/fonts. Then you go to your HOME directory. Build a link. "ln -s HOME/.local/share/fonts .fonts". Then you gnome-terminal preference window will show Powerline fonts. Select "Literation Mono Powerline". Building link is very important step for gnome-terminal.

```
if has('gui_running')
set guifont=Literation\ Mono\ Powerline\ 12
let g:airline_powerline_fonts = 1
endif
```

1.4.3 File explorer

NerdTree

- **Use ctrl+N to toggle nerdtree window**, then Ctrl+C change window, once you are in nerdtree window, you can use all motion command in normal mode to move your cursor. You also can change to visual mode and use 'y' to copy a file name and paste it back to your main editor file.
- **shift+i will show or hide hidden file.**
- When in the NERDTree window, press 'm'; you should see a menu at the bottom. Type in 'a' for add childnode. Now input the directory

you want to create, making sure to add a '/' at the end, otherwise the script would create a file.

- In the NERDTree window, you can press 'm' and 'l', then you will get whole directory informaiton. Then you can use mouse to copy it and paste it back to your main edited file.
- When you are in the Nerd window, you can press Enter to open in current windows. "s" will open in vsplit window, "i" will open in a split window, If you forget, press "?" to toggle to show all the useful commands.
- For some very deep file, you can use :Bookmark bm. Then "B" will show all the Bookmark table. Then you can navigate Bookmark table to open a bookmark file.
- You can use "u" to jump up one level in directory tree. Or use "C" to change current directory as tree root."r" will refresh the directory.
- By now, preview will open a new buffer inside vim. So this feature is not what it really like. There are some solutions, but I don't want to try them right now.
- use p to jump to parent folder position, and .. go up level

Ctrlp

- use ctrl+p in normal mode(not in insert mode) to invoke ctrlp plugin, Don't use :CtrlP command

ctrl+f,b	cycle between modes
ctrl+r	switch regex mode
ctrl+d	switch to filename search instead of full path
ctrl+o, t or x	open, open file in new tab or in new split
ctrl+z	mark and unmark files

- CtrlP will open "current working directory" in vim. You can use :cd directory to change "working directory". You also can input .. to go up parent directory. Default, CtrlP also look for .git as a flag of "Whole project". You can custom your favorite setting according to your context.
- CtrlP also provide "most recent files" function.

- **With Ctrlp and NerdTree, vim has powerful tool to deal with files and directory. You can navigate file system structure by Nerd, add mark by Nerd, or search them quick by CtrlP**

1.4.4 Motion

Easy motion

- Jump to word, use <leader><leader>w,b,W,B,e,E,ge,gE.
- For multi-line, use <leader><leader>j,k.
- :help easymotion.txt give you detail information.
- **f command is very very usefule!**.So I map it just one <leader> before f command.
- If you want to jump to long distance, <leader>f is your best friend.
- You also can try <leader><leader>n,N command, it will repeat you last / command and give each match a tag.
- "<leader><leader>." will repeat the you easy motion command.
- **By now, I map f and F to single <leader>,And by now, It support d and c command**.

Match it

- It's old plugin, so maybe it will not work very well, Don't expect it too much.
- It support <p> </p> jump and /begin /end jump.
- It don't need if..else if.... jump. But you can build .vim/after/plugin/c.vim and cpp.vim. and add below to it. And it will support if..else if... else jump and switch...case. default. jump
- g% will jump backward.

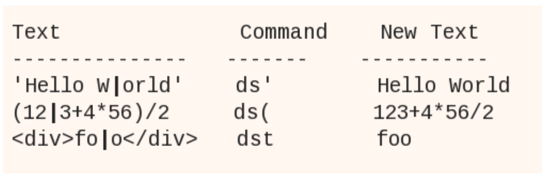
  ```
  let b:match_words= '\%(\<else\s\+\)\@<!\<if\>:\<else\s\+if\>:\<else\%(\s
  ```

- There is another python matchit plugin. It support for(while)...break(continue). if..elif..else, and try..except..finally jump.

1.4.5 Surround

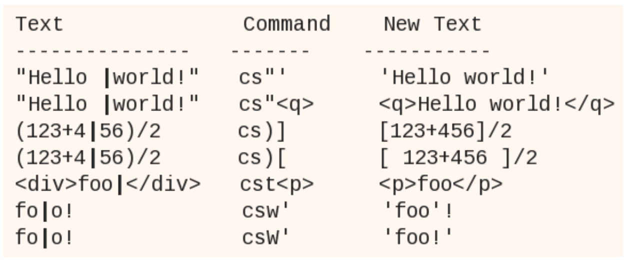
Surround

- There are three main actions. delete(ds), change(cs) or add surrounds(ys)
- When you use delete or change, you should put your cursor inside a pair of surrounds.



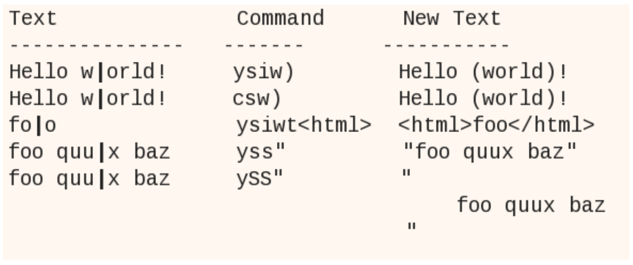
```
Text                Command    New Text
---------------    -------    -----------
'Hello W|orld'      ds'         Hello World
(12|3+4*56)/2       ds(        123+4*56/2
<div>fo|o</div>     dst         foo
```

- below is change command list:



```
Text                Command    New Text
---------------    -------    -----------
"Hello |world!"     cs"'        'Hello world!'
"Hello |world!"     cs"<q>     <q>Hello world!</q>
(123+4|56)/2        cs)]       [123+456]/2
(123+4|56)/2        cs)[       [ 123+456 ]/2
<div>foo|</div>     cst<p>     <p>foo</p>
fo|o!               csw'        'foo'!
fo|o!               csW'        'foo!'
```

- Surroundings can be added with the same "cs" command, which takes a surrounding target, or with the "ys" command that takes a valid vim motion.



```
Text                Command      New Text
---------------    -------      -----------
Hello w|orld!       ysiw)         Hello (world)!
Hello w|orld!       csw)          Hello (world)!
fo|o                ysiwt<html>  <html>foo</html>
foo quu|x baz       yss"          "foo quux baz"
foo quu|x baz       ySS"          "
                                      foo quux baz
                                  "
```

- You can use csw" or ysiw" to add surrounds. ys can follow a valid vim motion. Such as ys3w<p> will add <p>word1 word2 word3</p>
- ys follow 1) text object, 2) f 3) ss.

1. ysiw" is different with ysw". ysw" will just add " in the exact cursor position, but ysiw" will add " around word which cursor is in.
2. ysf;) will add () around before ";".
3. For a single line. You can use yss command. It's better than Emmet. In Emmet, you have to visual select this line first.
4. **If you want to add surrounds to random scope, You can use visual mode, with some easy-motion command to select a scope, after it, you can use "S" to add surround**
5. **By now, EasyMotion support single <Leader>, so you can use ys\f<ch><ta>" to add " around scope, it's quicker.**

- In Visual mode, press <S-s> (captive S). Then input a surround that you want to add. Don't use lower case "s".
- You need install repeat plugin too. For example: there is text "<p1><p2>word</p2><
  You put cursor inside word. Then you can use dst. Then press . It will delete pair of <p1> and <P2>. repeat plugin just for Surroun plugin.
- Surrounds are useful for edit html doc. It's overlap with Emmet. For example. In visual mode. `<C-y>,` will trigger expand mode in Emmet. Then input <h1>. Or you can use <S-s>, then input <h1>. It will add a pair of <h1>

Repeat

- . only repeat edit command in norm mode. And only for these native edit command. It will not repeat move command
- move command are most single character, So I don't need to use . to repeat it. If you like, there is plugin for it.
- For some customized edit command, such as Surrounds below. native . command will not support it. So, I need to repeat vim plugin installed.

1.4.6 Html and text

Vimtex

- Vimtex is lightweight plugin for Vim. I didn't use compile ability right now.

- Vim tex provide another text objects. Such as "ae ie", "ac ic" and "ad id".
- By now, I map F3 to :VimtexTocToggle.This is most useful feature which I used.
-]] in insert mode finish environment automatically.

[ ] command navigate by section.

- (cd)*(se sc sd) six commands.

```
1)\begin{aaabbb}
ac is "\begin{aaabbb}"
ic is "begin"
dsc will delete "\begin{}", aaabbb is left.

ae is whole enviroment
ie is inside enviroment
dse will delete "\begin{}" and "\end{}"
```

- By now, I don't delimiter, it used mainly for math equation. So I don't use "ad id" and "dsd" very often.

Emmet

- <C-y>, is trigger key in Emmet. I map "hh" to expand abbreviation.
- Below is simple table, detail can be google by"emmet abbreviations syntax". Abbreviation is the most important topic in Emmet. Besides, it, It provide other useful command. For practical usage, you still need to come back to look at its tutorial.

html:5	whole page
>,+,^	child,sidling, parent
*	how many times output
div#id, div.class	with id and class to a tag
$	will attach number with elemental attribute name
text { }	add text to element

- An example, type `div>p#foo$*3>a`, The <C-y>, <div> <p id="foo1"> <a href=""></a> </p> <p id="foo2"> <a href=""></a> </p> <p id="foo3"> <a href=""></a> </p> </div>

- Some directly edit commands.

<c-y>+k	remove a Tags pair and content inside
<c-y>+d,D	Balance tag outward and inward
<c-y>+a	move to URL, add anchor to it
<c-y>+/	toggle comment
<c-y>+i	In image tag, refresh image
ctrl+n,N	move to next available position

- This tutorial is very simple, website give you detail examples, you can learn it when you work on html pages.
- You can type p, then type hh, It will expand <p></p> and insert cursor will be in the middle.
- For some exist test, first select, then type <C-y>,, It will move curser to bottom to wait for you input tag.You just need input "h1", don't need to input "<h1>". Emmet will add < > for you automaticlly. If you use surrounding, select, and shift-s. input "<h1>".

1.5 Programmer tips

1.5.1 Basic

- Build a IDE-like Vim, You need below plugins:
  1. File explore: NerdTree and CtrlP.
  2. Tag jump: tagbar, cscope and easyGrep, gD command and Ctrl+](need tags file), Ctrl+t return.
  3. Error jump: quickfix.
  4. Syntax: syntactic plugin(For python and C++, YCM will use Syntax show syntactic errors)
  5. Autocomplete: SuperTab, Vim-snippet, YCM, Ctrl+p.
  6. Debug: pyclewn.
  7. Comment: Nerdcomment.
  8. Fold: For C/C++, just :set foldmethod=syntax, For python language, use simplyFold plugin
- For some large scale C/C++ projects, There are a lot of source code which are in a deep directory structure. You can use below tips:

1. `$ find . -name "*.c" -o -name "*.cpp" | xargs git add ↵`
   to add all source code files to git, Then you can build a branch test. You can apply this methods to all .cxx and .hpp and .h files.
2. For ctags and cscope and doxgen, **you can download my project template cmake-test from github. There are src-tool.sh in src directory, you can use them automatically.**
3. Use :set path, then you can jump to header file by "gf" command, then use <C-ˆ> to return. A more useful command is <C-w>f command, it will open header file in a new windows.
4. First, you need to use Nerdtree to get all directory information, Then set path+=<paste directory here>. In the end, you can use gf command to open header file.
5. Use F8 to open tagbar windows, it shows all global variable and function in this files.
6. If you want to search withing specific directory, you can use easy-Grep tool. Detail can be seen esayGrep section.

- :set number; syntax on and set ai; Whenever you hit Enter to start typing on the next line, vim automatically will indent to the same amount as the previous line.
- Python indent show plugin is <leader>ig. You can change its color if you dislike default configuration.
- Why jumping to brace is so important? First, use previous commands to jump to open brace. Then `=\%` will format the matched scope. `v\%` will select the whole scope. After you select, you can input = command to reformat.
- Reformat will perform action according to the previous lines indents. So before reformat, you can use == reformat the previous line of code to right indent.
- ctrl+v will invoke visual-block, you can decrease indent of block or delete a block of comments.

  1. ctrl+v, then select block of comment source code
  2. don't press Esc, press I (capital i)
  3. input //
  4. press Esc, then, all the block will be commented.
  5. A better method is to use Nerdcomment.

1.5.2 Fold

- For fold C/C++ correctly, write big bracket after class, function and if.. statement, and put it next line the first column, in this way, you can use [[command to jump to the fucntion beginning quickly. I didn't use fold very often.
- In vim ,there are different fold methods, and you can't combine them together. The most useful Syntax, And if you want to use zf command, you have to change it to "manual" method.
- fold commands are below, they are VIM commands, not plugin command.
- Just remember three common commands za, zo and zc.

key	**action**
za, zo, zc	toggle, open, close
zj,zk	jump to previous and next fold
zM, zR	Close , Open all
[z,]z	move to the start,end fold
zf\unskip	fold to string

1.5.3 Syntastic

- :Errors to open error windows.
- Everytime when you save file, syntax error will listed on left side.
- You can use :lne to jump to next Syntastic error, use :lclose to close Errors windows. They don't show on the quickfix window.
- There are two checker for latex, one is chktex, you can download and install it from source code. The other is lacheck. lacheck will not ignore text in lstlising enviorment, and it will report a lot of errors. It's annoying. chetex can ignore content in lstlisting enviorment.
- You can goto tex document directory, you can build a .chktexrc file and add below to it.

```
CmdLine{
--nowarn 1
}
```

```
VerbEnvir{  verbatim comment listing verbatimtab rawhtml errexam picture texdra
  filecontents pgfpicture tikzpicture minted
}
```

- For Syntax error use :Errors, it open location list. For compile error, use quickfix mode, It use quickfix window. nowarn use to suppress some unnessary warning message.

- You can add `let g:syntastic_tex_checkers=['chktex']` to tell Syntastic use chktex to check latex.

- For latex document, there are a lot of unimportant error, You need to supress them. When you use lacheck, you can't suppress warning. But you can add tex.vim file to after/ftplugin to deal with some tex type file. In this file, you can add let g:syntastic_quiet_messages to it, detail can be seen this file.

- chktex doesn't check unmatch big brace, but lacheck will not ignore lstlisting env. **So, They are not good tool for latex check, just a refence, not perfect at all, Don't spend time on them**.

- default use chktex, after write, jump to Errors windows and search "error", If there is no error, You can use lachek in you command line to look for unmatch brace error if you document doesn't have lstlisting env. That is all.

- **For Cpp latex document, use chktex, in other document, use lacheck and don't use lstlisting enviroment**

- For python language, you need to install flake8. You can use `$ pip install -user flake8 ↵` to install it. It will install it in ./local/bin directory. You don't need configure Syntastic plugin, default it use flake8.

- For html language, You need install tidy, By now, I download it from github and use cmake to build it. detail can be found in build document.

- My web use a lot php file. So You need to add this line to .vimrc. So when you edit .php file, when you save it. It will use tidy to check all the html content in php file.

```
let g:syntastic_filetype_map = { "php": "html" }
```

- :SyntasticInfo will show some helpful informaiton, when syntastic doesn't work, you can run this command to see some detail.
- For C/C++ language, Syntastic use clang to check errors in C/C++ language. If you use YCM, It will trigger syntastic check automaticlly.
- If you instal YCM, for cpp file, :Errors will not work anymore. When you reach error line, error message will be shown in the bottom automatically.

1.5.4 Code navigation

- gD, go to the func or variable defination. It's just jump the first appearance in the current file. It has no any semantic. Another otpion is Ctrl+], it's just like :tag command. It will need you have tags file produced by ctags command. For example, you can use ctags a.cpp to produce tags file, then in you vim, you can use Ctrl+]. The problem is by now, ctags doesn't produce local varaible inside a function. This problem can be resolved by easyGrep partly.
- Program motion:

  1. C/C++ jump bracket:

     ```
     fun()
     [s"[["
        ...
     }"[]"

     main()
     {"[["
        {"2[{"
      {"[{"
        {"?{"
        }
        {"F{"     } *
     }
     {"]["
     .....
     }"]]"
     %\end{lstlisting}
     ```

     (a) * is your position

   (b) `[[,[],][,]]` First is directory, second is opening or closing brace. They just jump to the first column
   (c) Find match open brace, you should use f/F or /or? command.
   (d) `[{ or ]}` will jump unmatch braces.
   (e) `[( or ])` has same usage.
   (f) The above four commands can be used to go to the start or end of the current code block. It is like doing "%" on the '(', ')', '' or '' at the other end of the code block, but you can do this from anywhere in the code block.

