Basic Knowledge

Yan Zhao

Contents

Chapter 1

Linux

1.1 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 used for desktop, and Mint is a 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.

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.

`$ sudo dmesg | more ↵` will show booting procedure messages. If you have some error in booting, you can use this message to check it.

1.2 virtual machine

You can install Linux in **Virtual Box** which is very good virtual machine. After you install it, you need to install guest addition, then restart the computer. If you want to try different version linux, a better way is to use docker, it's lightweight than virtual box.

You can also specify the share folder between host and guest. You need to run `$ sudo groupadd usbfs ↵` and `$ sudo groupadd vboxsf ↵`. Then use `$ cat /etc/group ↵` to check if you add successfully. Then `$ sudo adduser yan vboxusers, ↵ $ sudo adduser yan vboxsf ↵`. In this way, you can use shared folder between host and guest.

Another popular option is VMWare. It will support copy/paste and windows size will be changed properly. The newest version is vmware workstation pro 17.

```
vmware-toolbox-cmd -v #to see which VMware you are running. if not
sudo apt install open-vm-tools
sudo apt install open-vm-tools-desktop
```

VMware Workstation with Hyper-V Mode ("Windows Hypervisor Platform") Introduced due to Windows updates (especially from Windows 10 onward). Uses Microsoft's Hyper-V as the underlying hypervisor. VMware is no longer fully in control of the virtualization stack. Known for performance degradation and limited compatibility with some VMs and features. Needed when Hyper-V is enabled (e.g., for WSL2, Windows Sandbox, Device Guard).

2. VMware Native Mode ("VMware Virtualization Engine") Traditional VMware virtualization. Uses VMware's own hypervisor directly. Offers better performance, feature completeness, and stability. Works only when Hyper-V is disabled on the host.

In one word, if you want to use docker + wsl2 combination, it's good to make hyper-v, but vmware maybe doesn't work well. If you disable hyper-v, vmware works well, but docker+wsl2 has something wrong. That is all.

1.3 Shell

1.3.1 Shell type

There are two basic shelles, 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.

Before you install vmware, disable hyper-V and memory integrity. Detail can be googled.

1.3.2 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`␣'  # just print $PATH
echo "$PATH␣`pwd`␣\$PATH"  #expand $PATH and execute 'pwd'
```

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 even there is no a.h and a.c in the **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. Let me explain below: The first * is to avoid shell to interpret it. But in the new version linux, you don't need \ any more; The second one need to add double quote to avoid interpret, And the third one need to use shell to expand it.

```
yan.zhao@MB-XTPX4XQG9G test % find . -name *.c
./a.c

yan.zhao@MB-XTPX4XQG9G test % find . -name "*.c"
./a.c
./linux/a.c
./linux/b.c  #shell give "*.c" directly to find, find interpret this command inside it
    .
```

1.3.3 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.

Shell Type	Reads Config Files	How It's Started
Login shell	`/etc/profile`, `~/.profile`, etc.	SSH, console login, `bash -login`
Non-login shell	`~/.bashrc` only	Terminal app, scripts, GUI terminals

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 behavior. Such as expansion order**.

Three methods to use $

```
today=$(date) # preferred over legacy backticks `...`
echo ${today} #{} is better style
$$ $? $0  #Special shell variable
```

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. To fix history problem, always use exit command in terminal windows.

1.3.4 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, join, paste, expand, split, tr, awk, sed, less**

Commands that:

- Require interactive input (vi, nano, passwd)
- Don't read from stdin by default (e.g., mkdir, cd)

For example, `$ cat file | mkdir ↵` won't work — mkdir doesn't take input from stdin. 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. A easy way is to use `$ echo "Yan" | command ↵`. If it's work, then command support pipe.

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.

Command	Description	Example
grep	Filter lines by pattern	`cat file | grep "error"`
awk	Field/text processing	`ls -l | awk '{print $9}'`
sed	Stream editor (replace/edit text)	`cat file | sed 's/foo/bar/'`
cut	Extract specific columns	`ps aux | cut -c 1-20`
sort	Sort lines	`cat file | sort`
uniq	Remove duplicate lines	`sort file | uniq`
wc	Count lines, words, or characters	`cat file | wc -l`

Command	Description	Example
xargs	Convert stdin to command arguments	`find . -name "*.log" | xargs rm`
head	Show first n lines	`cat file | head -n 5`
tail	Show last n lines	`journalctl | tail`
tee	Output to terminal and a file	`cat file | tee out.txt`
tr	Translate or delete characters	`echo "abc" | tr a-z A-Z`

1.3.5 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` ↵, this command will add 'x' to all user, group and other.

`$ 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.

1.3.6 Terminal tips and key shortcut

You can use `$ xdg-open file ↵` to use default app to open the file. Such as `$ xdg-open a.pdf ↵`.

Motion shortcut keys are list below: Ctrl+p or up down arrow key can go to the previous command, this is very useful. Ctrl+a, k will delete very long command line. alt+f, b jump with word. Another useful is Ctrl+r.

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 key shortcut.**

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:

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

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/k` 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.

1.3.7 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.

In terminal, Ctrl+C is not for copying here. It is a control character: sends SIGINT to stop a running command (like killing ping, or interrupting a script). To copy in the terminal, use: Ctrl+Shift+C – Copy; and Ctrl+Shift+V – Paste

in moder ubuntu, we use wl_copy and wl_paste. -p will use primary(highlight text) -n will not include newline.

1.4 File and Dir

1.4.1 Basic

`$ du -h -max-depth=1 ↵` list current sub directory size.

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. `$ */ ↵` tells shell to expands to all the directorys, only * will expands all directorys and files.

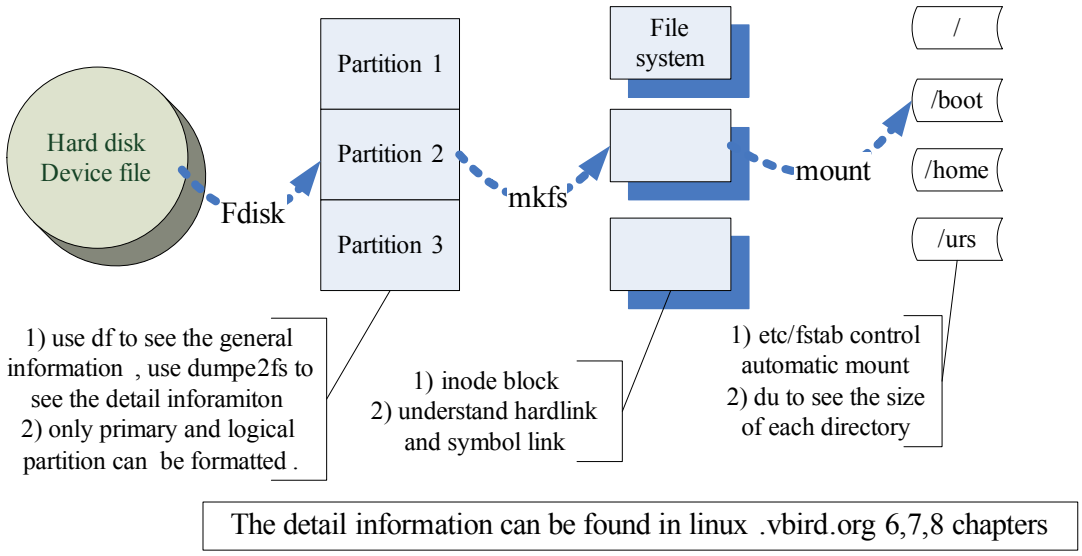
/usr stands for UNIX Software Resource, isn't users directory. It associate with software. /users includes all the users name, don't confuse them. FHS recommend linux developer install their application into the different dirs inside of /usr: such as /usr/bin and /usr/lib. Don't build their separately 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

1.4.2 partition and mount point



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 /.

1.4.3 Permission or file mode

`$ ls -al` ↵ will list all the files ownership and permission. Basically, all the Linux system support `$ ll` ↵ command directly, so you can use it 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 owner's 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!

1.4.4 Commands about File

du means disk usage. `$ 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. `$ cd` ↵ will return the home directory.

`$ 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.

1.4.5 Find command

`$ find` ↵ basic is to follow a path directly after find. A common usage is `$ find .` ↵

`$ find . -name *.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. **The key points is: if you don't add double quotes, the shell will interpret *.txt first. If you add double quotes, shell will not interpret(expand) it and give it to find command directly.**

In the previous example, you also can use `$ find . \*.txt` ↵. Use \ to stop shell to expand it.

`$ find . -name "ab"` ↵ will not find "abc" file. You need to use "ab*" to get it. A better expression is "*ab*" will find more files.

`$ 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.

find can follow two directorys name, and it will recursive search sub directory automatically. `$ find /home/user/docs /home/user/projects -type f -name "*.txt"` ↵

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 ".."
```