2. % can be used jump brace, with Matchit, You can use it jump if else in C and try except in python. You need to another file to support it, detail can be found MatchIt section.
3. Visual Mark is also a good plug in, usage is very simple, mm visual mark, F2 and shift F2 navigate.
4. Tagbar support python and C at the same time. Cscope only support C/C++.
5. EasyGrep is also a good plugin, detail can be found in below plugin part. 1) Just like cscope, it can be used for python file and produce a list reference position, 2) It support replace mode, and it's very helpful for refactory.

- **Tagbar give yo global scope function name and variable, but It will not give you local variable. It support C/C++ and python file**
- **easyGrep only support multi file search, If you need local file, you can use :lvim, It will show a list result in location window, you can use :lopen, :lne, :lpr, :lclose command**
- **cscope support a little syntastic analytic, Anytime when you modify files, you need to produce new cscope.out file and it doesn't support python language**

EasyGrep

- \vv will search buffer default. You can use :GrepOptions turn off it. A common used options is \vyr.
- First, you need to understand cwd. You can check :echo getcwd() to know current working directory.

- All kinds of configuration play import role in EasyGrep. You can use :let g:EasyGrepRoot to check configuration options value. It doesn't need to follow value.

- Usually, a large C/C++ project has deep recursive directory path structure. You should open your vim in the root directory. If you want to find in all files. You should use \vyr to turn on recursive options. After you use \vyr, you should use :let g:EasyGrepRecursive to check if you turn on it.

- Another usefule option is :let g:EasyGrepRoot, default value is cwd. You can use :let g:EasyGrepRoot = "repository" to search all your .git. You should run :GrepRoot command directly, it will list all options and then you can select one.

- **If you want to toggle recursive, you can run :GrepOptions and then press letter r, it's much easier than remember :let g:EasyGrepRecursive name.**

- Usually, Resursive search will take long time, If you just want to find it's defination, you should use ctags to produce tags file. If you want to find calling or caller relationship, you can use cscope to produce tag index files.

- add let g:EasyGrepMode = 2 to .vimrc. It will only look for the same kind of file extension.

- <leader>vv, look for word, <leader>vV match whole word. It will show all the result below the window, just like quickfix mode.

- <leader>va will add find to current quickfix list.

- The most useful feature is <leader>vr, it will find all match word, and ask you if you like replace it with certain target word. It's useful to refactory you code. "y" will accept and continue, "l" will last , "n" will skip, "a" will replace all. ^E ^Y will scroll.

- **You can use NerdTree to change CWD, Then, EasyGrep will find all files in CWD, in this way, you can customize where to search. In :GrepOptions, you can press e command to see which files will be searched.**

Tagbar

- Tagbar can show all the tag(function, variable and macro...) in a different windows. It's better than taglist. But you need to install Exuberant ctags(Use archive manager in mint to install, It's very easy)
- common used shortcut.

s in tag window	change order
space in tag window	show tag prototype in command line
- and + in tag window	fold and unfold
q in tag window	quit tag window

- In Tagbar windows, just like NerdTree window. Press ? will show your basic key shortcut.
- g<C-]> will list all the options
- auto run cags when you save files
- You don't need run any command. Just add it to .vimrc or use :TagbarToggle directly. By now, I mapped it to F8
- Ctag default doesn't parse the local variable inside function, You can turn on it in ctags, but it will make tag file much bigger, and most of time, If you can keep your function shorter, you will not need list all local variable inside a function in the tagbar windows. So default this option is disable.
- Usually, you can keep tagbar window open, it can help you navigate your source code more quickly.

Cscope

- You can use Tagbar and cscope at the same time. It will help you to jump to tags very quickly.
- cscope is another application which you can navigate tags in source code.
- In vim, you don't need any plugin. Just like quickfix. You need to use vim –version to see if there is +cscope in output.
- Run `cscope -Rbq` in you directory which include C source code, it will output three files. One is cscope.out. It's basic index file.

- Default, cscope only scan C file. If you want it to scan C++ file. Build **cscope.files** and add all your C++ file into it. Or you can use find command to write c++ file name into this cscope.files.`$ find . -name '*.cpp' > cscope.files ↵`
- :cs add, then press tab, It will loop previous files. select **cscope.out**, then enter.
- Use below commands to look for tags `:cs find`
- <C-R> and <C-W> you can paste current word into command line. <C-R>" paste you just yank content.You must press " after <C-r> at once, don't wait " appear. Pay attention to, all these command, you have to be in command line mode by typing :, The cursor will go to command windows.
- Download cscope_map.vim, and put it in the .vim/plugin directory. Maybe you need comment line 42 because it will add cscope.out file twice, it will cause error. This plugin will help you to extract word under cursor. You can use Ctrl+\then press s,g,c,d,t.. Use it in the normal mode.

's'	symbol: find all references to the token under cursor
'g'	global: find global definition(s) of the token under cursor
'c'	calls: find all calls to the function name under cursor
't'	text: find all instances of the text under cursor
'e'	egrep: egrep search for the word under cursor
'f'	file: open the filename under cursor
'i'	includes: find files that include the filename under cursor
'd'	called: find functions that function under cursor calls

- ctrl+\d and c will show all calling functions(c) and called function(d). It's command in cscope.
- ctrl+\s and ctrl+\t are different, result of t will be greater than s, It just think it as text, even it's in the comment. But s will only search source code. It also include context information, such as function context. **You need to use s command more.**
- Add `set cscopequickfix=s-,c-,d-,i-,t-,e-` to cscope_map.vim, It will show multi result in quickfix window. Then you can use :cw window to open it, and :cn and :cp to navigate it, after that, use "ccl" to close quickfix windows. Default window will close automatically when you press enter, I don't like it.

code navigation conclusion

- conclusion 1: without semantic jump

command	usage	note
gD	Jump to the first position	No semantic
[i, [I	Show first	1) no jump 2)lower case just show one, upper case show all 3) no smenatic
/, ?	search	no semantic
*,#	search	Just like / and ?, don't need type
:lv var %	Show all list	1)No semantic
\vv	show all in quickfix	1)No semantic 2)easy-grep
<C-o>, <C-i>	jumt back	

- If you don't want to jum [i or [I are good choice, they just show without jump
- If you want to show and jump in one files, use :lv var %.
- If you want to find var in all files, use \vv and turn on the recursive options.
- these command which don't support semantic also support python langauge.
- conclusion 2: semantic jump

Ctrl+]	Jump to global def	1) semantic 2) need tags
<C-\> g	jumt to global def	1) semantic 2)cscope
\gg, \gl, \gf	Jump to def	1)YCM, 2) can jump local 3)Need correct ycm_con.py

- 1)<C-\>(Cscope) 2)<C-]>(ctags) 3)\gg(YCM) These three command support semantic jump. **I should use these three commands more when I write C++ code.** Other just search or jump without C/C++ semnatic support

1.5.5 Other Edit

NerdCommenter

- Main commands lists:

[count]<leader>cc	Comment out
[count]<leader>cu	UnComment out
[count]<leader>c<space>	Toggle Comment
[count]<leader>cn	Force nest comment

- You can select or use number before these command to give a scope.

Visual-Mark

- Ctrl+F2 only work in gvim, In vim use mm.
- In visualmark.vim, there is space after keymap mm, So everytime when you press mm, It will fold document. you need to go to visualmark.vim and delete Space character after mm keymap.
- How did I find this? you can use :nmap to found all the current command list. **This is very useful when you found there is something wrong with one command.**

1.5.6 quickFix

- When you use cmake, you can run `:set makeprg`make -C /you/own/build=, then when you run :make, it will output error to quickfix windows.
- Default makeprg is "make", but you can use change it to:

```
:set makeprg=gcc\ -Wall\ -ohello\ hello.c
```

  Add \ before space. In this way, you don't need Makefile in this directory. But I recommend you should include a new make file your project directory.
- Use :make command, quickfix window will not show up. If you want to open quickfix window, you should use :cw.
- First use :cw open quickfix window. Then in this windows, you can use :cn and :cp jump. The cursor will jump in code window too.

- **You don't need install any Plugin. But For any C++ project, You should build a make file.** Detail can be found in the next chapter. You can save a make template for simple C++ project.
- command ":cl" will not use very often. You can only see, when you press enter, it will exit this windows.

cc	show error detail
cp	pre error
cn	next error
cl	show all error
cw	If there is error, open quick fix

1.5.7 pyclewn

- vimgdb is old, Don't use it anymore.

```
./configure --enable-shared --with-ensurepip=yes
pip install --install pyclewn
```

- If you system has pip, you can use pip intall, if you don't have root right, with –user, detail can be found google "pyclewn install".It's not installed by vundle.
- First use :Pyclewn gdb to invoke pyclewn. Then use :Cfile myprog. Then use :Cbreak 10, :Crun, :Cnext, use :C follow gdb command. Then :Cquit quit gdb, and Cexitclewn to exit Pyclewn plugin. That is basic usage.
- Next step, you need to know all the basic command in gdb and pdb.
- Open a new terminal, run command tty to know terminal name. Such as /dev/pts/11
- In you gdb, use `:Ctty /dev/pts/11` command to redirect stdout and stderr to new terminal
- For pdb, you can redirect output to another terminal windows.

```
:let g:pyclewn_args="--tty=/dev/pts/4"
:Pyclewn pdb %:p
```

- In your new terminal, use sleep 9999 or sleep(9999). Then you can use new terminal as input.

- RUN:

gdb	pyclewn	:Cmapkeys	custom mappings
next	:Cnext	<C-n>	n,N
step	:Cstep	S	s, S
finish	:Cfinish	F	f
until	:Cuntil	void	u

- Start:

gdb	pyclewn	:Cmapkeys	custom mappings
run	:Crun	R	r
continue	:Ccontinue	C	c

- Breakpoints:

gdb	pyclewn	:Cmapkeys	custom mappings
break	:Cbreak	<C-b>	b
clear	:Cclear	<C-k>	e
info break	:Cinfo break	B	B

- position:

gdb	pyclewn	:Cmapkeys	custom mappings
where	:C where	W	w
frame n	:C frame n	void	void
up down	:Cup :Cdown	<C-u> <C-d>	F6, F5

- varaible:

gdb	pyclewn	:Cmapkeys	cutom mappings
print	:Cprint	void	p(variable in cursor)
set var	:Cset var	void	void
info locals	:C inf..	void	L
info args	:C inf..	void	A
dbgvar expr	:C dbgvar	void	void

- F11 toggle pyclewn, and F12 toggle custom key mappings. All the keymappings are in the .vim/macros/gdb_mappings.vim. I have added it my git repository.

1.5.8 AutoComplete

YoucompleteMe

- **In you .cpp file, add appropriate .h file, and save it. Then YCM can popup corresponding prompts.**

- In 16 color mode, YoucompleteMe popup windows has bad color scheme. you can add two statments in .vimrc to change it. But by now, new version YCM seems to have better color. If you think default color is ok, You don't need to use below statements.

  ```
  highlight Pmenu ctermfg=Bule ctermbg=White guifg=#000000 guibg=#66cc66
  highlight PmenuSel ctermfg=White ctermbg=Blue guifg=#ffffff guibg=#5cadff
  ```

- YCM support python, For support python, you need to install Jedi, You can use pip install –user Jedi. It will install a package in .local/lib/python2.7/site-package.(It's not execuate binary)

- By now, YoucompleteMe support C/C++, python. For C/C++, you need install clang.

- Ycm has identify autocomplete, and semantic autocomplete.

  ```
  let g:ycm_auto_trigger = 1
  let g:ycm_min_num_of_chars_for_completion = 99                "
  let g:ycm_key_invoke_completion = '<C-a>'
  nnoremap <leader>jd :YcmCompleter GoToDefinitionElseDeclaration<CR>
  ```

  It will turn on two autocomplete, min_num option will turn off identify, but you can use <C-a> to trigger identify autocomplete manually. Semantic autocomplete will triggered by ., -> , :: and some directory path symbol such as /.

- Use ctrl+n and ctrl+p will navigate options, Once select, input at the same time, Esc means that you will accept it. If you don't want to select, navigate to select Nothing.

- "Ycm_show_diagnostics_ui = 1" will disable Syntastic check for C/C++, So :Errors will not work for C/C++. But when you save, a error tag will appear in vim gutter, when you move to this line, a error desrcip-tion will appear in the command line window.

- A better way is let g:ycm_always_populate_location_list = 1. In this way, you can use :lne to jump to next syntactic error. You don't need to run make command again and again.

- When you select a option, a preview window will popup in the upper part. You can use :pc to close preview windows.

- If you install YoucompleteMe, You can install Syntastic, YCM will use Syntastic to show some error message.
- You need to compile YCM to complete ycm_core.so file. Git has install explanation. If you system is too old, you should upgrade your system first. I recommend that you use new version Ubuntu or xubuntu, Then download llvm pre-build binary from llvm.org to get libclang.so. **Don't try to compile YCM on old system. It has a lot of random problem if you config your own G++ compiler**
- After you compile, you need .ycm_extra_conf.py file.
- You need to add some include path directory into .ycm_extra_conf.py file.`g++ -E -x c++ - -v < /dev/null`. It will show all the include path. Then you can modify .ycm_extra_conf.py file. But a better method is to use YCMGenerater, It will produce your own project .ycm_extra_conf.py
- For YCM-generator, It's easy to use, jut go to .vim/bundle/YCM-Generator, then run config_gen.py pro_dir. The pro_dir should includes Makefile file in it. It will run make clean first, then dry-run make to collect all the compiler flag. It remind me the understand buildspy.By now, it also support CMakeLists.txt file.
- If YCM doesn't work, add two lines to .vimrc

  ```
  let g:ycm_server_keep_logfiles = 1
  let g:ycm_server_log_level = 'debug'
  ```

  First run :YcmRestartServer.Then run :YcmToggleLogs <tab> select stderr log file. All the log files are saved in /tmp directory.
- run below commands to see If there are something wrong with YCM.

  ```
  ~/.vim/bundle/YouCompleteMe/run_test.py
  ~/.vim/bundle/YouCompleteMe/third_party/ycmd/run_test.py
  ```

Ulitsnippet and Vim-Snippet

- g:UltiSnipsListSnippets=<C-a> and <C-tab> doesn't work on some terminal emulator. Anytime you want to use a snips, you can press <C-a> to take a look. It will show all content in more pager,(which doesn't support search). You can press G<cr> to quit this more pager.

- <C-a> command can recognize you file type automatcially. When you press <C-a> in tex, It only show all tex snippet.
- By now, I mainly work on tex, C/C++, python, and html. So I need to remember some common used snippets in these four file types.
- Ulitsnips is engine, Vim-Snippet include a lot of template which is written according to Ulit engine syntax.
- Basically, you can go to Ulit directory to see your favorite language snippets.
- For example, input cl, then press tab, It will insert the whole class.
- Then, You can use <C-j> and <C-k> jump back and forward editable points.
- In this direcotry: .vim/bundle/vim-snippets/snippets, you can see a lot of template for different language. Such as C, C++, Python etc. You can open them to see what snippet it support. For example, all the STL contianer, vector, or map.
- Once you are in the end of edit point, You can't jump back again. If there is three edit points, You can jump from two to one, but once you get to three, you can't jump back two and one.
- In the future, If you now Ulit syntax, then you can make your own snippets.
- **Some common use pattern are: main, fun, for inc ndef def, if ife el elif.**

incc def #if	include
main	main
fun	fun
td st	typedef struct
vector set map	âĂć
cl	class
cout pr	printf
for fori fore	
iter itera	
dfun0	

SuperTab

- SuperTab is just easy invoke method of vim's ins-completion. Detail can be seen :h ins-completion.
- Default is ctrl+p, and When you change another insertion method, tab will remember it until you exit insert mode. Of cource, you can customize it, detail can be seen SuperTab website.
- Use tab to switch all the options.
- It doesn't conflict with vim-snippets.
- Ctrl+p will remember all the name in this buffer, so you can use long varaible name in your c/c++/python code.then use tab to pop them up.
- If you just remember first few letters, then press them, then press tab, then a window will popup with all options.
- If you don't remember the first letter, You can use vim default auto-complete Ctrl+p, and Ctrl+n.

    1. **Just use them in insert mode, not in normal mode: In normal mode, Ctrl-n will invoke Nerdtree plugin, and Ctrl-P will invoke CtrlP plugin.**
    2. Just use them in Source code, when you forget some previous varaible name.
    3. Ctrl-P will list all the previous word from bottom, So you should use it when you are programming.
    4. Once you are in popup window, Just use Ctrl-n and Ctrl-P to move your cursor.
    5. When you use Ctrl-P, a word has been inputted into vim, Then you can press Ctrl-n, It means you don't select any word in the popup window, then word will disappear. Then input what you want to input a few beginning characters, Then delete one, popup window will be refresh to filter according to your input. That's a little trick.