1.4.6 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

1.5 net

The file /etc/resolv.conf in Linux is used to configure DNS (Domain Name System) resolution — that is, how your system translates domain names like example.com into IP addresses.

ping 8.8.8.8 is fast, but ping google.com is slow, then DNS server has problem.

The file /etc/wpa_supplicant/wpa_supplicant.conf is the main configuration file for wpa_supplicant, the tool responsible for managing Wi-Fi connections on many Linux systems, especially in headless or minimal environments (e.g. Raspberry Pi, embedded systems, or servers without NetworkManager).

sudo systemctl status wpa_supplicant.service sudo systemctl enable wpa_supplicant.service

This command is used to enable the wpa_supplicant service to start automatically at boot time. What it does: When you enable a service, it creates symbolic links in the system's startup directories, so the service is started automatically when your system boots.

`$ sudo ip route del default via 100.87.195.237 ↵`

sudo rfkill enable wifi sudo rfkill unblock wifi rfkill list

The rfkill command in Linux is used to enable or disable wireless devices (such as Wi-Fi, Bluetooth, etc.) on your system. It's typically used to control the hardware state of network interfaces without having to interact with the system's GUI or network management tools.

`$ sudo ip route show ↵` show what route you are using to connect.

```
default via 100.116.93.65 dev wwan0 proto dhcp src 100.116.93.64 metric 204
default via 192.168.0.1 dev wlan0 proto dhcp src 192.168.0.182 metric 303
10.0.40.0/23 dev eth0 proto dhcp scope link src 10.0.40.33 metric 202
100.116.93.0/25 dev wwan0 proto dhcp scope link src 100.116.93.64 metric 204
192.168.0.0/24 dev wlan0 proto dhcp scope link src 192.168.0.182 metric 303
```

The file /etc/dhcpcd.conf is the configuration file for the dhcpcd (DHCP Client Daemon) in Linux. This daemon is responsible for obtaining network configuration (like IP address, DNS, gateway, etc.) from a DHCP server on the network.

By editing this file, you can configure network interfaces, set static IP addresses, and manage other DHCP-related settings.

1.6 service

Open a terminal and create a new service file under /etc/systemd/system/, for example: sudo vim /etc/systemd/system/myapp.service

```
[Unit]
Description=My Custom App Service
After=network.target

[Service]
Type=simple
ExecStart=/path/to/your/app --optional-args
Restart=on-failure
User=youruser
WorkingDirectory=/path/to/your/app/directory

```

```
[Install]
WantedBy=multi-user.target
```

systemd is a modern init system and service manager for Linux, designed to replace traditional tools like: SysVinit (classic init scripts), init.d.

/lib/systemd/system/ Default unit files provided by packages (like cron, ssh, nginx, etc.) Installed automatically when you install software

/etc/systemd/system/ Used for: Custom unit files (you create). Overrides of default service files. Safer to edit — these take priority over /lib/systemd/system/

1.6.1 log

`$ cat /var/log/syslog ↵` and `$ sudo journalctl -r ↵` are two major log check commands.

1. Your app sends log messages to the local syslog service. These messages end up in /var/log/syslog or the systemd journal.

```c
#include <syslog.h>
int main() {
  openlog("myapp", LOG_PID|LOG_CONS, LOG_USER);
  syslog(LOG_INFO, "This is a syslog message from myapp");
  closelog();
  return 0;
}
```

2. Using stdout/stderr (recommended with systemd) Write logs to standard output/error. If your app runs as a systemd service, systemd-journald automatically captures these. You can then view logs with journalctl -u yourapp.service.

1.7 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.

1.8 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.

1.9 Application

BusyBox is a single binary that includes a wide range of essential Linux utilities, many of which are commonly used in embedded systems and environments with limited resources. It is often called "The Swiss Army Knife of Embedded Linux" because it provides a lightweight, compact, and highly customizable way to run a full range of Unix commands, all bundled into a single binary file.

1.9.1 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.

1.9.2 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.

1.9.3 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.

1.9.4 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.

1.9.5 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

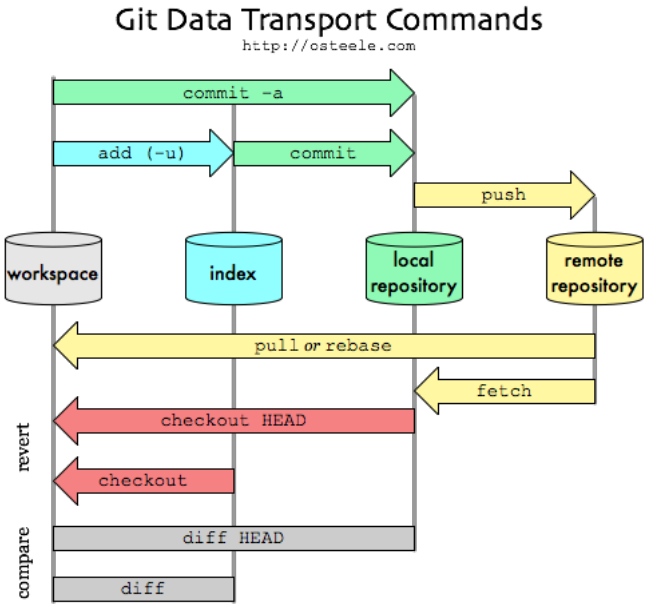
xmodmap -e "add Contrlo = Menu" (this statement is very important)
xmodmap -e "keycode 133 = Control_R"

Chapter 2

git

2.0.1 Basic introduction

main usages



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.

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: remote, repository, branch, commit, paths and files. You need to know object of a command, such as merge command, it needs two branches, not two

commits. Checkout branch will switch to its branch, and checkout commit will cause detached-head, etc.

```
git remote add bhanu git@github.com:aleti-bhanu-infovision/vizio_sdk.git
//bhanu is remote,  vizio_sdk is repository,

git fetch bhanu
// fetch should follow remote,  default remote is origin.

git checkout DEV-13932_6.0
```

origin is a remote 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 HEAD^ 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.

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. because * has been interpreted by shell

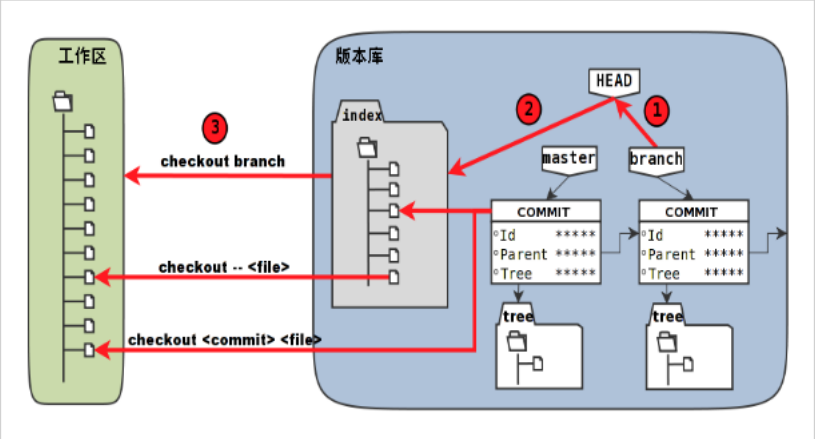
```
git add Documentation/\*.txt  //this will add certain pattern recursive.
git ls-files -co --exclude-standard | grep '\.java$' | xargs git add  //another
    way to add pattern recrusively
```

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 git checkout command

1. If there is path? if yes, it will not change HEAD, just overwrite index or workspace
2. if there is no path? to see if there is branch, if it's not branch, but a commit, it will cause detached head and it's not good practice.

```
git checkout [commit] -- path
//overwrite workspace, no commit means from index
//with commit, it also overwrite index

git checkout branch
git checkout -b branch.  #build a new branch
```



reset command

1. reset follow path, it will not change reference(HEAD and branch), just overwrite index. With new git restore command, this usage become obsolete.

```
reset HEAD -- a.cpp
//just oppsite operation git add a.cpp
```

2. if no path. it will follow a commit, pay more attention to –soft, –mixed and –hard.

both reset and commit has key point. **See if they follow path?** , if follow path, the main purpose is overwrite. Without path, **checkout commit 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 cat-file -t e695606
commit
git cat-file -t f58d
tree

//tree object
git cat-file -p f58da9a
100644 blob fd3c069c1de4f4bc9b15940f490aeb48852f3c42    welcome.txt
```

diff main usages.

```
git diff              //work<->index
git diff HEAD     //work<->HEAD
git diff --cached    //index<->HEAD
```

Three useful commands

```
git reflog   <refname>@{<n>}  //HEAD@{1} , this is used by time travel.
git rev-parse
git rev-list
```

```
git checkout vo-1255  //checkout remote branch
git checkout -b vo-1255-yan --track origin/vo-1255  //change branch name.
```

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.

git configure

`$ git config -e [-global|system] ↵` -e will open a editor. –global is for current user, –system is for current computer /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

git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.ci commit
git config --global alias.st status

git config --global alias.oneline 'log --pretty=oneline'

git config --list #list all the configuration command options
```

In windows, .gitconfig will be saved in C:\Documents and Settings\Administrator directory. linuxcommandgit config –global core.editor notepad (use notepad as default editor)

in Mac, install p4merge with

```
brew install --cask p4v     //use p4v as new name here.
```

then, you need to add below configuration in the global level,

```
[user]
name = Yan Zhao
email = yan.zhao@vizio.com
[merge]
keepBackup = false
tool = p4merge
[mergetool "p4merge"]
cmd = /Applications/p4merge.app/Contents/Resources/launchp4merge "$BASE" "$REMOTE" "
    $LOCAL" "$MERGED"
keepTemporaries = false
trustExitCode = false
keepBackup = false

[diff]
tool = p4merge
[difftool "p4merge"]
cmd = /Applications/p4merge.app/Contents/Resources/launchp4merge "$REMOTE" "$LOCAL"

[color]
status = auto
```

```
branch = auto
ui = auto
```

GitHub or gitlab

Don't use any Chinese in version control, If your English is not good, use Pinyin.

Usually, You need to work and know ssh related knowledge. A basic introduction is in a webpage "Set up personal SSH keys on Linux _ Bitbucket Cloud _ Atlassian Support" in the ref directory. `$ ssh-keygen -t rsa -C yan.zhao.74@gmail.com ↵`

For gitHub, I have two github account:

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

In windows, your public key has been saved in /c/Users/zhao/.ssh/id_rsa.pub(for windows) and /.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). If you copy .ssh from windows to linux, you should chmod g-r(wx) id_rsa to make it private. Then you should run `$ ssh-add ↵` command. Once you finish it, `$ ssh -T git@github.com ↵` will test you key setting. (github has details explanation.)

A better idea is just use ssh-keygen on different compouter, don't copy private key unless you have strong reason.

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.

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.

When there are two accounts in github, you need to generate two private key with two different email address, you can specify different key file names when you are use **ssh-keygen**. One tip is to modify the second Host, with your account name add after github.com

```
Host github.com
  HostName github.com
  User git
  IdentityFile ~/.ssh/id_ed25519
```

```
5.
6. Host github.com-zhaoyan
7.   # with account name here
8.   HostName github.com
9.   User git
10.  IdentityFile ~/.ssh/id_ed25519_person
11.  # add this public key to second account
```

Once you have this config, restart a new terminal. when you want to clone, use this command

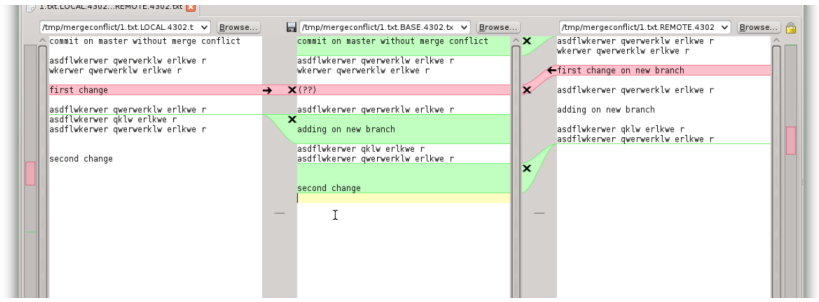
```
1. git clone git@github.com-zhaoyan:zhaoyan/new_doc.git
2. ssh -T git@github.com-zhaoyan
3. #test one account
4. ssh -T git@github.com    #test another account
```