Auto-complete Conclusion

- Basically, There are three kinds of methods for auto-completion:
  1. YCM for programming.
  2. Ulitsnippet for programming.
  3. ins-completion in VIM.
- For text format file. You can use ins-completion. By the way, you can use Supertab plugin to avoid type complex keystrok.
- For Programming file. Mainly you should use YCM and Ultisnippet. You also can use ins-completion(insert word in comment and insert previous line in code). Supertab is also useful because I map Ultisnippet to <C-j>.
- **For text file, use ins-completion and tab as syntax sugar. In tex or html file, you also can use tab to trigger snippet,such as "enum, item, center, and tab"**
- In vim, **Once you select, it will insert it to document, You need to get used to this style. Accept is just close the popup windows.**
- YCM usage:
  1. **1) trigger 2) select 3) accept**.
  2. use <c-l> to trigger, or use . or file path / to trigger.
  3. In popup window, you can <C-e> to quit.
  4. **If it's easy to select what your want.** default it's in filter status, you can use 1) tab s-tab 2) <C-n><C-p> 3)up, dow to select.
  5. **If it's not easy to select what you want**. You also can keep typing, it will help you filter the pop up windows.
  6. There are three way to accept 1) keep input another character 2) enter 3) <C-y> **prefer to use method 1, method 2 is usefule for python because it will change to next line 3)last use <C-y>, for example you want to input tab next, so you have to use <C-y> first, it's slowly**
- Ins-completion usage:

1. Just like YCM, it also include **1) trigger 2) select 3) accept**.
2. use <C-x><C-.> to trigger, use <tab> to save keystroke when you want to repeat next time.
3. <C-e> quie.
4. Different with YCM, **insc-completion is in selected status.** Easy select , just keep use three ways to select what you want.
5. If not easy to select, <C-n> or <C-p> to unselect anythin, then keep typing to filter popup window.
6. There are also tree way to accept. just like YCM.
7. **When YCM working in C or python file, ins-completio stop working in filter mode, I found a solution, in .vim/bundle/YoucompletMe/autoload/youcompleteme.vim file. modify two functions. comment two statments as below:**

```
function! s:OnInsertChar()
  call timer_stop( s:pollers.completion.id )
  if pumvisible()
    "call feedkeys( "\<C-e>", 'n' )
  endif
endfunction

function! s:OnDeleteChar( key )
  call timer_stop( s:pollers.completion.id )
  if pumvisible()
    "return "\<C-y>" . a:key
  endif
  return a:key
endfunction
```

- For C++ and Python file

content	usage
word	<C+x><C+k> and <C+x>s(set spell)
file	1)YCM ./ and / trigger 2)<C+x><C+f> insert cwd file
. -> #	YCM
Previous keyword	1) <C+x><C+p> 2) YCM <C+l>
code snippet	<C+j> snippet
previous line	<C+x><C+l>
keyword and definition	<C+x><C+i> and <C+x><C+d>

1. Just use word in comment, don't :set spell, it will cause a lot of spelling error.
2. tab is used in supertab, and it's just shortcut of vim's ins-completion. So you need to understand <C+x><C+.>
3. YCM is different with <C-x><C-p>. YCM allow you modify your input, and popup window will change according to your input. <C-x><C-p> when you input or delete a character, pop-up window will disappear.
4. <C-x><C-p> is different with YCM. YCM has better understand of your code. <C-x><C-p> is just repeat any word previous cursor.
5. For C/C++ code 1)file 2)<C-l> 3)<C-x><C-l> 4) <C+j>. They are more useful.
6. <C-x><C-i> and <C-x><C-d> will give you a lot of prompts, maybe it's not very useful.
7. Type the first one or two letters will give you more accurate prompts.
8. By now, 1)YCM 2)supertab 3)snippet 4)ins-completion. They are all avaibile in our VIM.

- Other document

content	usage
word hline spell	<C+x><C+k> <C+x>s(:set spell)
file	<C+x><C+f> insert cwd file
Previous keyword	<C+x><C+p>
previous line	<C+x><C+l>
vimtex and emmet	plugin or <C-j> to insert snippet

Chapter 2

Developing tool

2.1 gcc

2.1.1 gcc basic

- `$ gcc -c hello.c ↵` will only produce object file hello.o. Then you can use `$ gcc hello.o -o hello ↵` to produce executable file. You can use two steps just compile modified c file. It will save you a lot of compile time. `$ gcc -c *.c ↵`, then `$ gcc *.o -o last ↵`. You can omit the -c option to compile and link in one step. A better method is to use make file.

- `$ gcc -Wall hello.c -o hello ↵`–Wall is an important options, you should always use it.

- But sometimes, these warnning are normal, you can use below to disable for specific statements.

```
#pragma GCC diagnostic error "-Wuninitialized"
    foo(a);         /* error is given for this one */
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wuninitialized"
    foo(b);         /* no diagnostic for this one */
#pragma GCC diagnostic pop
    foo(c);         /* error is given for this one */
#pragma GCC diagnostic pop
    foo(d);         /* depends on command line options */
```

- default is –O0, if you want to use gdb to debug your applicaiton, you should use –g.

- In general, `$ gcc -Wall hello.c -lm -o hello ↵` the compile options –lNAME will attempt to link object files with a library file libNAME.a or libNAME.so in the standard library directories.
- gcc -c will invoke ccl and as. gcc -o will invoke ccl, as and ld
- In a new linux system, sometimes you need to install build-essential package, it contains C/C++ language include file and library file.
- -DNAME defines a preprocessor macro NAME

```
#ifdef NAME
printf ...
 #endif
```

- `$ gcc/g++ -E -dM file.c ↵` will output only result from preprocessor and all defined macros.
- use `$ ldd ↵` can see what libs does an application depends.
- `$ nm ↵` can tell you what functions there are in a lib: there are T,U, and W categories. `$ nm -D -defined-only libname.so ↵`
- `$ ar -t ↵` can see what .o files are included in a lib. will see contents in a .so file.
- for .so use file command, for .a use objdump -x to see if they are 32 or 64 bits.
- Use `$ nm -D -defined-only libname.so ↵` to get the symbol names from your dynamic library. The –defined-only switch shows you only the symbol that are defined in these files, and not references to external functions. An alternative is to use objdump, and catch only the symbols in the text section :
- About optimization, a good book is "An Introduction to GCC-Brian_Gough", You can google it.I have download it into my ref directory. For basic knowledge, google "GCC and Make Compiling, Linking and Building C/C++ Applications".

2.1.2 include

- Defaultly, gcc searches the following /usr/local/include/ and /user/include for header file.

- You can use -I to add your include path, but It will only affect current gcc/g++ command.
- You also can use `$ C(CPLUS)_INCLUDE_PATH=... g++... ↵`. You should put them on the same line. CPLUS environment will only valid in this line. -I method is better because it save typing.
- `-I\users\yzhao\myinclude` There is no space between -I and path variable.
- Double quote header will be searched from current directory.
- Any search will NOT recursive automatically, So you must use "path/-foo.h" to specify the exact position.
- For search order. quote>system. -I will be the first in system. -I will follow from left to right.
- `g++ -E -x c++ - -v < /dev/null`. It will show all the include paths and order.
- cpp -v hello.c will also show all the include paths.
- Based on above knowledge. You can deal with most of tasks right now. If you need harder requirement. you can google "Options for Directory Search gcc". There is a gcc document to introduce this topic.

2.1.3 linker

- By default, gcc searches /usr/local/lib/ and /usr/lib/ for lib files.
- libchild.a is based on libbase.a, then you have to put -lchild in front of -lbase, or it will produce dependent problem.
- `ld -o output /lib/crt0.o hello.o -lc` You can input .o file directly to ld command. -lc means libc.a or libc.so. It will produce "output".
- Unless you have some very specific platform integration requirements, or have reasons not to use gcc(g++), I can hardly think of any advantage of invoking ld directly for linking. Any extra linker-specific option you may require could easily be specified with the -Wl, prefix on the gcc command line (if not already available as a plain gcc option).

- If the linker is being invoked indirectly, via a compiler driver (e.g. 'gcc') then all the linker command line options should be prefixed by '-Wl,' . An example can be found below when you use "shared object name". Such as libhello.so.0.0.1
- You can use -L to add you library PATH, and -l library name. In you source code, you only give function name, so you have to use -l to specify the libary name.
- ld –verbose | grep 'SEARCH*" will show you all the default ld search library paths.
- LIBRARY_PATH. you can write it in you .cshrc or .bashrc to affect all your terminals. Or put it in front of gcc/g++ command, which must stay in the same line.
- The linker will search your -L directories in the left-to-right order in which they appear in the command line and it will search all your -L directories before the default linkage directories.
- Then It will search LIBRARY_PATH. Last it will search default search library path./lib /usr/lib and /usr/local/lib. (I am not quite sure about this conclusion, I get it from web. I didn't confirm it in my experiment.)
- `g++ -E -x c++ - -v < /dev/null`. It will show all the include path and LIBRARY_PATH and COLLECT_GCC options.
- gcc -v give you link library path order and detail information.

2.1.4 compile .so

- Compile .so without soname.(The soname is often used to provide version backwards-compatibility information)

  1. Use below command to compile .so file in linux.
     `gcc -shared -o libpong.so -fPIC pong.c`
     This command will prodouce a libpong.so file. Then you need to create pont.h file include all the declartion.
  2. Then you can make a client programme. It need to inlcude pong.h file then use below gcc command to compile.
     `gcc -o test test.c -lpong -L.` You need to use -lpong to specify libname. -L is to specify library search path, dot represent current directory.

- compile .so with soname
  1. `gcc hello.c -fPIC -shared -Wl,-soname,libhello.so.0 -o libhello.so.0.0.1`
  2. When you have libhello.so.0.0.1. run `$ ldconfig -n .` ↵, It will produce a soft link libhello.so.0
  3. When you compile client program `gcc main.c -L. -lhello -o main`. You need to build soft link manually ln -s libhello.so.0.0.1 libhello.so. Pay attention here. 1) no number after .so 2) use ln -s command. not use ldconfig -n . After that, When you run `$ ldd main` ↵, you will see main is only depended to libhello.so.0
  4. after that, you can produce libhello.so.0.0.2...9. Then run ldconfig-n. libhello.so.0 will point to the newer version .so. You don't need recompile main at all.
  5. nm -g libhello.so will list all symbol in a .so file
  6. .a is static library, and .so is dynamic library. Sometimes, a lib will provide lib.a and lib.so at the same time. gcc will use the lib.so first. You can use –static to tell gcc to use lib.a version.

2.1.5 load .so

- `$ ldd` ↵ will tell you what lib you are using in your executable programme.

- for elf format applicaiton. It will search elf DT_RPATH >LD_LIBRARY_PATH >/etc/ld.so.cache > /lib and /usr/lib

- `$ readelf -d libfftw3_mpi.so` ↵ | grep RPATH and see if it has /usr/lib64/ as a library path. If it exist, chrpath -r<new_path> <executable> to change the rpath in the library

- export LD_DEBUG=libs you can debug the search path used to find your libs.

- For system effect, 1) add .so to /lib or /usr/lib. 2) edit /etc/ld.so.conf, then run ldconfig to produce /etc/ld.so.cache.

- When you add .so to /lib or /usr/lib, You don't need to modify /etc/ld.so.conf. but you need run ldconfig. Or this library will not found

- Add .so to other paths(excludes /lib /usr/lib), You need to modify /etc/ld.so.conf. then run ldconfig.

- in Ubuntu, # echo "/path-to-your-libs/" > /etc/ld.so.conf.d/your.conf After that run sudo ldconfig
- /lib/ld-linux.so.2 is dynamic loader.
- If you don't have write permission, you can use LD_LIBRARY_PATH
- ldconfig is just run time. It has nothing with compiling.
- LD_PRELOAD tell loader: pick up a fun in specified .so first.
- For complex .so problem, such as same name in different .so. You should see below two documents.

  ```
  The LD_DEBUG environment varaible.
  http://www.bnikolic.co.uk/blog/linux-ld-debug.html

  THE INSIDE STORY ON SHARED LIBRARIES AND DYNAMIC LOADING
  ```

- For .so file, You can use gcc implicit link, then use LD_LIBARY_PATH to dynamic load it. You also can dynamic open a .so file and load a function to call. It in your main.c. In this way, you don't need set LD_LIBARY and more

```c
#include <stdio.h>
#include <dlfcn.h>
int main(int argc, char *argv[]){
        void *dl = NULL;
        int (*add)(int a, int b);
        dl = dlopen( "./libtest.so", RTLD_LAZY);
        if( dl == NULL ){
                printf("so loading error.\n");
                return 1;
        }
    add = (int(*)(int, int))dlsym(dl, "add");
        if( dlerror() != NULL ){
        printf("fun load error.\n");
                return 1;
        }
        printf("%d\n", add(1, 2));
        return 0;
}
```

2.2 gdb

2.2.1 start

- `$ gcc`
  `g++ -g file.c`
  `file.cpp ↵` you need –g to compile souce code before gdb
- `$ gdb -tui ↵` start good GUI mode
- `$ gdb app ↵` just load symbol information, then you can use run arg1 arg2.. to run this problem.
- `$ help ↵` will list classes of commands then type help followed by class name. or help followed by command name.
- you can use another terminal to compile this file and then in you gdb to kill and run again. all the breakpoints will be keep.
- you can kill and run app again at any time.

2.2.2 break

- `$ break 19 ↵` or `$ break test.c:19 ↵` or `$ break function1 ↵`, for C++, you need to tell break function list of argument types. such as `$ TestClass::testFunc(int) ↵`
- `$ info breakpoints ↵` will list all the breakpoints
- After you have list all the breakpoints, you will know the number of them, then you can use `$ disable 2 ↵` to disable the second breakpoint. ignore 2 5 will skip the number 2 and number 5 breakpoints.
- `$ tbreak  ↵` will just stop once, then it will be removed.

2.2.3 Contrl Running

- When your program is running, send ctrl+C to stop it, and you can type continue command to restart execution.
- `$ until line or function ↵`
- list command show you current context information.

- `$ step ↵` will go into the function and `$ next ↵` will go over the function.
- `$ print ↵` will output the variable value, and `$ set ↵` will set variable value.
- `$ call function ↵` and `$ finish ↵` will finish current function.
- look at the contents of the current frame, you can use `$ info frame ↵` and `$ info locals ↵` and `$ info args ↵`

2.2.4 stack

- `$ backtrace ↵` will show you the the whole stack frame, on each level, there are numbers on left.
- `$ frame 2 ↵` will just show that level information.
- `$ gdb bt ↵` will tell you which file, which function and which line you are current in.

2.2.5 advanced

- `$ info registers ↵` will see all the cpu registers information.
- disassemble main to see assembly code.
- for x command , you can use 4xw or 4wx, they are both ok. size modifiers include(b,h,w,g). Format include(o,x,d,u,f) and (t,a) and (c, s) and i.
-

2.3 Automaticly Build

2.3.1 make

- Makefile uses compiler and shell programming tools(such as rm, cp etc) together! `$ make ↵` command will look for makefile automatically first. So you should write you own Makefile.
- You also can use make -f to specify you own makefile name, A Makefile can be regarded as a script file.