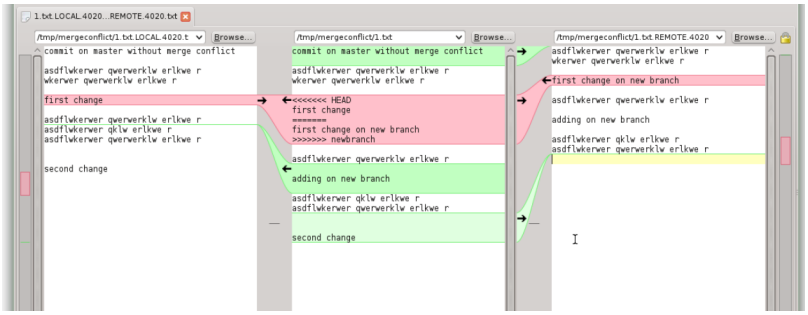
GUI

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. $BASE is the local and remote the common ancester. $MERGED is the content like below:






```
1. [diff]
2. tool = meld
3. [difftool]
4. prompt = false
5. [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"
```

Above is for linux, below is for Windows. In window, you also can use meld. but you need different config file. (I only test diff, but didn't test merge)

```
[diff]
  tool = meld
[difftool]
  prompt = false
[difftool "meld"]
  cmd = \"C:/Program Files (x86)/Meld/Meld.exe\" $LOCAL $REMOTE

[merge]
  tool = meld

[mergetool "meld"]
  keepBackup = false
  trustExitCode = false
  prompt = false

[mergetool "meld"]
  path = C:\\Program Files (x86)\\Meld\\Meld.exe
# cmd = meld "$LOCAL" "$MERGED" "$REMOTE" --output "$MERGED"
  cmd = meld "$LOCAL" "$BASE" "$REMOTE" --output "$MERGED"
```

1. put them in the `new\_doc\.git\config file`. **Don't put it in .gitignore file.**
2. In git bash, if you run meld –version has no output, Locate the Meld installation directory. Open the Windows Start Menu and search for "environment variables." Click on "Edit the system environment variables." Click the "Environment Variables" button. In the "System variables" section, find the "Path" variable and select it. Click "Edit" (or "New" in some Windows versions). Add a new entry with the path to the Meld directory Click "OK" on all windows to save the changes.
3. If there is still a bug, you need to try below:

```
A temporary workaround is to copy "C:\Program Files
(x86)\Meld\lib\libgirepository-1.0-1.dll" to "C:\Program Files (x86)\Meld\
    libgirepository-1.0-1.dll". Copy it up one directory.
```

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.

Scenario	After resolving conflicts	Then do this
`git merge`	`git add <file>`	`git commit` or `git merge -continue`
`git rebase`	`git add <file>`	`git rebase -continue`
`git cherry-pick`	`git add <file>`	`git cherry-pick -continue`

The basic usage is: git diff is terminal interface. It's used by remote log or no gui tool(other people's computer). If you want to use GUI app, you need to explicit callwith git difftool

when git merge trigger conflict, it will overwrite the original file with all conflicted contents in it. such as

```
<<<<<<< HEAD
#define _GNU_SOURCE main
=======
#define _GNU_SOURCE BBB
>>>>>>> bt
```

When you run git merge –abort, it will return back to original non conflict status.

Then git merge will exit with failure. Next, you can use git mergetool. After you exit the mergetool, The file has been saved and confict format disappear, and the file has been added already, so you can run git merge –continue or git commit directly. That is the basic procedure. git merge –continue –no-edit to avoid a annoying editor pop up. git merge –continue is better, it produce a commit with "merge bt" message automaticlly.

2.0.2 Basic commands

status log show

`$ git status -uno ↵` **will not show untracked 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 included.
```

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

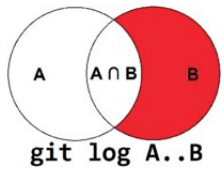
`$ git status -s ↵` show two columns , the first is staging, the second is working tree. if you modify a file, then add. then you modify a file again. Now git status -s show MM a. guess what it means?

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

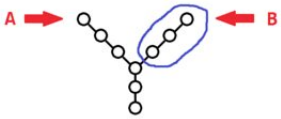
```
git log —pretty=oneline #just show simply log information
git log –p   #show ci log and source modification each commit.
git log –2  #just last two commits
git log —after 2015–12–01  #show all commits after date
git log —oneline  #simple informaitons
git log —abbrev–commit —pretty=oneline #simple 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 log —author=Bob # only show Bob's commit.____
```

Here is a visual representation of `git log A..B`. The commits that branch B contains that don't exist in A is what is returned by the commit range, and is highlighted in red in the Venn diagram, and circled in blue in the commit tree:

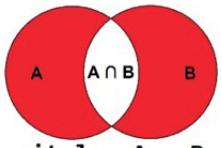


git log A..B

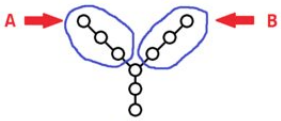


git log A..B

These are the diagrams for `git log A...B`. Notice that the commits that are ***shared*** by both branches are not returned by the command:



git log A...B



git log A...B

`$ 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 and commit

When you add, You should know below procedure.

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, we "snapshot" this tree and give this tree a sha value. give a tree sha value, and copy whole tree out, add tree object to commit object, and give commit object a sha value.

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 if you have committed it. 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. This is especially useful if you:

- Added a file to Git accidentally (like a .env or a large file).
- Want to remove a tracked file and add it to .gitignore.

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 -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.

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 1` ↵, 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 and second operation will be different. Once you have shared your local branch, don't use these two commands. So don't change commit which you pull from public or you have pushed to public.

remove rename

Basic command of remove and rename. **You have to use git rm and git mv, don't forget git in the front of command**.

```
1. git rm a // It will delete from work space and index
2. git rm --cached a  //just delete a from index.
3. git commit -m "delete file a"
4.
5. git mv a b  #equal three command: mv a b; git rm a ; git add b ;
6. git commit -m "rename a to b"
```

diff

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

```
1. git diff      ##(1) and (2)
2. git diff --cached   ##(2) and (3)
3. git diff HEAD   ##(1) and (3)
4.
5. git diff tag      ##tag and HEAD
6. git diff tag file    ##just compare a file (only one file name)
7. git diff tag1..tag2   ##two tags( you can omit two dots)
8. git diff SHA11..SHA12  ## two commits
9. git diff tag1 tag2 file or  git diff tag1:file tag2:file
10.
11.#tag can be a alias of remote
12. git remote add xjsff git://github.com/xjsff/hello-world.git
13. 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.

checkout reset

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.

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. git checkout -b new-branch-name to build a new branch, then commit it first. Then checkout master. merge whith new-branch-name. 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.

If HEAD->master, reset will move HEAD->master together, if HEAD->commit(not branch), reset will only move HEAD. at this time, It's still detached head status.