- The basic part of Makefile is:

  ```
  Target: prerequisites
  tab comm.and
  ```

- To check which one has changed, if someone has change, it will call command. That is the most important, you must remember it all the time. And it is not difficult, isn't it?
- Comment is #, just like comment statement in script file.
- Define variable in shell script: A="var_name" (no space with quote). Define variable in makefile A = var_name (with space no quote)
- Use variable in shell script $A, use variable in makefile $(A). You have added a parenthesis around variable name.
- Make -p to print the all default MACRO, $@ is the names of the file to be made, and $? Is the names of the changed dependents.
- PWD :=$(shell pwd) I need to explain two thing, the first is difference between := and =, := only expand this macro once. PWD is macro. After you define this macro, you can use it later in you file with $(PWD). Seconde $(shell command) will call shell command and return back result to this variable.
- make internal variable list:
- @echo can be used to output string. It also can output the variable
- use @ to call shell command without output command itself. Use âĂŞ to tell make to ignore any error. Even b.txt is not exist, if you put - before rm, it will continue to run all: If you don't put - before rm. make will stop at rm b.txt command. And rm a.txt will not be called at all.

  ```
  all: all_1
  rm a.txt

  all_1:
  @echo "no go to here"
  -rm b.txt
  ```

- Make all that is ok, don't add any other element. Use default to run when you don't give make and argument
- A := $(wildcard *.a) ALL_B :=$(wildcard *.b) A_B :=$(A:%.a=%.b)
- First, when you deal with a list of files, you use := ; second when you need a and b you need use A_B to express this set.
- n order to figure out the default paths used by gcc or g++ as well as their priorities you examine the output of the following commands:

```
For C:    gcc -xc -E -v -
For C++: gcc -xc++ -E -v -
```

- A simple make example for C/C++ project. There are two points here: **1) Use $(CXX) and $(CC), and never hard-code a value for $(CXX) or $(CC). Let make define them for you!**

  2) It is bad manners to overwrite CFLAGS, CPPFLAGS, CXXFLAGS, or LDFLAGS using := or =, but it is ok to augment them with +=

```
#
#  "program_NAME" with a value of "myprogram".
# a lowercase prefix (in this case "program") and
# an uppercased suffix (in this case "NAME"), separated
# by an underscore is used to name attributes
# change it with your own name here
program_NAME := myprogram


# all files in the current directory ending in ".c".
# The $(wildcard) is a globbing expression.
program_C_SRCS := $(wildcard *.c)
program_CXX_SRCS := $(wildcard *.cpp)

# This names all C object files that we are going to build.
# substitution expression, simply replaces ".c" with ".o"
program_C_OBJS := ${program_C_SRCS:.c=.o}
program_CXX_OBJS := ${program_CXX_SRCS:.cpp=.o}

# This is simply a list of all the ".o" files,
#both from C and C++ source files.
```

```
program_OBJS := $(program_C_OBJS) $(program_CXX_OBJS)

# This is a place holder.  For example:
# program_INCLUDE_DIRS := ./include,
program_INCLUDE_DIRS :=

# This is a place holder. For example:
# used program_LIBRARY_DIRS := ./lib,
program_LIBRARY_DIRS :=

# This is a place holder. For example:
# program_LIBRARIES := boost_signals,
program_LIBRARIES :=


# -I$(includedir) expand to -I./include, then add to CPPFLAGS
# Remember that CPPFLAGS is the C preprocessor flags,
# compiles a C or C++ source file into an object will use this flag.
CPPFLAGS += $(foreach includedir,$(program_INCLUDE_DIRS),-I$(includedir))

# Since the LDFLAGS are used when linking,
# this will cause the appropriate flags to be passed to the linker.
LDFLAGS += $(foreach librarydir,$(program_LIBRARY_DIRS),-L$(librarydir))
LDFLAGS += $(foreach library,$(program_LIBRARIES),-l$(library))

# This indicates that "all", "clean", and "distclean" are "phony targets".
# Therefore, "make all", "make clean", and "make distclean"
# should execute the content of their build rules,
#even if a newer file named "all", "clean", or "distclean" exists.
.PHONY: all clean distclean


tags:
        echo "$(program_CXX_SRCS)" > cscope.files
        rm cscope.out cscope.in.out cscope.po.out
    cscope -Rbq
        rm tags
        ctags $(program_C_SRCS) $(program_CXX_SRCS)

# This is first build rule in the makefile,
# "make" and executing "make all" are the same.
# The target simply depends on $(program_NAME),
# which expands to "myprogram", and that target is given below:
```

```
all: $(program_NAME)


# The program depends on the object files
#(which are automatically built using the predefined build rules...
#nothing needs to be given explicitly for them).

# The build rule $(LINK.cc) is used to link the object files
# and output a file with the same name as the program.
# Note that LINK.cc makes use of CXX, CXXFLAGS, LDFLAGS, etc.
# On my own system LINK.cc is defined as:
# $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH),
# so if CXXFLAGS, CPPFLAGS, LDFLAGS, and TARGET_ARCH are undefined,
# but CXX is g++, expand to g++ $(program_OBJS) -o $(program_NAME).

$(program_NAME): $(program_OBJS)
	$(LINK.cc) $(program_OBJS) -o $(program_NAME)

#
# This target removes the built program and the generated object fil
# The @ symbol indicates that the line should be run silently,
# and the - symbol indicates that errors should be ignored
# (i.e., if the file already doesn't exist, we don't really care,
# and we should continue executing subsequent commands)

clean:
	@- $(RM) $(program_NAME)
	@- $(RM) $(program_OBJS)



#
# The distclean target depends on the clean
#target (so executing distclean will cause clean to be executed),
# but we don't add anything else.

distclean: clean
```

2.3.2 autotools

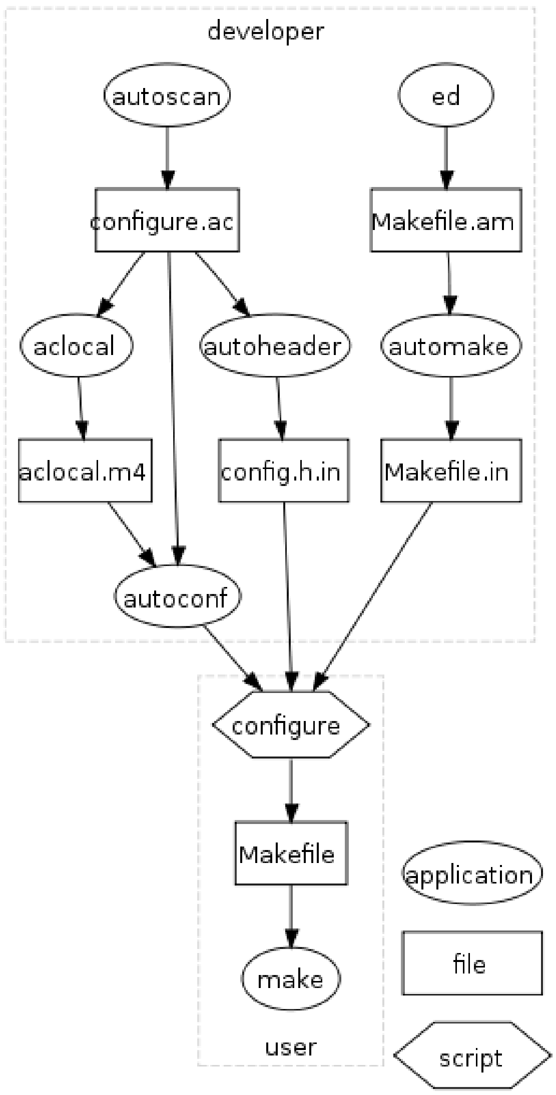
- autotools includes some separate tools, such as autoreconf, autoconf, automake, autoheader and acolcal.m4
- **Three important points:**

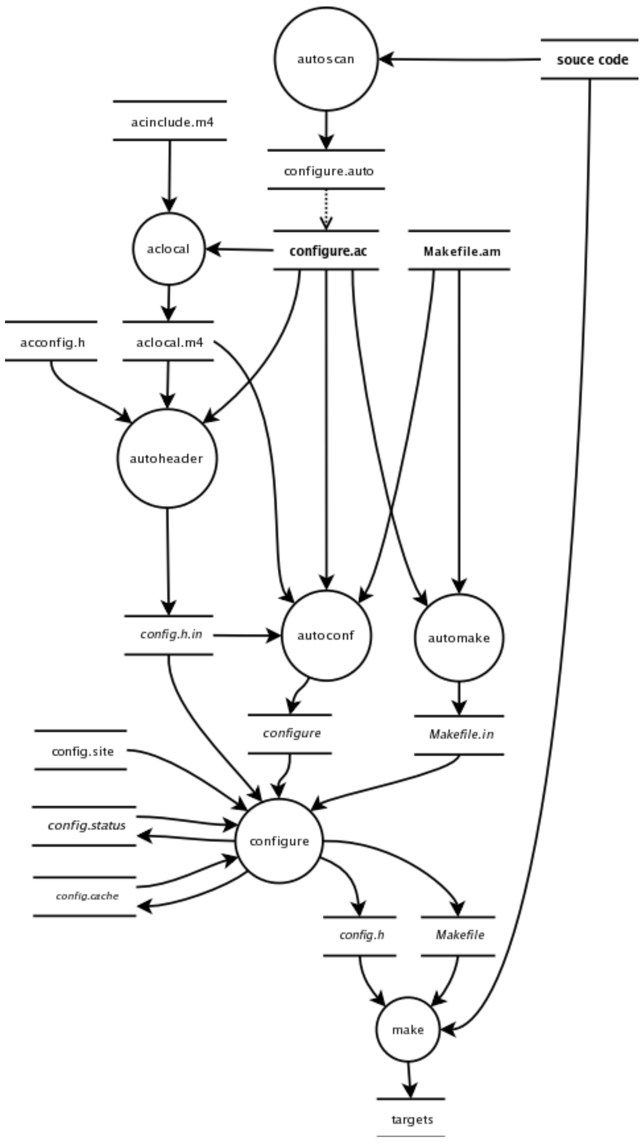
1. edit makefile.am to write your file content structure

2. use autoscan to produce configure.scan, rename configure.ac, then modifiy to make you source code more portable.

3. run autoreconf to run all the tools behind the scene. You also can run the command step by step.

- Two simple fig.s can be seen here.





- Basic steps:
  1. create project
  2. autoscan, then rename configure.scan ==> configure.ac
  3. aclocal
  4. autoheader(optional, it will produce config.h.in. But if you use AC_CONFIG_HEADER, you must run it.)
  5. Makefile.am(If you have multi-deep directory, make sure each directory has one)

6. libtoolize âĂŞautomake âĂŞcopy âĂŞforce(if configure.ac use libtool)
7. automake âĂŞadd-missing
8. autoconf, it will produce configure command
9. build another build_dir, parallel with source project
10. cd build_dir. Then ../source/configure –prefix=/home/yan/install
11. make && make install

- I have add some good reference paper to ref directory. When you really need them, just read them first.
- By now, it support VPATHS, you can download source code, then make build1 subdir inside, then cd build1 sub, run ../configure, it will put all obj,bin and lib file in build1 sub directory. if you want to build another version. make build2 subdir inside, then run ../configur –different-conf. You can use the same set source code in this way.

config.ac

- A example configure.ac

```
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ([2.69])
#must have
AC_INIT([Hello], [0.3], [zhaoyan.hrb@gmail.com])
#change it
AC_CONFIG_SRCDIR([config.h.in])
# Test if configure.ac is in the right directory

AC_CONFIG_HEADERS([config.h])
#use config.h to customize you source code

AC_CONFIG_MACRO_DIR([build-aux/m4])
# move your local m4 macro to target machine
# to avoid portable problem

AM_INIT_AUTOMAKE([foreign])
#use foreign, don't need README... five files.
```

```
case $host in
    cpu-*-os*)
        COMP="aaa bbb"

        ;;
    *)
        COMP="ccc"

        ;;
esac
# ../src/configure --host=cpu-yan-os
AC_SUBST(COMP)
# COMP will be aaa bbb, then use AC_SUBST to replace
# COMP in the Makefile.am , "SUBDIRS = $(COMP)"
# in this way, aaa bbb will be compiled, otherwise,
# ccc subdirectory will be built


# Checks for programs.
AC_PROG_CXX
AC_PROG_AWK
AC_PROG_CC
AC_PROG_CPP
AC_PROG_INSTALL
AC_PROG_LN_S
AC_PROG_MAKE_SET
## AC_PROG_RANLIB

AC_ARG_ENABLE([ember_paths],
[  --enable-ember-paths    cross-compile for EmbeR.],
[case "${enableval}" in
  yes) ember_paths=true ;;
  no)  ember_paths=false ;;
  *) AC_MSG_ERROR([bad value ${enableval} for --enable-ember-paths]) ;;
  esac],[ember_paths=false])
AM_CONDITIONAL([EMBER_PATHS], [test x$ember_paths = xtrue])

LT_INIT
# use libtools, detail can seen in libtools section

CXXFLAGS='-std=gnu++1y -g -O0'
```

```
# You can set users variable for make here.

# Checks for libraries.
AC_CHECK_LIB([lib1], [fun1])

# Checks for header files.
AC_CHECK_HEADERS([fcntl.h stdlib.h])

# Checks for functions.
AC_FUNC_MALLOC
AC_CHECK_FUNCS([gethrtime gettimeofday])

# Checks for typedefs, structures, and compiler characteristics.
AC_CHECK_HEADER_STDBOOL
AC_C_INLINE
AC_TYPE_INT64_T
AC_TYPE_MODE_T
AC_TYPE_SIZE_T
AC_CHECK_TYPES([ptrdiff_t])

AC_CONFIG_FILES([Makefile
                 src/Makefile
                 src/aaa/Makefile
                 src/bbb/Makefile
                 src/ccc/Makefile
                 ])
#which make file you want to produce.

AC_OUTPUT
```

- Main function of congiure.ac is to check lib, function and header file in the target system, then give error or config.h. In the end, you source code can custimize its behavior accordingly.

- Another function is to set some variable, such as CXXFLAGS.

- configure what subdirectories to be built on the different platform. In another word, what Makefile should be produced.

Makefile.am

- For recursive directory, each directory should have Makefile.am. on parent directory, just set "SUBDIRS = sub1 sub2"
- use libtool to produce .so

```
AM_CXXFLAGS = -Wall -fPIC
if SWITCH
AM_CXXFLAGS += -fprofile-arcs -ftest-coverage
endif
//For ENABLEGCOV, you can use to
../src/configure --enable-switch to turn on it.

lib_LTLIBRARIES = libyan.la

libyan_la_SOURCES = \
src1.cpp \
src2.cpp \

AM_CPPFLAGS = -I$(top_srcdir)/src/includes

libyan_la_LDFLAGS = -version-info 0:0:0 -L/usr/local/lib -L$(ODDS_LIBS)
libyan_la_LIBADD = -llibabc
```

- produce .exe

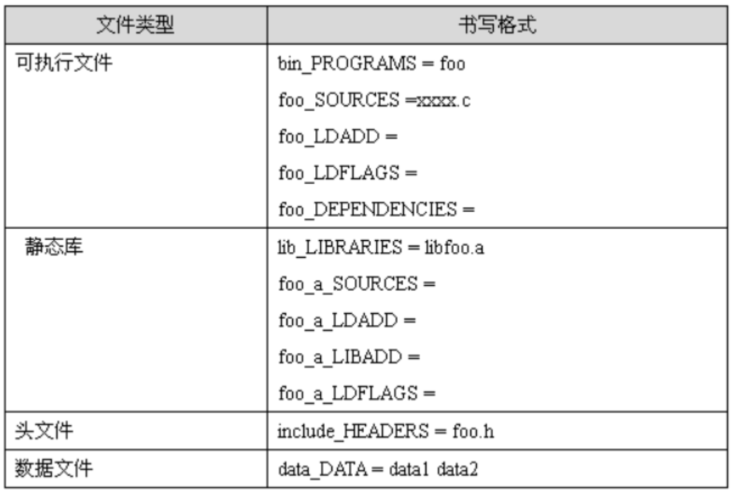
```
include $(top_srcdir)/common.am
bin_PROGRAMS = exe1

exe1_SOURCES = \
main.cpp  \
init.cpp

exe1_CPPFLAGS = -I$(top_srcdir)/src/includes
exe1_LDFLAGS = -L/usr/lib64 -L/usr/local/lib64
exe1_LDADD = -lboost_log -lboost_setup
```

- For bin, lib, include, they are position. PROGRAMS LIBRARIES and HEADERS, they are categories. So for "bin_PROGRAMS", it means that I want to produce an executable file and the executable file will be in bin directory.

- include_HEADERS means that this file will be installed in include directory.
- A simple picture



文件类型	书写格式
可执行文件	bin_PROGRAMS = foo foo_SOURCES =xxxx.c foo_LDADD = foo_LDFLAGS = foo_DEPENDENCIES =
静态库	lib_LIBRARIES = libfoo.a foo_a_SOURCES = foo_a_LDADD = foo_a_LIBADD = foo_a_LDFLAGS =
头文件	include_HEADERS = foo.h
数据文件	data_DATA = data1 data2

libtools

- libtools is mainly used to produce .so file for different OS.
- configure.ac and Makefile.am also support it.
- In makefile.am use lib_LTLIBRARIES, not lib_LIBRARIES(that is used to build static library.)
- How to compile a .so? See automake.am example in the previous section.
- How to use a .so? You can use .la file directly in you automake.am. Pay attention to the two points: compile .so you need to use _LIBADD to specify dependency. when you use .so you need to use_LDADD to specify .so dependency. At the same time. .so will be baked into exe with rpath, so you can run your main directly. Don't need to set LD_LIBRARY_PATH varaible.

```
bin_PROGRAMS = main.c
main_LDADD = /path_to_la/libabc.la
#la is special file extension of libtool
```

- A good introduction file can be found in "use GNU Libtool build lib" by wuxiaohu in Chinese version.

2.3.3 CMAKE

- The best way to study is see a simple and complete template file. You can google "most simple but complete cmake example" and take a look.

- In the github, I have camke-test project, you can download it.

- There is a good tutorial of cmake, "CMake Practics"(Chinese version). I have add it to the ref directory. It introduces almost all the basic usage and explanation. I need to read it before you want to use CMake to manage your own project.

- If you want to use gtest in your project. You can google "Why is it not recommended to install a pre-compiled copy of Google Test (for example, into /usr/local)?" So CMake use "ExternalProject_Add" command. It download gtest source code and use the same Compile flag to compile it.

- A better document about gtest and CMake is "Unit testing with GoogleTest and CMake".

- You can use target_link_libraries(myexecutable mylib) to link to the library "mylib". The compiler will use its default way to find the specified library (e.g. it will look for libmylib.a on Linux). The compiler will only look in the link_directories(directory1 directory2 ...), so you could try that command to add the required directories to the search path.

- When "mylib" is also compiled with CMake this will be recognized and everything should work automatically.

- When you want the user to specify a directory you can use a cached CMake variable. set(MYPATH "NOT-DEFINED" CACHE PATH "docstring").

- Once you run cmake once, you don't need run it again, even you modify other CMakeLists.txt in the subdirectory, you just need run make, it will detect modification automaticlly.

2.4 git

2.4.1 Basic knowledge

- *A basic rule: For single person, always pull (fetch and merge) before you work, commit and push after you work. For multi person, fetch and merge even before each commit. Maybe other people have committed a new version on the remote repository.*

- Run `$ git gc ↵` every month to run it to optimize.

- In Linux, you can `$ sudo apt-get install git-all ↵` and meld(merge tools), In windows, you can download msysGit and P4Merge. You need edit your .gitconfig file, showed in the next section. you can use meld or P4Merge to visualize conflicts in source code. In Mac, you can download git from git home website. or install Xcode, when you install Xcode, It will install git in /usr/bin. but it will not install gitk GUI tool, so you can download git from home website and install another git on /usr/local/bin directory and use gitk in this directory.

- The two characteristics about git is **Distribute** and **Branch**, Distribute support working offline, Branch can make you manage branch easily and efficiently.

- You need to know a few important conceptions: repository, branch, commit, paths and files. You need to know object of a command, such as merge command, it needs two branch. not two commits. Checkout branch will switch to its branch, and checkout commit will cause detached-head, etc.

- **origin** is a repository, **master** is a branch, **HEAD** is a ref to a branch. A repository can have many branches, so you can use origin/master to specify one of them. A branch can have many commits, so you can use HEADˆ to refer to it.

- You also need to know some low-level knowledge about git. such as commit–>tree–>blob, commit. You can use `$ git log ↵` to know the sha value, then use `$ git cat-file -t (or -p) sha ↵` to check them.

- you can use commit –allow-empty to produce a lot commits to use as test. then use `$ rebase -keep-empty ↵` to learn how to change history. Only some practical usage can teach you some knowledge. For Git , Dirty Hand is very very important.

- checkout and merge are different, checkout is used to **overwrite present with history**, merge is used to **combine present with history**.
- When a command can be followed by <paths>, then <paths> can be:
  1. a file
  2. *.ext Only current level directory, not recursive child directory.
  3. . (all file) All files recursively under child directory.
  4. /path_name(path_name and recursive)
  5. /path_name/*(no recursive).
- When you read manual, you need to know, the same command has different usage when followed by different objects. For example, when checkout followed by a branch name, followed by a commit or followed by a <paths>, detail can be seen in git book p103. reset followed by <paths> or followed by a commit is also different, detail can be seen git book p96. The main points is **checkout commit with path will change HEAD ref, and reset without path will change branch ref(such as master).**
- You need to understand difference between tree-ish and commit-ish. commit-ish can be used as tree-ish, but tree-ish can't be used as commit-ish.
- git cat-file and git show-ref are two useful commands, show-ref can give you big pictures of whole git project, and cat-file can help you to see deeply into each sha value. Detail information can be found in git document.

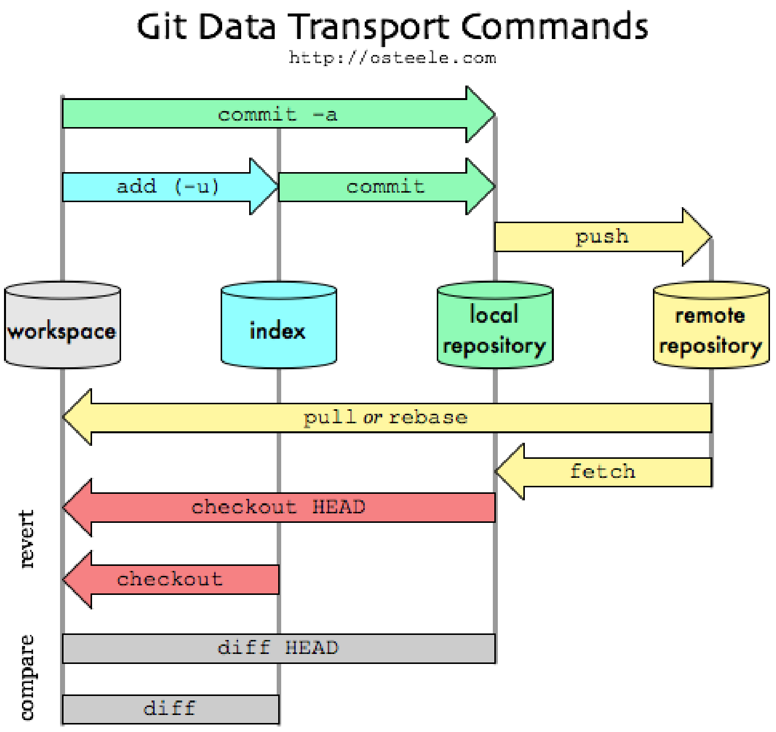
git configure

- `$ git config -e [-global|system]` ↵ -e will open a editor. –global means user, –system means /etc/.gitconfig. If you omit options, It just produce a .gitconfig file for this repository. (local) Most of time, you don't have right to write in –system level.
- It's a social website, you need to find some friends here and exchange idea. The first thing you should do is to tell other peoples who you are.

```
git config --global user.name "zhaoyan"
git config --global user.email zhaoyan.hrb@gmail.com
```

- In windows, .gitconfig will be saved in C:\Documents and Settings\Administrator directory.
- `$ git config -global core.editor notepad ↵` (use notepad as default editor)
- git config –list list all the configuration command options
- /.gitconfig template file, it will customize your own git behaves. (color and log alias is more useful). By now, I didn't use git very often, so I didn't give shorter alias name to each command.

```
[alias]
co = checkout
ci = commit -s   ## -s means to add name and email. Important when working with
st = status
br = branch
oneline = log --pretty=oneline --since='2 days ago'
onelog = log -p -1
[color]
status = auto
branch = auto
ui = auto
```



GitHub or gitlab

- For gitHub, I have two github account:

  1. zhaoyan.hrb@gmail.com zhaoyan;
  2. yan.zhao.74@gmail.com YanZhao

  A project hello-world, Main project is on zhaoyan, then I use YanZhao account fork main project. project link is :
  git://github.com/zhaoyan/hello-world.git(read only)
  git@github.com:zhaoyan/hello-world.git(modifiable)

- `$ ssh-keygen -t rsa -C yan.zhao.74@gmail.com ↵`

- Your public key has been saved in /c/Users/zhao/.ssh/id_rsa.pub(for windows) /.ssh/(for Linux). Then you should paste the public key to the account in github.com.

- You can copy id_rsa.pub and id_rsa to other computers. so you don't need to run ssh-keygen command any more. But when you copy id_rsa to linux or Mac, you need `$ chmod g-r id_rsa ↵` to make your private key only readable for youself. Or when you push or fetch, git will refuse your request. (ssh -v will give you verbose information. You must restart ubuntu after you move id_rsa to /.ssh).

- once you finish it, `$ ssh -T git@github.com ↵` will test you key setting. (github has details explanation.)

- In your .ssh directory, if you find a id_rsa file, don't change it. Because this file maybe used to connect some remote server. You can build or modifiy config file in /.ssh diectory. Add something below. 1) you can use your own id_rsa_old name here, how to use ssh-keygen to product different name, you can see manual. 2) **Don't add Port 443 statment first, if you use ssh -T to get no response, then add Port 443, then it will work**.

  ```
  Host github.com
     Hostname ssh.github.com
     Port 443
     IdentityFile ~/.ssh/id_rsa_old
  ```

- Sometimes, you can't see project link, you can click the little eye(watchers) on the right upper corner. pull request will show up.
- **Don't use any Chinese in version control, If your English is not good, use Pinyin.**
- In order to do anything in Git, you have to have a Git repository. This is where Git stores the data for the snapshots you are saving. There are two main ways to get a Git repository. **One way** is to simply initialize a new one from an existing directory, such as a new project or a project new to source control. **The second way** is to clone one from a public Git repository, as you would do if you wanted a copy or wanted to work with someone on a project.

GUI

- There are two main GUI usages in git: 1) gitk 2) gui diff and merge
- gitk is GUI application. gitk –all will list all the branches.
- For git diff and git merge commands, you can use some gui tool. For Linux, meld is the best option. You need to change you .gitconfig to add [diff] and [difftool] section. Pay attention. They finish the same task. Make you `$ gui difftool ↵` invoke the GUI diff(merge) application. You need to configure in this way. For mergetool, You should select one of two options. Usually, It will put your local workspace file left, and you merged branch right. and middle is your last result. I prefer to put $BASE in the middle. But you can change it in your .gitconfig anytime.

```
[diff]
    tool = meld
[difftool]
prompt = false
[difftool "meld"]
cmd = meld "$LOCAL" "$REMOTE"
[merge]
    tool = meld
[mergetool "meld"]
path = /usr/bin/meld
keepBackup = false
trustExitCode = false
```

```
# Choose one of these 2 lines (not both!) explained below.
cmd = meld "$LOCAL" "$MERGED" "$REMOTE" --output "$MERGED"
    cmd = meld "$LOCAL" "$BASE" "$REMOTE" --output "$MERGED"
```

- For Mac, I haven't test if meld can be used. But I have tried diffMerge, On its website, you can find how to config .gitconfig file.Detail can be found in diffMerge website. I will not duplicate it.

- For diff, you will not change git diff command. Only git difftool invoke GUI. For Merge, git merge just modify your file in work space to diff format when there is conflict. Then you need to run `$ git mergetool ↵` to invoke GUI application. Just remember: **First run git merge, if conflict, then git mergetool**

- When you run `$ git mergetool ↵` Meld will open three panel windows. Middle one is your result. left and right are two conflict branches. When you finished. just click save and exit. Then run `$ git status ↵`, you can see it is ready for to add and commit.

2.4.2 Basic commands

status log show

- `$ git status ↵` you need to use this command very often.

- `$ git status -uno ↵` **will not show untracked files.** `$ git log -author=Bob ↵` only show Bob's commit.

- `$ git status -s ↵` show two columns , the first is staging, the second is working tree. if you modify a file, then add. then you modify a file again. Now git status -s show MM a. guess what it means?

- `$ git log ↵` can show you all the commits history.

```
git log --pretty=oneline #just show simply log information
git log -p   #show ci log and source modification each commit.
git log -2  #just last two commits
git log --after 2015-12-01  #show all commits after date
git log --oneline  #simple informaitons
git log --abbrev-commit --pretty=oneline #simple informaitons
git log experiment..master #show commits on master, not on experiment
        a--b--e--f(master)
            \c--d(experiment)
git log origin/master..HEAD #what have you push to remote?
```

- `$ git show ↵` to examine a single object. such as `$ git show v1.0 ↵` and`$ git show master:book.tex ↵` will output source file. It can used on commit(just like log command), or tree (just like cat-file -p). or plain blobs,(just like cat-file -p).

add commit

- When you add, You should know:
  1. There is tree in index, but we don't give this tree a sha value,
  2. For any new added file, produce a blob object, and assign a sha value to it, and add it to tree object.
  3. For any added file, produced new blob object, because sha has changed. Updated tree object.
  4. When you commit, give a tree sha value, and copy whole tree out, add tree object to commit object, and give commit object a sha value.
- You can use `$ git status -uno ↵` to hide all untraced files. You also can produce a **.gitignore** file in you project, it will work on present and all children directory. When you use `$ git status ↵`, It will not show many untracked files. Example is below. edit .gitignore file.

```
# comment
*.a   #ignore all files with .a extention
!lib.a  #but include lib.a
build/   #ignore all file in build directory
doc/*.txt  #ignore all txt files in doc/ but doc/server/arch.txt will be includ
```

- You should avoid using `$ git add . ↵` to add so many unnecessary files. If you run it accidentally, You can use `$ git reset ↵` to revert add which you just ran. Or you can use `$ git rm -r -cached . ↵` to delete it from index, but keep file in working directory.
- When there are a lot of files to add, `$ add -i ↵` is a good choice. `$ add -u ↵` will only add tracked files, no dot is needed.
- Current working directory is (1), Index file or staging is (2) and Git local repository is (3)
  (1) -> (2) -> (3)
  `$ git add ↵` (1) -> (2)

`$ git commit ↵` (2) -> (3)
`$ git commit -a ↵` (1)->(3) Don't recommend use it.

`$ git add . ↵` add all files. Be careful to use this command
`$ git add -u ↵` it's very often used command. or It can be used in such situation: rm *.txt, then git add -u will delete *.txt from staging.

- commit command usually don't pay attention to files or directory, it just product a commit to produce a sha object(commit).
- `$ git commit -amend ↵` don't produce new commit, just modify last commit. But it will change previous commit sha value.
- If you add a file a.c, but you don't commit it. How to revert? `$ git reset a.c ↵` Will remove a file named a.c from the current index, the "about to be committed" area, without changing anything else. contrary to `$ git add -a.c ↵` To undo git add . use git reset (no dot).
- If you haven't commit yet, so you can't use reset now. At this time you can use `$ git rm -cached ↵` command, then commit it again.
- Previous method will produce two commits. if you want to just modify current commit. git rm first, then use `$ git commit -amend ↵`. Or you can also use `$ git reset HEAD ↵`, then maybe you need git add or not, depends on your context. last `$ git commit -c ORIG_HEAD ↵`. Here reusing the old commit message. reset copied the old head to .git/ORIG_HEAD; commit with -c ORIG_HEAD will open an editor, which initially contains the log message from the old commit and allows you to edit it. If you do not need to edit the message, you could use the -C option. **This method only reuse old commit message, not very useful in my view point**
- Both(commit –amend) and (reset... commit again) will produce new commit. In fact, any commit sha include current operation second will be different.
- **So don't change commit which you pull from public or you have pushed to public**
- You can use `$ git ls-files -s ↵` to see all files in the index and their corresponding sha value.

- You can use `$ git ls-tree -l HEAD ↵` to see all files in the HEAD commit.

remove rename

- Basic command:

```
git rm a // It will delete from work space and index
git rm --ached a  //just delete a from index.
git commit -m "delete file a"

git mv a b  #is three command: mv a b; git rm a ; git add b ;
git commit -m "rename a to b"
```

- git rm will delete file from working directry, git rm –cached will only delete file from index. **They have nothing with commit**

diff

- diff basic usage. Usually, you should follow a file name after diff command.

```
git diff     ##(1) and (2)
git diff -cached   ##(2) and (3)
git diff HEAD   ##(1) and (3)

git diff tag     ##tag and HEAD
git diff tag file    ##just compare a file (only one file name)
git diff tag1..tag2   ##two tags( you can omit two dots)
git diff SHA11..SHA12  ## two commits
git diff tag1 tag2 file or  git diff tag1:file tag2:file

#tag can be a alias of remote
git remote add xjsff git://github.com/xjsff/hello-world.git
git diff xjsff/master README  ##local README and README in xjsff/master
```

- `$ git diff -name-only ↵` just show changed files name. so you can compare it one by one. `$ git diff -name-status ↵` will show how do you changed files, add, delete or modify.

- Before checkout or reset, you'd better to use diff command to see if there are important content to avoid overwrite. that is a good habit.

- diff -u is good command, it will show context of difference. All the differences give by differences section.

- After git fetch, you can use `$ git diff master origin/master ↵` to see all the modifications, then decide if you want to merge. fetch+merge is better than pull.

- `$ git difftool - file ↵` to use meld to see the difference, it looks much better.

checkout reset

- **branch->commit, HEAD->branch, When not follow paths, checkout move HEAD, reset move (HEAD->branch, HEAD must point to a branch).** It's helpful for you to understand these two command.

- If HEAD->master, reset will move HEAD->master together, if HEAD->commit, reset will only move HEAD. at this time, It's still detached head status.

- There are two basic different usages for checkout:

  1. with paths(file), it just with <commit> to replace index and work space. At the same time, It will not change HEAD.
  2. without paths, if you follow a branch it will move HEAD to branch.
  3. without paths, if you follow a old <commit>, It will move HEAD to old <commit>. HEAD will be a ref to a branch, When It points a real old commit, It will be "detached HEAD" and all commits after HEAD may be discards in the future.
  4. `$ checkout <commit> - paths ↵` or `$ checkout branch ↵`. Don't use in`$ git checkout <commit> (empty) ↵` without paths, It will put HEAD in detached state. **After checkout, follow a branch, or follow /patth/file name.**
  5. `$ git checkout ↵` just like `$ git status ↵`

- When checkout and reset follow file, they can be a file, *.cpp(some files) , . (all files), /path_name(path_name and recursive) and /path_name/*(no recursive).