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)
```

The default mode for git reset is –mixed.

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.

tag

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

No space in tag name.

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.

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.
    coding...., commit, last git stash pop.
git stash push –m "you␣messaage" #
git stash list #list all stash
```

```
git stash pop stash@{1}  //or
git stash apply stash@{1}
```

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

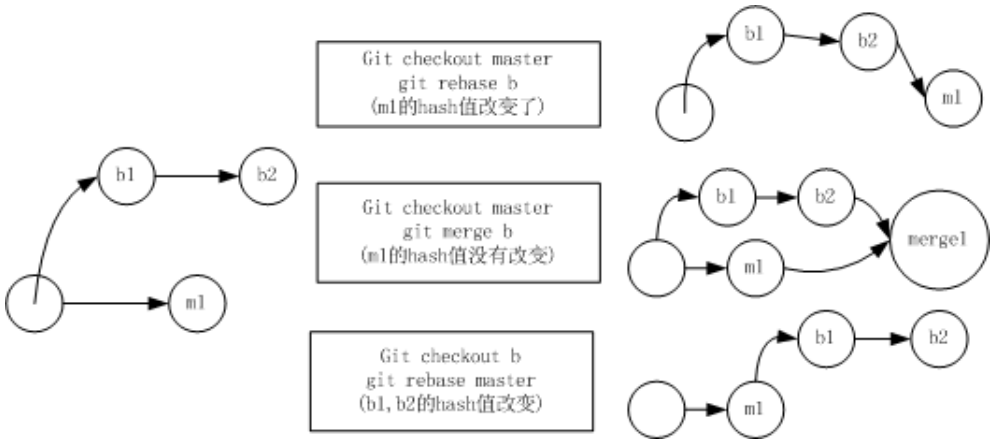
Never rebase branches or trees that you pulled. Only rebase local branches. Never ever rebase a branch that you have pushed, or that you pulled from another person

rebase command steps: git rebase master

1. 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.

rebase command and merge command is little different.

- You can think merge and rebase are two commands operate on branch.
- 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



rebase has –onto options. Detail can be found in git rebase document. Just google it.

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.

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
```

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.
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:zhaoyan/test.git ↵`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 name, here experimental-by-yan is remote-tracking-branches name, erperimentallocal name. <source-name>:<destination-name>

2.0.3 History

history representation

About two dots and three dots difference, the best explanation is here:

```
https://stackoverflow.com/questions/462974/what-are-the-differences-between-double-dot-and-triple-dot-in-git-com
```

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confict,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。

Change current latest commit. `$ git commit -amend ↵` not only change the message, but also the working tree contents.

revoke latest commit.

```
1. git commit –m "Something terribly misguided"              (1)
2. git reset HEAD~                                           (2)
3. << edit files as necessary >>                             (3)
4. git add ...                                               (4)
5. 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**

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

Delete all old commits before A

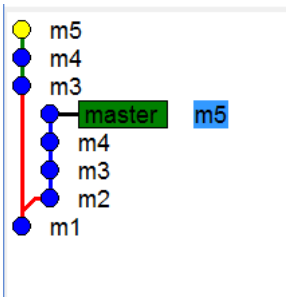
1. `$ echo "Commit from tree of tag A" | git commit-tree A^{tree} ↵` It will produce a <SHA value>. This command will produce a new commit with <SHA value> and without any parent. It's a root commit.
2. `$ git rebase -onto <SHA value> A master ↵` All the history behind of A will be deleted.

Delete some old commits by cherry-pick

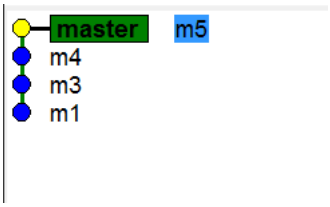
1. `$ git checkout C ↵`, C is a tag, by now, working tree is C status, Here, you can't use `$ git reset C ↵`, because it will put master to C, then D, E and F will not be accessed.
2. `$ git cherry-pick E and F ↵` , delete D commit ,cherry pick will produce a new commit SHA value, such as E* and F*.
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. In fact, rebase is just using cherry-pick one by one automaticlly, so this method perform the same steps with the previous one.

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.



   `$ 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.
2. `$ 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

- 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

- 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

Who made some modification.

```
1. git blame -L 12,13 hello.html
```

2.0.4 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

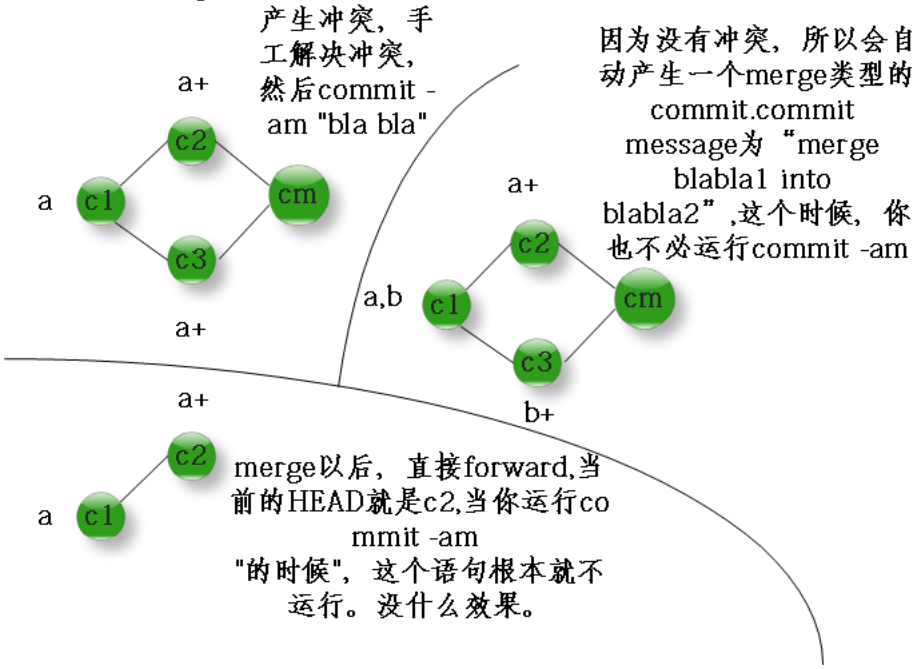
checkout -b can be thought as short hand of two commands 1) branch create a new branch, and 2) checkout to switch it.

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 possible merge result:

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. After you resolve conflict, run git add file-name and git commit.
4. You can see figure below to understand it better.



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
```

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

If you don't config, when local branch has divergent, you will see below error message, just select one pull.ff is the safe, you should run git fetch first, then use merge or rebase accordingly.

```
1. hint: You have divergent branches and need to specify how to reconcile them.
2. hint: You can do so by running one of the following commands sometime before
3. hint: your next pull:
4. hint:
5. hint:   git config pull.rebase false  # merge
6. hint:   git config pull.rebase true   # rebase
7. hint:   git config pull.ff only       # fast-forward only
```

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.

2.0.5 conflict

In git, there are four commands will cause conflict, such as **revert, merge, cherry-pick,rebase**. If you see document, you will find all comand are with –continue, –abort and –skip 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.

2.0.6 Common used commands

Run `$ git gc ↵` every month to optimize.

Use below command to know which branch you are in.

```
cat .git/HEAD
// show something like this ref: refs/heads/TVPF-34751-6.0-34
//This command is much better than git branch

git rev-parse HEAD master  //check each reference sha value.
e695606fc5e31b2ff9038a48a3d363f4c21a3d86
4902dc375672fbf52a226e0354100b75d4fe31e3
```

```
$ git merge --no-commit --no-ff $BRANCH

//To examine the staged changes:
$ git diff --cached

//And you can undo the merge, even if it is a fast-forward merge:
$ git merge --abort
```

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. `$ git remote -v ↵` to see what origin is.
3. `$ 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 have to use "origin" here, if you don't specify branch, default is master. master(main) is default local branch, you don't need to create it explicitly. this command push local content to the server.

move to new computer

copy private key from my desktop to new computer, if it's windows, it is C:/User/yzhao/.ssh change name to id_rsa.git in git bash.(This method is not good enough, It's better to generate private key and add public key to the github website.)