- If you are in detached HEAD state, 1)You don't modify work space, just use checkout master to set HEAD to master again. 2)If you modify work space and you want to keep it. commit it first, remember <commit-sha>. Then checkout master. merge <commit-sha>. Then HEAD will move to master and master will point to merged commit.

- If there is no commit following the checkout command, only file, it will use file in index to overwrite work sapce. `$ git checkout .` `or git checkout file` ↵ (2)->(1).

- If there is commit, `$ git checkout HEAD .` ↵ (3)->(1) and (2) dot represents all the files. **This is a dangerous command. Save or commit you local work first.**

- **checkout will overwrite work space directly, it is not like merge, and doesn't produce conflict files. So be careful!**

- you can checkout a file from different history. :

  1. `$ git checkout v1.2.3 - filename` ↵ tag v1.2.3
  2. `$ git checkout stable - filename` ↵ stable branch
  3. `$ git checkout origin/master - filename` ↵ upstream master
  4. `$ git checkout HEAD - filename` ↵ the version from the most recent commit
  5. `$ git checkout HEAD^ - filename` ↵ the version before the most recent commit
  6. `$ git checkout xxxx -filename` ↵ xxxx is commit version number.

- Similarly, There are two different usages for reset:

  1. with paths, it will not move HEAD and master (reset commit). `$ git reset <commit>- <paths or filename>` ↵ will overwrite index with commit, (3)->(2). it just like contrary operation of add.
  2. Without paths, it will reset (HEAD–>master) together to a new <commit>, and all commits after reset commit will be discards(delete commits, dangerous!). Because It reset (HEAD–>master) together, then It doesn't have detached head problem.
  3. Without paths, there are three options you can use:

```
git reset [--soft|hard|mixed ] <commit>
## soft just reset commit
## mixed reset commit and (3)-->(2)
## hard reset commit and (3)-->(2)-->(1)
```

- How to save from wrong reset command? When you reset to a <old-commit>. All the commits after <old-commit> will not found easily unless you can remember all the commit sha value. A better method can be use `$ git reflog show master | head -5` ↵. It will show master@0...n. It represent all the master ref history. You also can use it to show head moving history.

- HEAD@num can be used as follow to change commit history after a <old-commit>

  1. `$ git checkout <old-commit>` ↵ # detached head
  2. `$ git commit ...` ↵ #last commit sha is 123abc
  3. `$ git checkout master` ↵ # move head to branch
  4. `$ git reset -hard HEAD@1` ↵ # reset it to old commit 123abc.

- followed by <old-commit>, both checkout and reset –hard will modify all files in work space.

- If you modify a file in (1), you can restore it from (2) with `$ git checkout -a.c` ↵ or from(3) with `$ git checkout HEAD -a.c` ↵. They will overwrite (1) forever, so be careful. If you want to keep it, you can do:

tag

- push tag to remote. `$ git push origin <tag_name>` ↵ And the following command should push all tags (not recommended): `$ git push -tags` ↵

stash

- Sometimes I have a situation that I am working on some feature on my own branch and suddenly someone comes to me and says that something really important has to be fixed or improved on the main branch. Usually it happens when I am in the middle of very important changes which are not ready to be committed for some reason.

- Normally, I would have to save the changes (diff) into some file, switch to the main branch abandoning any changes, apply the fix or improvement and commit it. Then I could switch back to my own branch, apply the changes (patch) from the file and continue the work. While it is not something difficult, it can be done much easier with Git.

- When you use `$ git stash` ↵, It will run `$ git reset -hard` ↵ automatically. so all you work will disappear. you can use `$ git statsh pop` ↵to revert it.

- stash will save both working and index.

- stash will only save files which have added to index.

- **After stash, system will call reset –hard HEAD automaticlly, so if you have new files, you'd better add it to index before you use stash command.**

```
You need to modify a bug in release version. First stash, then checkout release
git stash save "you messaage" #
git stash list #list all stash
git statsh pop #
```

rebase revert

- Reverting has two important advantages over resetting. First, it doesn't change the project history, which makes it a "safe" operation for commits that have already been published to a shared repository. For details about why altering shared history is dangerous, please see the git reset page.

- Second, Git revert is able to target an individual commit at an arbitrary point in the history, whereas git reset can only work backwards from the current commit. For example, if you wanted to undo an old commit with git reset, you would have to remove all of the commits that occurred after the target commit, remove it, then re-commit all of the subsequent commits. Needless to say, this is not an elegant undo solution.

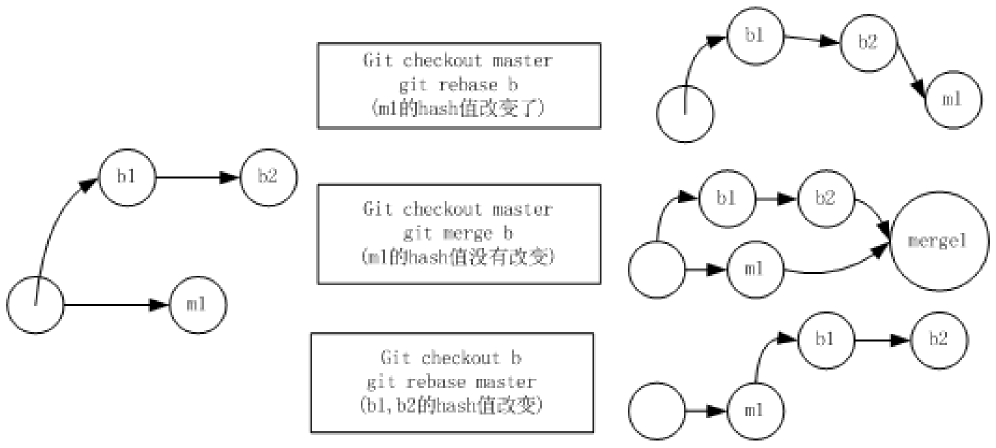
- revert will produce a new commit too. `git revert HEAD^^` produce a new commit

- **Never rebase branches or trees that you pulled. Only rebase local branches. Never ever rebase a branch that you pushed, or that you pulled from another person**
- rebase command steps: git rebase master
  1. now you are in branch1,
  2. All changes made by commits in the current branch but that are not in master are saved to a temporary area.
  3. reset –hard master
  4. The commits that were previously saved into the temporary area are then reapplied to the current branch, one by one, in order.
- You can ommit the –onto, it will
- rebase command and merge command is little different. **When you are in test branch, master branch has a new commit, You can rebase on new commit on master. Because in the end, your test branch will be merged back to master, rebase often can save you a lot of conflict problem in the future.**
- **When you in the master branch, and When test branch has finished or partly finished, You can use git merge command to merge test branch into master branch**
- You can think merge and rebase are two commands operate on branch.
- rebase basic usage. 1) in master branch , commit a new commit (m1), in test branch, commit few times(t1..t2).

```
    t1-->t2
  /
m-->m1
```

2) checkout test branch, then `$ git rebase master.` ↵ /* means that SHA value has been changed.

```
        t1*-->t2*
      /
m-->m1
```



- rebase has –onto options. Detail can be found in git rebase document. Just google it.
- rebase branch1 means 1) you are not in branch1 now 2) branch1 is melding point, 3) all commits value will be changed.
- You can use rebase on current branch, in this way, you want get clean history, and most of time, you use it with -i option.
- You can use rebase to rebase your branch work on master branch. it's another usage of rebase.

remote push pull fetch

- Fetch is not just a child command inside of pull, from result, you can think that "pull = fetch + merge". But, in fact, Fetch is extract all the remote branches to local. It's configured in remote subsection in .git/config
- origin is local alias of remote repository url. You can see it in .git/confg file.

```
[remote "origin"]
fetch = +refs/heads/*:refs/remotes/origin/*
url = git@github.com:zhaoyan/new_doc.git
```

- For push and pull, you have to make sure you are in one branch, and it will only push and pull one branch from remote.

- You have to make sure local branch tracking the remote one. In anoter word, for a branch, you should see a config content in .git/config

  ```
  [branch "master"]
  remote = origin
  merge = refs/heads/master
  ```

- Now, if there is branch on remote, you can first fetch, then git checkout -b local-branch-name remote-branch-name. In this way, a configure content will be added to .git/config automaticlly, and you local-branch will track remote one.

- track means that when you in local-branch, you can run push and pull without any argument to syncronize local and remote branch.

- If have a local branch, but remote doesn't have one. You have two ways:

  1. If you git version is greater than 1.7, then you can use git push -u origin <branch>. -u is important option, and if you want to change branch name, you can use loca-name:remote-name sytastic trip.
  2. If you use old version, use git push origin branchB, then add below to .git/config file

     ```
     [branch "branchB"]
         remote = origin
     merge = refs/heads/branchB
     ```

- don't use pull, just use fetch, then merge. When you run `$ git merge origin/master master ↵`, there are three possibilities.

  1. Working directory no modification, then fast-ward merge and working and index will be updated.
  2. Working directory has modification, merge will fail.
  3. Working directory has modification and commit, merge maybe ok, then working and index will be updated. merge maybe conflict, then use merge tool to resolve conflict and commit to produce a commit manually.

- origin is a alias of remote repository, `$ git remote add origin git@github.com:zha` ↵just give a alias name, It will not real create remote repository, you need to log in github to create it manually. Beside add, you can show, rename and delete a these alias. with these alias, you can `$ push or fetch origion ↵` directory, don't need to write the address of the remote repository.

```
git remote add paul git://github.com/paul/test.git
git remote -v
git remote show paul #show paul all the informations, including branch.
git remote rename paul pa
git remote rm pa #delete pa, because he will not contribute the system.
```

- push command is followed by a repository name and a branch name. such as `$ push origin master ↵`. It has a lot of syntax, can be use change remote branch name and delete remote branch.

  1. git push origin experiment #push a branch to server
  2. git push origin local:experiment #change local branch name, and push to server.
  3. git push origin :experiment #delete local branch ,use empty name
  4. git push origin erperimental:experimental-by-yan #give remote-tracking-branches other nameïijŇhere experimental-by-yan is remote-tracking-branches nameïijŇerperimentallocal name. <source-name>:<destination-name>

2.4.3 History

history representation

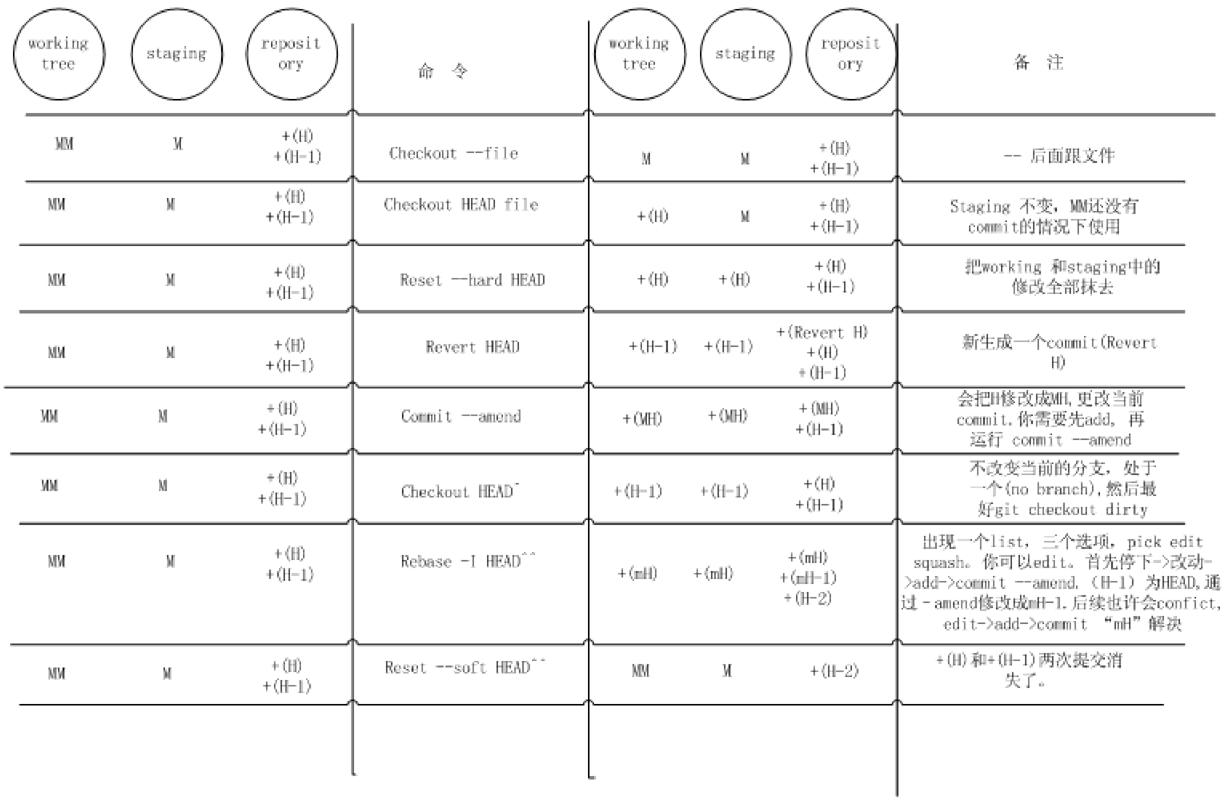
- No space in tag name.
- Common used history commit ref name list:

```
ORIG_HEAD, COMMIT_EDITMSG
HEAD, MERGE_HEAD, FETCH_HEAD

HEAD^: #HEAD's parent, it's ORIG_HEAD
HEAD^ or HEAD^1   # first parent
HEAD^^ or HEAD^2 # second parent
HEAD~4 :

HEAD:README.txt #A file inside a commit
```

history change



working tree	staging	repository	命 令	working tree	staging	repository	备 注
MM	M	+(H) +(H-1)	Checkout --file	M	M	+(H) +(H-1)	-- 后面跟文件
MM	M	+(H) +(H-1)	Checkout HEAD file	+(H)	M	+(H) +(H-1)	Staging 不变，MM还没有commit的情况下使用
MM	M	+(H) +(H-1)	Reset --hard HEAD	+(H)	+(H)	+(H) +(H-1)	把working 和staging中的修改全部抹去
MM	M	+(H) +(H-1)	Revert HEAD	+(H-1)	+(H-1)	+(Revert H) +(H) +(H-1)	新生成一个commit(Revert H)
MM	M	+(H) +(H-1)	Commit --amend	+(MH)	+(MH)	+(MH) +(H-1)	会把H修改成MH,更改当前commit.你需要先add，再运行 commit --amend
MM	M	+(H) +(H-1)	Checkout HEAD^	+(H-1)	+(H-1)	+(H) +(H-1)	不改变当前的分支，处于一个(no branch),然后最好git checkout dirty
MM	M	+(H) +(H-1)	Rebase -I HEAD^^	+(mH)	+(mH)	+(mH) +(mH-1) +(H-2)	出现一个list，三个选项，pick edit squash。你可以edit。首先停下->改动->add->commit --amend.（H-1）为HEAD,通过－amend修改成mH-1.后续也许会conflict,edit->add->commit “mH”解决
MM	M	+(H) +(H-1)	Reset --soft HEAD^^	MM	M	+(H-2)	+(H)和+(H-1)两次提交消失了。

- **Three basic modification history usage:**
  1. Delete or combine certain commits;
  2. Delete new commits after a point
  3. Delete old commits before a point
- **Three common commands for modifying history: checkout, cherry-pick and reset. Other useful command is rebase, you can think that it's a combination of previous three commands. Detail can be seen in rebase command**
- **Change current latest commit** `$ git commit -amend ↵` not only change the message, but also the working tree contents.
- revoke latest commit.

```
git commit -m "Something terribly misguided"           (1)
git reset HEAD~                                         (2)
```

```
<< edit files as necessary >>                              (3)
git add ...                                                (4)
git commit -c ORIG_HEAD                                    (5)
```

1. This is what you want to undo
2. This leaves your working tree (the state of your files on disk) unchanged but undoes the commit and leaves the changes you committed unstaged (so they'll appear as "Changes not staged for commit" in git status and you'll need to add them again before committing). If you only want to add more changes to the previous commit, or change the commit message1, you could use git reset –soft HEAD instead, which is like git reset HEAD but leaves your existing changes staged.
3. Make corrections to working tree files.
4. git add anything that you want to include in your new commit.
5. Commit the changes, reusing the old commit message. reset copied the old head to .git/ORIG_HEAD; commit with -c ORIG_HEAD will open an editor, which initially contains the log message from the old commit and allows you to edit it. If you do not need to edit the message, you could use the -C option. **-c option is just reuse last time commit message**

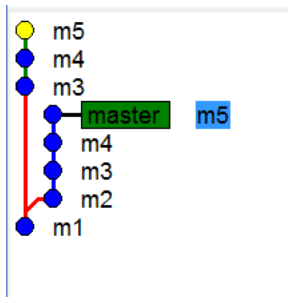
- **Delete some new commits** `$ git reset -hard HEAD^^^ ↵` will delete `HEAD^^` and `HEAD^` and HEAD three commits forever, It's is a dangerous command, be careful about it.

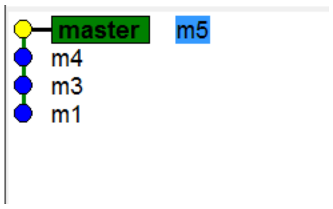
- **Delete all old commits before A**

  1. `$ echo "Commit from tree of tag A" | git commit-tree A^{tree} ↵` It will produce a <SHA value>. This command will produce a new commit with <SHA value> and without any parent. It's a root commit.
  2. `$ git rebase -onto <SHA value> A master ↵` All the history behind of A will be deleted.

- **Delete some old commits by cherry-pick**

  1. `$ git checkout C ↵`, C is a tag, by now, working tree is C status, Here, you can't use `$ reset C ↵`, because it will put master to C, then D, E and F will not be accessed.





2. `$ git cherry-pick E and F ↵` , delete D commit ,cherry pick will produce a new commit SHA value, such as E* and F*.
3. `$ git checkout master ↵` It will end Head detached status.
4. `$ git reset -hard HEAD@{1} ↵`. reset HEAD–>master to new F* commit.

- **Delete some old commits by rebase**
  1. `$ git rebase -onto m1 m2 m5 ↵`, You need to know that m2 m5 will not include m2, It will produce a new commit m3, m4 and m5, SHA value will be different. And from the figure, you can see that master branch is still on old m5 commit. so you need perform step 2.
  2. `$ git rebase -onto m1 m2 master ↵` will keep you track master branch, after that you don't need to run step 2 below. It will be easy way to do that.
  3. `$ git checkout master ↵` and `$ git reset -hard HEAD@{1} ↵`. After these two commands, you can see m2 has disappeared.

- **Combine two old commits by cherry-pick**
  1. `$ git checkout D ↵`, by now, working tree is D status,
  2. `$ git reset -soft B ↵` , because HEAD is D now, go to B, –soft means working tree is still D status
  3. `$ git commit -C C ↵` then `$ git cherry-pick E and F ↵`.

4. `$ git checkout master ↵$ git reset -hard HEAD@{1} ↵`. Detail can be found in "Got Git".
5. **It will not produce conflict, just delete a commit, if you want to delete modification inside a commit, maybe it will produce conflict**.

- **Combine two old commits by rebase**

  1. `$ git checkout D ↵`, D is a tag, by now, working tree is D status,
  2. `$ git reset -soft B ↵` , because HEAD is D now, go to B, –soft means working tree is still D status
  3. `$ git commit -C C ↵`
  4. `$ git tag newbase ↵` **give a tag name to avoid remember sha-value**
  5. `$ git rebase -onto newbase D master ↵` then check `$ git branch ↵`to see if it's on branch master, if it's yes
  6. `$ git tag -d newbase ↵`
  7. Detail can be found in "Got Git".

- **difference cherry pick and rebase**

  1. rebase and cherry pick will change commit SHA value, so don't use it on any commit that you have pulled or you have pushed. cherry pick will not change
  2. cherry pick should be used in change few commits, and rebase can be used to change a lot of commits at the same time.

- **delete all the commits in a merged branch**

  1. `$ git rebase -i ↵` will open an editor, then you can give some commands inside this file. the name of this file is git-rebase-todo file.
  2. git rebase -i will run it according to this file.
  3. git rebase -i <sha before the branches diverged> this will allow you to remove the merge commit and the log will be one single line as you wanted. Detail can be seen rebase command.
  4. A detail can be seen in google "squashing commits with rebase"

blame

```
git blame -L 12,13 hello.html
\\Who made some modification.
```

2.4.4 Branch

Basic conception

- Usually, there are three common use branches. One is release branch which can be based on one release commit. Usually, it used to fix some bugs in release, then you can merge back master branch. Another is feature branch, when you add a feature, and you don't know if it's good or can be finished properly, you can produce a feature branch. At the same time, keep master branch keep growing.

- Feature branch is usually based on current HEAD, and release branch is usually based on one release tag. So you need to build a tag first.

- After you finish your Feature branch or Release(bug fix) branch. There are two options, if only you work on the branch, such as Feature branch, you can merge it back to master branch, and push master to remote.

- Or if other people also need this branch, such as Release branch, you need to push it to remote, so other people can update their own work according to your work. How to push local branch to remote and keep track-able, see remote section.

- **For Release branch, you should switch back master, and pick or merge commits in Release branch back to master branch**

- **For Feature, branch, If developing-time is short, just merge it back master, and push master to remote. If developing-time is long, At this time, There is new commits on master, at this time, you should pull back master, and perform rebase command**.

- If you project is based on other work, you also need vendor branch. You don't commit on it. just keep track with upstream new version , then merge back with you master branch.

- when you want to merge others work, you'd better build a branch first.

- Branch is based on commit, When you merge a branch back to master, you may delete it if you don't need it anymore.

- For branch, you can undo a merge commit, or you can delete all the commits in one branch after you have merge. Detail can be seen in section History

Branch command

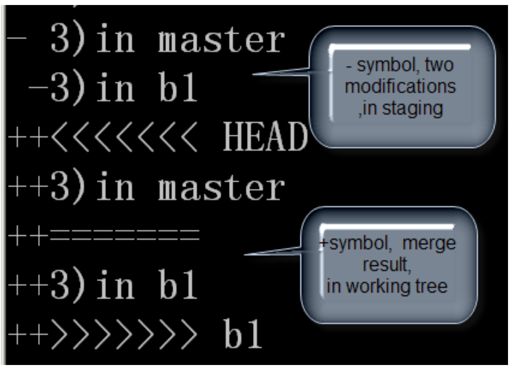
- checkout -b can be thought as short hand of two commands 1) branch create a new branch, and 2) checkout to switch it.

  1. `$ git branch -r` ↵ list all the branches on the remote
  2. `$ git branch` ↵ list all the local branches
  3. `$ git branch -D b1` ↵ delete the b1 branch forcefully
  4. `$ git branch -m master mymaster` ↵ rename the branch
  5. `$ git checkout -b new-branch master` ↵ new-branch based on master and switch to it.
  6. `$ git checkout -b new-branch` ↵ build new-branch based on current branch.
  7. `$ git checkout branch-name` ↵ switch to branch-name

- merge command is followed by two branches, no files name. such as `$ git merge origin/master master` ↵

- Three merge results:

  1. Fast-Forward merge, You don't need run commit it.
  2. without conflict merge, git will produce a new commit automatically.
  3. conflict merge , Git will not resolve conflict for you. it will put working tree into a specify conditions. (file in work space has been modified with conflicted format). you need manually resolve it.

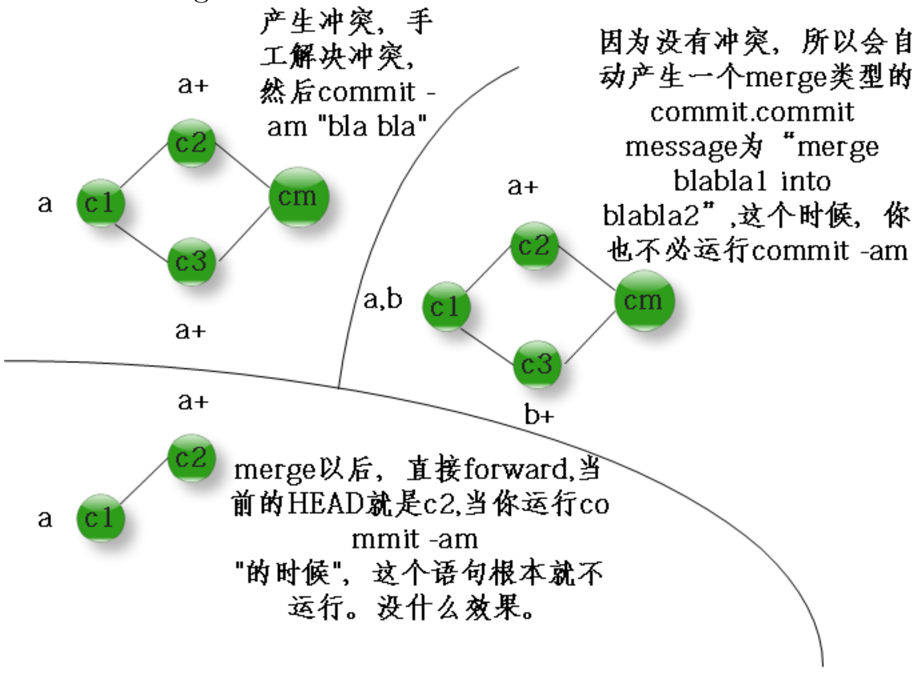


```
- 3)in master
 -3)in b1
++<<<<<<< HEAD
++3)in master
++=======
++3)in b1
++>>>>>>> b1
```

- symbol, two modifications ,in staging
+symbol, merge result, in working tree

After you resolve conflict, run git add file-name and git commit.

4. You can see figure below to understand it better.


产生冲突，手工解决冲突，然后commit -am "bla bla"
a+
a
c1
c2
c3
cm
a+
因为没有冲突，所以会自动产生一个merge类型的commit.commit message为 “merge blabla1 into blabla2”,这个时候，你也不必运行commit -am
a+
a,b
c1
c2
c3
cm
b+
a+
a
c1
c2
merge以后，直接forward,当前的HEAD就是c2,当你运行commit -am "的时候"，这个语句根本就不运行。没什么效果。

- Three kinds of merge:

  1. straight merge. Merge all the commits in one branch into another.
  2. squash merge. Change all history into one commit. `$ git merge -squash contact ↵`. You need merge first, then commit.

     ```
     git checkout master
     git merge --squash bugfix
     git commit
     ```

  3. cherry-pick. git cherry-pick 32176f(another commit in other branch). or git cherry-pick -n 32176f, which n means that you don't want

commit immediately. When you git commit later, without m, then editor will open, all cherry you picked commit message will be in this editor.

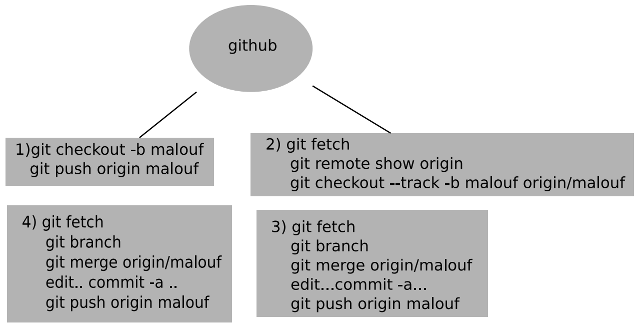
- `$ git merge-base b1 b2 ↵` found the common ancestor
- `$ git cherry -v master test ↵` In master branch, found all commits in test, but not in master.
- You've already committed the merge that you want to throw away, use this command: `$ git reset -hard ORIG_HEAD ↵`.

Remote Branch

- Branch can be in two different positions. One is in the local place, you can use git branch to check them. The other is remote-tracking branches, you can use git branch -r to check them. The name format is "origin/remote-branch-name", origin is not branch name, it's repository name. About how to synchronize local and remote branch? and how to make local branch is track-able with remote branch. you can see remote subsection.
- You can't checkout remote branch, such as `$ git checkout origin/branch1 ↵`. In order to do so , you have to create a local branch based on remote branch, such as `$ git checkout -track -b branch1 origion/branch1 ↵`, then work on the local branch1. after you finish it, you can push it back. with –track, you can push or fetch without specify remote repository name. –track option can be omitted here.
- `$ git checkout -b new-branch origin/new-branch ↵` build new-branch based on origin/new-branch. And add track ability. In this way, If you are in the local new-branch, you can push and pull directly.

Branch daily usage

- Single person, center control, with branch. When you finish the branch, you can merge it with master.
- Below figure show two different position are working on the same branch–malouf.



- Multi person, center control, with branch

  1. `$ git fetch upstream` ↵ get zhaoyan upstream the updated development process.
  2. `$ git checkout -b test_bla upstream/master` ↵ based zhaoyan master branch, build local branch test_bla (any name is OK.)
  3. edit, compile,
     git commit -am "s1"
     edit, compile,
     git commit -am "s2"
     While, upstream/master maybe changed, a new commit t1 appear.

     ```
          s1--- s2
     ____/___t1
     ```
  4. `$ git fetch upstream` ↵ when you want to update upstream, you have to get new commits
  5. `$ git merge upstream/master` ↵ If no conflict, produce a forward merge, if conflict, manually resolve and commit a new commit. `$ git commit -am "st1"` ↵

     ```
           s1--- s2
     ____/___t1___\st1__
     ```
  6. Or,**A more recommended method is** you can use the second method: `$ git rebase upstream/master` ↵ No conflict, produce a new commit. with conflict, manually resolve it, then `$ git add conflict_file` ↵, then `$ git rebase -continue` ↵, not need to commit.

     ```
             t1---s1*--- s2*
     ____/
     ```

7. git push origin test_bla
8. login github, look for you repository, checkout test_bla, then click "request pull" button.
9. If other accept you request, you can delete test_bla branch. `$ git branch -D test_bla ↵`
10. `$ git push origin :test_bla ↵` delete test_bla branch in github.
11. repeate 1-10.

2.4.5 conflict

- In git, there are a lot of commands will cause conflict, such as revert, merge, cherry-pick,rebase and so one. If you see document, you will find all comand are with –continue and –abort command options.
- **All command will perform three-way merge command, It will merge two commit based on two commit common ancestor.**
- c1 is commit 1, c2 is commit 2, and A is common ancester. Basic idea is use diff command to produce diff format information from A to c1. such as "2d1", which represent delete the second line in A and produce c1, in c1 there is only one line. Also produce diff format information from A to c2, such as "2a3,4", which represent from the second line in A, add another two line in c2. Then if two diff bot include number "2", it means that there is conflict, you need to resolve it manually.
- From previous explanation, When you consider if it will conflict, you should not only consider c1 and c2, you should consider their common ancestor and a serical diff format information.
- For merge and rebase command, common ancerstor is very clear. but for some command, such as revert and cherry-pick, common ancestor is not very clear. You can think common ancestor is older(c1, c2)-1 commit is their common ancestor.

2.4.6 Daily usage

Clone a project from github

1. `$ git clone git@github.com:zhaoyan/hello-worl.git ↵` you can copy the address from github website. I think that it will produce origin alias automatically.

2. modify... main.cpp

3. `$ git add main.cpp ↵` `$ git commit -m "modify sth..." ↵`

4. `$ git log ↵` to see if you have commit successfully.

5. `$ git remote -v ↵` to see what origin is.

6. `$ git push origin master ↵` This command works only if you cloned from a server to which you have write access and if nobody has pushed in the meantime. If you and someone else clone at the same time and they push upstream and then you push upstream, your push will rightly be rejected. You'll have to pull down their work first and incorporate it into yours before you'll be allowed to push. detail can be seen in the previous push command: produce non-fast-forward error.

Add a project to a git

1. `$ cd test ↵`, run `$ git init ↵`

2. `$ touch test.cpp ↵`, then `$ git add test.cpp ↵` or `$ git add . ↵`. dot means all the files. Directories are added automatically when adding files inside them. That is, directories never have to be added to the repository, and are not tracked on their own. You can say "git add <dir>" and it will add all files in there.

3. `$ git rm README ↵` you can delete a file

4. `$ git commit -m "first commit" ↵` or `$ git commit -a -m "first commit" ↵` This is simple form. Combine add and commit together. Git commit -a can't add new files. If you add some news files, you should use `$ git add newfile ↵`. then commit.

5. `$ git log ↵` or `$ git status ↵` After committing, you can use log command to check if it has been submitted successfully.

6. **Then, login github, create repository with name test**

7. `$ git remote add origin git@github.com:YanZhao/test.git ↵`

8. `$ git remote -v ↵`you can check remote repository

9. `$ git push -u origin master ↵` You can not emit origin, if you don't specify branch, default is master. master is default local branch, you don't need to create it explicitly. this command push local content to the server.