```
eval "$(ssh-agent -s)"

cd ~
run ssh -T git@github.com
ssh-add .ssh/id\_rsa.git
```

```
Host GitHub.com
  Hostname github.com
  User git
  IdentityFile ~/.ssh/id_rsa.git
  IdentitiesOnly yes # see NOTES below
  AddKeysToAgent yes
```

Update a project from github

1. $ git clone git@github.com:zhaoyan/hello-world.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 ↵

Company

```
  1) Most of time, just fetch and pull offical production_branch to the newest state.
  2) git checkout -b jira_banch and work on it.
  3) before push, do  git diff HEAD..origin/production_brach to see if there is update
     remote,
  4) if yes, pull production_branch, and git rebase production_branch.
  5) git push -u origin jira_branch
  6) go to github to create PR and ask for review.
```

```
  1) git checkout release-FUR6.0 // it will fetch from remote automatically
  //if there are multiple remote repositories configured then use below command
  //git checkout -b test <name of remote>/test

```

```
2) git fetch and git pull
// this command will pull new content. The first time you checkout, you don't need to run it. but after a few days, you need to run it to keep track the remote.

3) git checkout -b TVPF-27720 //where to find 27720? in your work, see a bug jira, with TVPF. it TVPF is based on release-FUR6.0 now.

4) git cherry-pick 283bc63 //where to find commit number? This is commit where merged to master, it's not you local commit.
//maybe we should not use cherry-pick. it will introduce master content into the FUR6.0. A better way is to use compare file, just add you logic into release-FUR manually.
//This time we use cherry-pick, if it's conflict, you should be alart, it's better --skip and add logic manually to TVPF branch.

//conflict,
edit the conflict file, it will have >>>>> <<<<<, delete or merge one part, and save
git add -u
git cherry-pick --continue
//exit the editor, default use vim. so use :wp
git push --set-upstream origin TVPF-27720 //I think that it's just like -u
//then go to the git website. create pr with name TVPF-27720 Port to 6.0
```

Another way to phrase the question is "What is the nearest commit that resides on a branch other than the current branch, and which branch is that?" This command is very useful for complicated branch management system.

```
//add below to ~/.gitconfig

[alias]
parent = "!git␣show-branch␣|␣grep␣'*'␣|␣grep␣-v␣\"$(git␣rev-parse␣--abbrev-ref␣HEAD)\"␣|␣head␣-n1␣|␣sed␣'s/.*\\[\\(.*\\)\\].*/\\1/'␣|␣sed␣'s/[\\^~].*//'␣#"
```

find which branch is a commit in?

```
git branch -a --contains <commit>
git reflog show --all | grep a871742
```

How to use patch?

```
git diff >jira.patch
git apply --whitespace=warn patchname.patch
```

how to build local branch

```
git checkout -b <branch>
Edit files, add and commit. Then push with the -u (short for --set-upstream) option:
git push -u origin <branch>
```

what does push -u mean?

git branch –set-upstream-to <remote-branch> sets the default remote branch for the current local branch. Any future git pull command (with the current local branch checked-out), will attempt to bring in commits from the <remote-branch> into the current local branch. One way to avoid having to explicitly type –set-upstream / –set-upstream-to is to use its shorthand flag -u as follows: git push -u origin local-branch This sets the upstream association for any future push/pull attempts automatically.

git log –raw to check how many files you have change.

How to check all changes in on branch

```
1. git merge-base hf-timeshare-protocol master
2. <show number>
3. git difftool HEAD..<showNumber> (don't forget the two dots.)
```

list all you local branch according to date.

```
1. git branch --sort=-committerdate
```

See one file change history. This file has been changed name, so use dash M

```
1. git log -M --follow -- Common/Microcontroller/SharedModules/SharedClasses/
       TwoWayValve.h
2.
3. git difftool HEAD f96b2a -M -- Common/Microcontroller/SharedModules/SharedClasses/
       TwoWayValve.h Common/Microcontroller/SharedModules/PT0126PumpAndValves/
       TwoWayValve.h
4.
5.
6. git difftool f96b2a..f96b2a^ -- Common/Microcontroller/SharedModules/
       PT0172APumpValve/TwoWayValve.h
```

list all files change in one commit

```
1. git show --pretty="" --name-only bd61ad98
```

Chapter 3

Developing tool

You need to know basic keyboard short cut, detail can be found in mac section below. There are two editor. one is vs code, the other is vim. vs code keyboard can be found in IDE section. There are three tools: git, terminal and double command. For terminal, common keyboard shortcut is ctrl+A, E, R, U. for double command, keyboard shortcut can be found in Double commander section.

3.1 Drawio

You can download and install it directly on the windows, don't need to access the website.

draw connection automatically by using the blue arrow around the object. That is very important tips.

1. Hover over an existing shape on the drawing canvas and four blue directional arrows appear.
2. Click on one of these arrows - the first item in the selection box that pops up will clone the shape and automatically draw a connector between them.

If you want to draw curved line, build connection first, then change line style to curve, it will be easier.

Ctrl + D: Duplicate selected objects A: Add text s: Create note D: Insert rectangle F: Insert ellipse C: Create line

3.2 tmux

A key idea about tumux is session and windows can be attached and detached.

tmux start, then type exit will finish the tmux. Tmux has a large number of shortcuts. All shortcuts need to be triggered through a prefix key. The default prefix key is Ctrl+b, which means you press Ctrl+b first, and then the shortcut will take effect.

For example, the help command's shortcut is Ctrl+b ?. The usage is as follows: in a Tmux window, press Ctrl+b, then press ?, and the help information will be displayed. Then, you can exit the help by pressing the ESC key or the q key.