Update a project from github

1. `$ git clone git@github.com:zhaoyan/hello-worl.git ↵` you can copy the address from github website.
2. modify... main.cpp
3. `$  git add main.cpp ↵ $ git commit -m "modify sth..." ↵`
4. `$ git log ↵` to see if you have commit successfully.
5. `$ git fetch ↵` to get origin/master.
6. `$ git diff master origin/master ↵` maybe you need to resolve conflict before you push.
7. `$ git merge origin/master ↵` If there are difference, you need to merge before you push, or it will produce non-fast-forward error. default is local master branch, you can't write in "git merge master" to skip "origin/master".
8. `$ git push origin master ↵`

Collaborate With Applications

- For codeblock, project file is pro1.cbp, It will produce two directory bin and obj, you need to modify .gitignore file, add bin/ and obj/ to ignore. good suggestion.
- For kdevelop:
  1. rm Makefile.in or .svn (if there are)
  2. go into the src directory, and run git init
  3. git add . and git commit
  4. git remote add origin git@github
  5. git push origin master
  6. another kdevelop, rm src
  7. git clone git@github src
  8. modify Makefile.am with you kdevelop project file.
  9. in configure dialog, Configure options->linker flager -> add -L./ and add lib to the debug/src directory.

10. compile and run.

- For latex, Only *.tex *.bib(reference) and /pic directory are useful, You need to add them manually.

single person on different computers

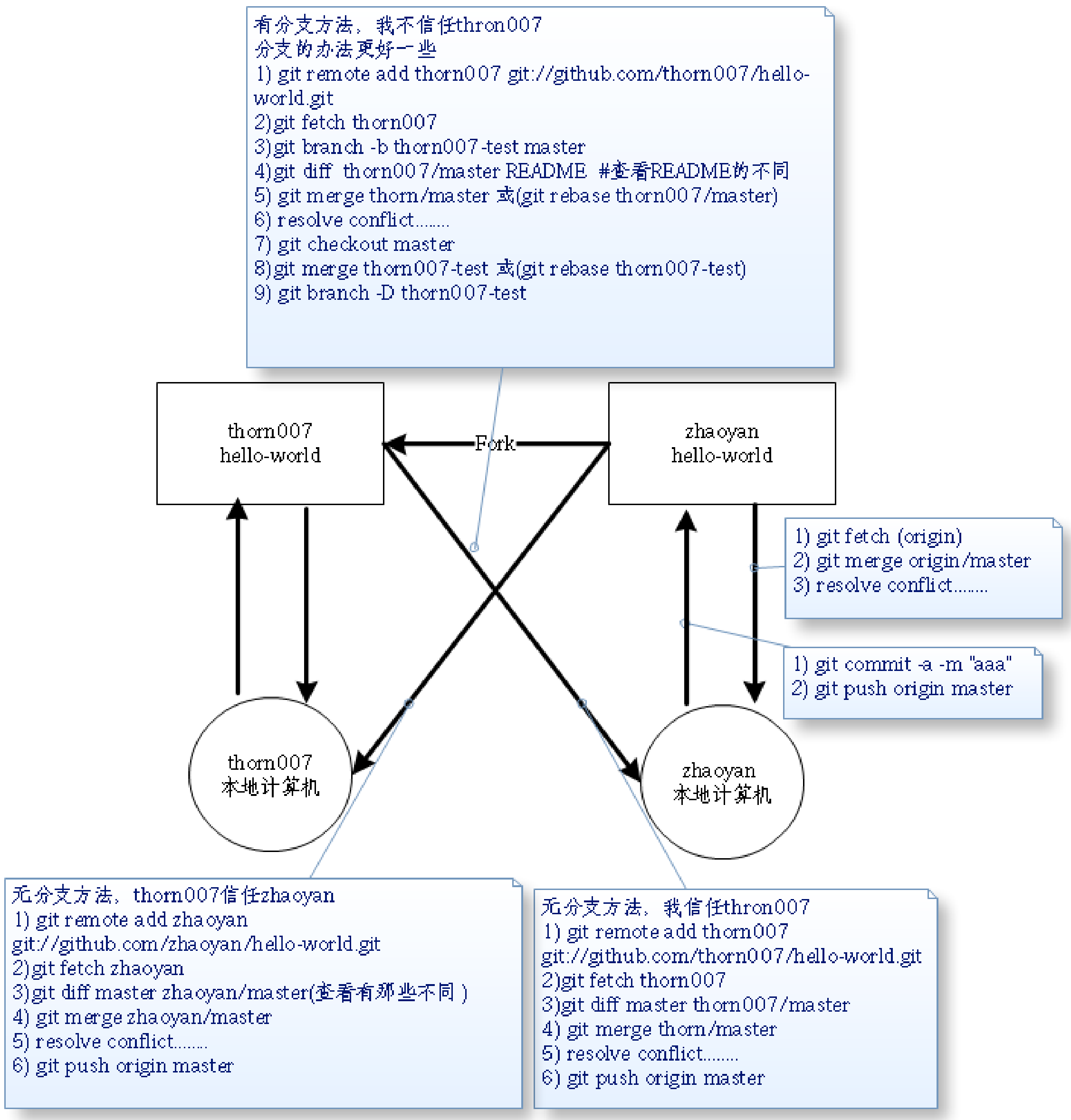
On computer 1:

1. `$ git log HEAD..origin` ↵ to check if there are differences. if true
2. `$ git fetch, git merge origin/master master` ↵ (git fetch give you more chance to examining it, it's better than pull)
3. `$ git commit -a -m " "` ↵
4. `$ git push` ↵
5. `$ git log HEAD..origin` ↵check push successfully

On Computer 2:

Same operation, when you merge, it will only produce forward merge, it's relatively easy.

cooperation



```
有分支方法，我不信任thron007
分支的办法更好一些
1) git remote add thorn007 git://github.com/thorn007/hello-
world.git
2)git fetch thorn007
3)git branch -b thorn007-test master
4)git diff  thorn007/master README  #查看README的不同
5) git merge thorn/master 或(git rebase thorn007/master)
6) resolve conflict........
7) git checkout master
8)git merge thorn007-test 或(git rebase thorn007-test)
9) git branch -D thorn007-test
```

```
1) git fetch (origin)
2) git merge origin/master
3) resolve conflict........
```

```
1) git commit -a -m "aaa"
2) git push origin master
```

```
无分支方法，thorn007信任zhaoyan
1) git remote add zhaoyan
git://github.com/zhaoyan/hello-world.git
2)git fetch zhaoyan
3)git diff master zhaoyan/master(查看有那些不同 )
4) git merge zhaoyan/master
5) resolve conflict........
6) git push origin master
```

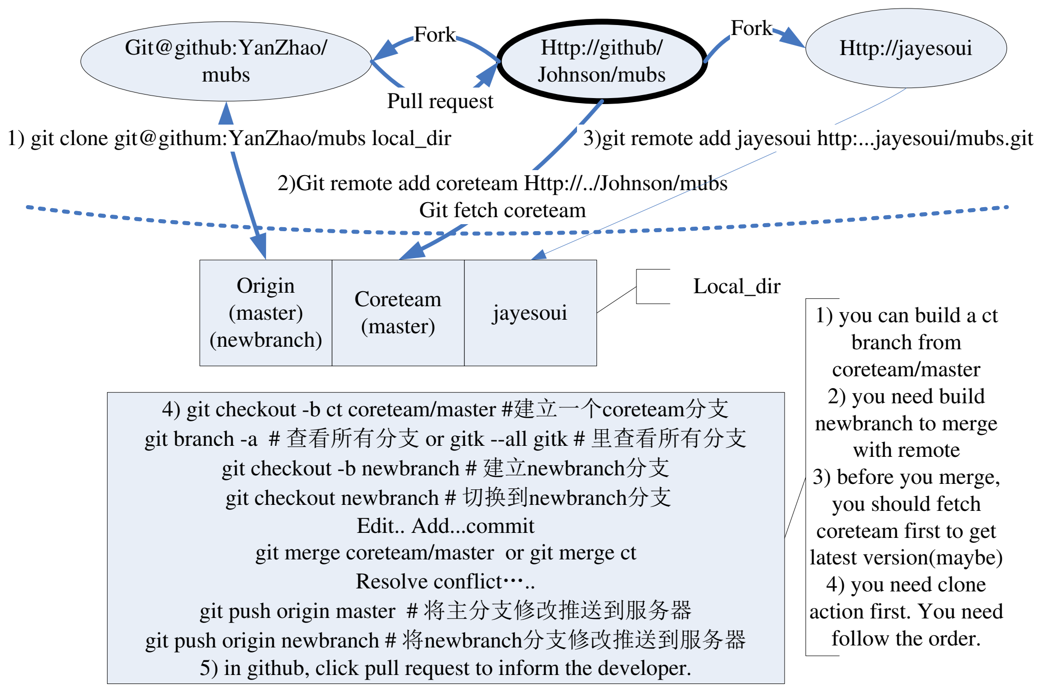
```
无分支方法，我信任thron007
1) git remote add thorn007
git://github.com/thorn007/hello-world.git
2)git fetch thorn007
3)git diff master thorn007/master
4) git merge thorn/master
5) resolve conflict........
6) git push origin master
```

- if modification is small , you can use email+patch; If the modification is big, you can use fork pattern
- you can clone a exist project from others people or on other computers,
- Use email topic

```
1) git clone http://www.bitsun.com/git/gittutorcn.git
```

```
2) edit and commit
//method 1 (develop on master)
$ git  fetch origin
$ git rebase origin
$ git format path origin  ->0001-your-buddy-s-contribution.txt
//method 2 (develop on branche, better)
$ git checkout -b patch_mubs
$ git checkout master
$ git pull
...
$ git checkout patch_mubs
$ git rebase master ( why I need rebase here, I want to know answer)

3)email 0001-your-buddy-s-contribution.txt to vortune@gmail.com
for vortune:
1) git checkout -b buddy-in
2) git am /path/to/0001-your-buddy-s-contribution.txt
```



4) git checkout -b ct coreteam/master #建立一个coreteam分支
git branch -a # 查看所有分支 or gitk --all gitk # 里查看所有分支
git checkout -b newbranch # 建立newbranch分支
git checkout newbranch # 切换到newbranch分支
Edit.. Add...commit
git merge coreteam/master or git merge ct
Resolve conflict…..
git push origin master # 将主分支修改推送到服务器
git push origin newbranch # 将newbranch分支修改推送到服务器
5) in github, click pull request to inform the developer.

1) you can build a ct branch from coreteam/master
2) you need build newbranch to merge with remote
3) before you merge, you should fetch coreteam first to get latest version(maybe)
4) you need clone action first. You need follow the order.

2.5 IDE

If you can access GUI, you can use code::block and kile to develop and documents. if you login by SSH, By now, I think that best tool is Emacs, with GUD, it can debug a program. Does Emacs support latex well? Yes, It has tex mode.

Windows has VS community version. Mac has Xcode, but I didn't try it too much.

In linux, There are many tools you can uses. The most advance tool is Eclipse. the medium tools are codelite and UltraGDB. And the simple tool is vim and gdb(gdb -tui)

ddd is old tool, when I use it in mint 17, It's difficult to set font. It seems that a bug for ddd. and ddd is old tool which seems to stop developing.

Then I try affinic debugger, It's a commercial software and need serial number, I look for and there is not many answers on google, Maybe it's not very popular, and the shortcoming is white background and you cann't change color theme. so I give it up.

codelite is a good tool, Setting->colours and fonts->Customize->C++ , you can select Themes: Monokai_2, and you can set font MonoSpace 11pt. in codelite, you need to set terminal as gnome-terminal in Setting->preferences->terminal "/usr/bin/gnome-terminal -t '$(TITLE)' -e '$(CMD)'" It also use xterm as debugger output. In mint, it output ugly font. you need to produce .Xresources file on you home directory. I keep it on the linux software backup.

UltraGDB is subset of Eclipse, if you just need front-end of GDB, It's the best one. You can set dark theme in Window->Preferences->General->Appearance

In conclusion, front end of GDB are UltraGDB and codelite. IDE are eclipse and codelite.

2.5.1 Understand

Understand is a good tools to read a large scale software.

- Don't add .h, .inc, or .inl files into understand project, It will be included by .cpp and analyze automatically.
- For C++, use strict option in project configuration. You can adjust c++ version in strict C++ configuration. By now, It's good to use

C++11, C++14 is too new

- In Understand ,you can use "Improve project accuracy"->"missing include files" to help you search head file automatically.

- For CMake file, you can use **CMake -CMAKE_EXPORT_COMPILE_COMMANDS -G "Unix Makefiles"** It will produce a compile_commands.json file. In the end, you can use `$ und ↵` to produce a project.und. It will includes correct configuration information. Don't use -G "Xcode", It will not produce compile_commands.json file. Detail can be seen in my EverNote bookmark.

- For gcc and g++ project, you can use buildspy tool in Understand, detail can be seen in my EverNote bookmark.

- If you select "terminal" color theme, inactive code will not visiable, you can change the background color to make it visible through "Preferences"->"Editor"->"Styles"

- When analyze a long or difficult file, "worker process killed after 2 mins", you can try "Ananlyze changed file " again. if 2 minutes is too short, you can configure it longer on this specify file by override configuration.

- Macro is key factor in understand project, you can define it maually, but recommended way is to use CMAKE or buildsyp or visual studio to prodce a understand project automatcilly. it will includes all the correct Macro defination and include path.

- You can use buildspy to build a understand project in linux, configure project to relative path "Portability" then you can move source code and understand project to Mac, and use understand in Mac to open it. the interface on Mac looks better than Linux.

- By now, I have two projects, one is llvm and the other is openuh. For llvm, when you run CMake in test1 directory, It will produce some head file, when you compile use xcode or makefile, It will produce some def file. But I didn't run can compiling in test1 directory, but I have really compiled src in cmake_release_build directory. When i use add automaticlly search headfile, it add some files in cmake_release_build directory. In Understand Project Browers, you can see these three directories, in fact, test1 and cmake_release_build are overlapped.

- for LLVM and openuh, when you configure and compile the source code, It will produced some addtional files to be used in compiling in build directory. so, In the end you understand project will also include build directory in the end.

Chapter 3

Other Tools

3.1 OS and phone

3.1.1 useful tips

- In linux or mac, if you want to print c++ source with linenumber, you can use
  `enscript -MLetter --line-numbers -p - --word-wrap a.cpp | pstopdf -i -o ~/out.p`
  you can use `brew install enscript` to install enscript. Maybe you will see a error: you can't write /usr/local/etc. then use `sudo chown -R ‘whoami‘:admin /usr`
  give you permission. and then try brew again.

Common used applications in each OS. in phone, I didn't install many app by now. it make you phone battery die very quickly. and I always sit in front of computer.

	mac	windows	linux
diagramming	OmniGraff ConceptDraw	visio	Dia inkscape
vector drawing	illustrator	coreldraw illustrator	?
edit	textmate	Ultraedit	Emacs
doc	mactex	Ctex	livetext
web	dreamweaver	dreamweaver	?
screenshot	jing snapZ pro	snagit	?
screencast	screen flow camtasia	camtasia	Xvidcap
download	amule or Vuze	emule	amule
audio editor	audacity(amateur) logic(profession)	gold wave(amateur) adobe audition(professional)	?
video editor	finalcut imoive(amateur)	premiere(professional) HuiShengHuiYing (amateur)	?

3.1.2 Phone

- In my phone, weibo, webchat and web browser are three information sources.
- Evernote is main note tool. It can sync between phone and computer. When you have a lot note in Evernote, you can import them to latex document if you have time.
- In phone, you can use Everclip, on computer, you can use Evernote web clipper(in chrome browser)
- In Weibo or Webchat, you can repost or share it on moment. That is very easy way to keep knowledge. If this knowledge is just useful for you, you can use Everclip text, download image, or copy URL, Then use share function in your phone to save them to Evernote.

3.1.3 mac

- command is window key if you use a windows keyboard.
- command+ switches between documents. command+tab switches between applications.
- quicksiver can open and close any applications

- homebrew can be used to install a lot soft package from git to /usr/local/bin. It's very easy just brew install. if you have older version in /usr/bin, you need to export PATH=/usr/local/bin:$PATH. It's not very convenient. For this problem, you can down load module software, and use module load and unload, it is environment management .

  ```
  brew install modules
  ```

-
command +T	open a new brower
command+T, command +	switch application or windows
command+shift+4	screen snap
command+option+esc	force app quit(you need to command+tab switch)
command+h	hide current windows
command+Q or W	close app or windows.

3.1.4 win7

- win+arrow key can dock windows to left,right,maximize and minimize, that is very useful
- win+p can control project camera
- win+1,2,3 can launch programme in task bar quickly
- when the explore becomes slowly, you should check tool–manager add-ons to see which add-on is slow.
- right mouse key can produce a jump list, the content of jump list will change according to the type of progamme.
- move windows to the left side can change it to 50% width.
- win+L to lock the windows

3.1.5 iexplore and chrome

- learn to use tab to explore the internet. that is very useful, don't try to open a new page in a new windows, but in a new tab. $\boxed{\text{ctrl+num}}$ to navigate the tabs. and $\boxed{\text{ctrl+click}}$ to open a link in a new tab. $\boxed{\text{ctrl+T}}$ open a new tab. $\boxed{\text{ctrl+w}}$ to close the current tab.