```
tmux new -s <session-name>
c+b d #detach
tmux ls #list
tmux attach -t 0
tmux kill-session -t 0
```

ctrl+b % split left and right, ctrl+b left arrow and right arrow. That is the most often use pattern. panel is more useful than windows in tmux.

3.3 Jenkins

3.3.1 install on windows

When you install Jenkins on windows, you need to setup a new user. "run service as local or domain user", At this time, you can input your username, if it doesn't work, create a new user in windows without any email or phone. (windows supports this options). then follow this step.

When installing a service to run under a domain user account, the account must have the right to logon as a service. This logon permission applies strictly to the local computer and must be granted in the Local Security Policy.

Perform the following to edit the Local Security Policy of the computer you want to define the 'logon as a service' permission:

1. Logon to the computer with administrative privileges.
2. Open the Administrative Tools and open the Local Security Policy
3. If the Local Security Policy is missing in your system, refer to the answer in the Where to download GPEdit.msc for Windows 10 Home? question on Microsoft Community to troubleshoot
4. Expand Local Policy [Note: it's ... Policies on Win Server] and click on User Rights Assignment
5. In the right pane, right-click Log on as a service and select properties.
6. Click on the Add User or Group... button to add the new user.
7. In the Select Users or Groups dialogue, find the user you wish to enter and click OK
8. Click OK in the Log on as a service Properties to save changes.

Then try again with the added user.

3.3.2 install on linux

You can see the install manual in Jenkins website

About use ssh in pipeline:

1. Install publish over ssh, ssh agent plugin and ssh pipelines step, three plugins.
2. On client, generate id_rsa pair. On server, cat id_rsa.pub(client) » authorized_keys.
3. create credentials, use ssh username with private key. ssh user name is just name, without ip address. it will generate a credentials with ID, you can use this ID in pipeline script below. The pipeline script include port and IP address.

About use email in pipeline: in Extended E-mail Notification

SMTP server: smtp.office365.com SMTP port: 587 Credentials: Username with password. user@mail.com password. Use TLS(check) No check OAuth 2.0(maybe it's depends on what server, just try different optoins, **In debug output, if you see SMTP: AUTH XOAUTH2 failed, disable OAuth 2.0, just use TLS**) Default user e-mail usffix: @mail.com(without username) Enable Debug Mode to see output detail

In E-mail Notification Use SMTP Authentication User Name user@mail.com password UseTLS SMTP port:587 Reply-to: user@mail.com Charset: UTF-8

Test configuration by sending test e-mail. just input user@gmail.com to see if you can receive it successfully.

About github or bitbucket

config

in multibranch configuration, in the Behaviors, add **First Build Changelog**, then the first build will also generate new changes contents.

3.3.3 config

install docker pipeline plugin. add jenkins to docker group sudo usermod -a -G docker jenkins, then restart jenkins.

inside docker, whoami is not jenkins, it's a little strange. + whoami whoami: cannot find name for user ID 134

git Security–>Git Host Key verification configuration->Host Key verification Strategy: change it to Accept first connection. when connect git, username is not matter.

ssh install publish over ssh, ssh agent and ssh pipeline steps, after that, restart your jenknin.

when connect with board, 1) build a credential with private key and password for board, 2) assign SSH_CREDENTIALS_ID with id 3) add public key to the board authorized_keys (echo ctrl+v »authorized_keys)

email System Admin e-mail address, otherwise, it shows " Domain of sender address nobody@nowhere does not exist". once you see nobody@nowhere, you need to config this email.

3.3.4 programming

A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.

There are two different syntax: Declarative Pipeline and Scripted Pipeline.

- All valid Declarative Pipelines must be enclosed within a pipeline block, for example:
- Like Declarative Pipeline, is built on top of the underlying Pipeline sub-system. Unlike Declarative, Scripted Pipeline is effectively a general-purpose DSL [1] built with Groovy. Most functionality provided by the Groovy language is made available to users of Scripted Pipeline, which means it can be a very expressive and flexible tool with which one can author continuous delivery pipelines.

sudo usermod -a -G docker jenkins add jenkins to docker group, so jenkins can use docker.

In a Jenkins Pipeline script using Groovy:

```
1. def email = "user@example.com"
2. echo "${email}"
```

Why the $ is needed:

The syntax `"${email}"` is a Groovy GString. It interpolates the variable `email` into the string. This tells Groovy to substitute the value of `email` inside the string.

Alternatively, you can just use:

```
1. echo email
```

But GStrings like `"Sending to $email"` allow you to build more flexible messages.

Context: Inside a Shell Script Block (`sh`)

If you're executing a shell command in Jenkins:

```
sh 'echo "${email}"'
```

This line is passed to a shell (like Bash), where `$email` is a shell variable. Bash needs the `$` to perform variable substitution.

Important: If `email` is defined in Groovy, it won't automatically exist in the shell. You must pass it explicitly:

```
def email = "user@example.com"
sh "email=${email}; echo \"Email is \$email\""
```

Note that:

- `email=$email` assigns the Groovy variable to a shell variable.
- `\$email` ensures Groovy doesn't try to expand `$email` before the shell sees it. (generate by AI, maybe it's not correct.)

Context	Needs $?	Reason
Groovy string (GString)	Yes	To interpolate the variable
Groovy plain variable	No	Can just pass variable directly
Shell script	Yes	Bash requires $ for expansion
Jenkins `echo` step	Yes (if in string)	For dynamic string construction

In a declarative pipeline, you can't use raw Groovy code freely. You must wrap it inside script : script runs on an allocated node already (so you don't nest node inside it). Used for logic, control flow, and dynamic variable creation.

```
pipeline {
  agent any
  stages {
    stage('Groovy Logic') {
      steps {
        script {
          def x = 5
          if (x > 3) {
            echo "x is greater than 3"
          }
        }
      }
    }
  }
}
```

In a declarative pipeline, the node is automatically managed by agent directive.

If you define agent any, Jenkins handles node under the hood.

Don't nest node inside script — that's almost always a mistake in declarative syntax.

A product pipeline script now: multibranch pipeline, get new commit, login the hardware, run compile. then login another server(vm), run compile in a docker, collect all the result, and return the result back to the developer with html format.

```
1.
2.  def skipRemainingStages = false
3.  def email = ""
4.  pipeline {
5.    agent any
6.
7.    environment {
8.      CI = 'true'
9.      SSH_CREDENTIALS_ID = '12531146-8669-40c0-8a67-b6d626bd84dd' //Don't change this
          value.
10.     SSH_REMOTE_PORT = 22
11.     SSH_REMOTE_HOST = '10.0.40.33'  //this is the product1 server IP address.
12.
13.     SSH_CREDENTIALS_ID_U24 = 'f31e9335-a726-438b-aff9-27b891a5a532' //Don't change
          this value.
14.     SSH_REMOTE_PORT_U24 = 22
15.     SSH_REMOTE_HOST_U24 = '192.168.65.128'  //this is the product1 server IP address.
16.     RECIPIENTS = ''
17.   }
18.   stages {
19.
20.     stage('Check for Changes') {
21.       steps {
22.         script {
23.
24.           bat 'if exist product1_build.log del product1_build.log'
25.           bat 'if exist product2_build.log del product2_build.log'
26.
27.           // Check if there are any change sets
28.           if (currentBuild.changeSets.isEmpty()) {
29.             echo "No changes detected, exiting pipeline."
30.             // Exit the pipeline without failing the build
31.             skipRemainingStages = true
32.             return
33.           }
34.         }
35.       }
36.     }
37.
38.     stage('Build product1') {
39.       when {
40.         expression {
41.           !skipRemainingStages
42.         }
43.       }
44.       steps {
45.         echo 'Build product1 ..... '
46.         script{
47.           bat '''
48.  ----------(
49.  ----------echo git stash
50.  ----------echo git fetch
51.  ----------) > checkout.sh
52.  ----------'''
53.         }
54.         bat ' echo git checkout %BRANCH_NAME% >> checkout.sh '
```

```
55.         bat '␣echo␣git␣pull␣>>␣checkout.sh␣' // if pull fail, we will not continue
                next compilation procedure.
56.
57.         script{
58.
59.           for (changeLogSet in currentBuild.changeSets) {
60.             // For each change set, iterate over the entries and get author name
61.             for (entry in changeLogSet.getItems()) {
62.               email = entry.authorEmail
63.               break  // Stop once the author name is found
64.             }
65.             if (email != "") {
66.               break  // Stop the outer loop once the author name is found
67.             }
68.           }
69.
70.           echo "Author␣Email␣${email}"
71.         }
72.
73.         script{
74.           withCredentials([sshUserPrivateKey(
75.           credentialsId: "${SSH_CREDENTIALS_ID}",
76.           keyFileVariable: 'identity',
77.           passphraseVariable: '',
78.           usernameVariable: 'pi'
79.           )]){
80.             def remote =[:]
81.             remote.name = 'product1'
82.             remote.allowAnyHosts = true
83.             remote.port = SSH_REMOTE_PORT.toInteger()
84.             remote.host = "${SSH_REMOTE_HOST}"
85.             remote.user = 'pi'
86.             remote.identityFile = identity
87.             try{
88.               sshPut remote: remote, from: 'checkout.sh', into: '/tmp'
89.               sshCommand remote: remote, sudo: false, command: "cd␣/home/pi/your/path
                      &&␣rm␣-f␣build.log"
90.               sshCommand remote: remote, sudo: false, command: "chmod␣+x␣/tmp/checkout
                      .sh;␣chown␣pi␣/tmp/checkout.sh;␣chgrp␣pi␣/tmp/checkout.sh;␣cd␣/home/
                      pi/your/path;␣cp␣/tmp/checkout.sh␣.;␣dos2unix␣checkout.sh;␣./checkout
                      .sh"
91.               sshCommand remote: remote, sudo: false, command: "cd␣/home/pi␣&&␣python3
                      ␣your/path/utilities/scripts/compile_product1.py"
92.             }
93.             catch( err){
94.               //sshGet remote: remote, from: '~/your/path/build.log', into: '
                      product1_build.log', override: true
95.               sshGet remote: remote, from: '/home/pi/your/path/build.log', into: '
                      product1_build.log', override: true
96.               throw err
97.             }
98.             sshGet remote: remote, from: '/home/pi/your/path/build.log', into: '
                    product1_build.log', override: true
99.           }
100.
101.
```

```
102.         }
103.       }
104.     }
105.
106.     stage('Build product2') {
107.       when {
108.         expression {
109.           !skipRemainingStages
110.         }
111.       }
112.       steps {
113.         script{
114.           withCredentials([sshUserPrivateKey(
115.           credentialsId: "${SSH_CREDENTIALS_ID_U24}",
116.           keyFileVariable: 'identity',
117.           passphraseVariable: '',
118.           usernameVariable: 'yan'
119.           )]){
120.             def remote =[:]
121.             remote.name = 'Ubuntu24'
122.             remote.allowAnyHosts = true
123.             remote.port = SSH_REMOTE_PORT.toInteger()
124.             remote.host = "${SSH_REMOTE_HOST_U24}"
125.             remote.user = 'yan'
126.             remote.identityFile = identity
127.             sshPut remote: remote, from: 'checkout.sh', into: '/tmp'
128.             sshCommand remote: remote, sudo: false, command: "cd /home/yan/ \texttt{
                    list} && rm -f build.log"
129.             sshCommand remote: remote, sudo: false, command: "chmod +x /tmp/checkout.
                    sh; chown yan /tmp/checkout.sh; chgrp yan /tmp/checkout.sh; cd /home/
                    yan/your/path; cp /tmp/checkout.sh .; dos2unix checkout.sh; ./checkout.
                    sh"
130.             sshCommand remote: remote, sudo: false, command: "cd /home/yan && sudo
                    docker run --user dev -v ./your/path:/home/dev/your/path nexus.
                    apptricity.com/repository/apptricity/iot/u18owrt19:18.19 /bin/bash -c
                    \"cd ~/your/path; python3 ../../utilities/scripts/compile_product219.py
                    \" "
131.             sshGet remote: remote, from: '/home/yan/your/path/build.log', into: '
                    product2_build.log', override: true
132.           }
133.
134.         }
135.       }
136.     }
137.
138.     stage('Test') {
139.       when {
140.         expression {
141.           !skipRemainingStages
142.         }
143.       }
144.       steps {
145.         echo 'Test'
146.       }
147.     }
148.   }
```

```
149.
150.   post {
151.     always {
152.       script {
153.         def file1Content = fileExists('product1_build.log') ? readFile('product1_build
                .log') : null
154.         def file2Content = fileExists('product2_build.log') ? readFile('product2_build
                .log') : null
155.
156.         // Combine the contents
157.         def emailBody =
158.         """
159. ␣␣␣␣␣␣␣␣<html>
160. ␣␣␣␣␣␣␣␣<body>
161. ␣␣␣␣␣␣␣␣<h3>␣Commit␣id:␣${env.GIT_COMMIT}␣</h3>
162. ␣␣␣␣␣␣␣␣<h3>product1␣compilation␣output:</h3>
163. ␣␣␣␣␣␣␣␣<pre>
164. ␣␣␣␣␣␣␣␣
165. ␣␣␣␣␣␣␣␣${file1Content}
166. ␣␣␣␣␣␣␣␣</pre>
167. ␣␣␣␣␣␣␣␣
168. ␣␣␣␣␣␣␣␣<h3>product2␣compilation␣output:␣</h3>
169. ␣␣␣␣␣␣␣␣<pre>
170. ␣␣␣␣␣␣␣␣${file2Content}
171. ␣␣␣␣␣␣␣␣</pre>
172. ␣␣␣␣␣␣␣␣</body>
173. ␣␣␣␣␣␣␣␣</html>
174. ␣␣␣␣␣␣␣␣"""
175.
176.         emailext (
177.
178.         body: emailBody,
179.         mimeType: 'text/html',
180.         subject: '$DEFAULT_SUBJECT',
181.         to: email
182.         )
183.       }
184.     }
185.   }
186.
187. }
```

Below is Jenkinsfile in linux. you can name this file as Jenkninsfile_compile. The source code will be `\var\lib\jenkins\workspace`, the user is jenkins. When you

```
1.
2. def skipRemainingStages = false
3. def email = ""
4. pipeline {
5.   agent any
6.
7.   environment {
8.     CI = 'true'
9.     SSH_CREDENTIALS_ID = '5e8f5a33-ca59-4ba0-a9aa-9aef6e6b2720' //Don't change this
           value.
10.    SSH_REMOTE_PORT = 22
11.    SSH_REMOTE_HOST = '10.0.40.33'  //this is the ranger server IP address.
```

```
12.   }
13.   stages {
14.
15.     stage('Check␣for␣Changes') {
16.       steps {
17.         script {
18.
19.           sh 'rm␣-f␣ranger_build.log'
20.
21.           // Check if there are any change sets
22.           if (currentBuild.changeSets.isEmpty()) {
23.             echo "No␣changes␣detected,␣exiting␣pipeline."
24.             // Exit the pipeline without failing the build
25.             skipRemainingStages = true
26.             return
27.           }
28.         }
29.       }
30.     }
31.
32.     stage('Build␣Ranger') {
33.       when {
34.         expression {
35.           !skipRemainingStages
36.         }
37.       }
38.       steps {
39.         echo 'Build␣ranger.........'
40.
41.         script{
42.           sh '''
43. ␣␣␣␣␣␣␣␣␣␣(
44. ␣␣␣␣␣␣␣␣␣␣echo␣git␣stash
45. ␣␣␣␣␣␣␣␣␣␣echo␣git␣fetch
46. ␣␣␣␣␣␣␣␣␣␣)␣>␣checkout.sh
47. ␣␣␣␣␣␣␣␣␣␣'''
48.         }
49.         sh '␣echo␣git␣checkout␣${BRANCH_NAME}␣>>␣checkout.sh␣'
50.         sh '␣echo␣git␣pull␣>>␣checkout.sh␣' // if pull fail, we will not continue next
                compilation procedure.
51.
52.         script{
53.
54.           for (changeLogSet in currentBuild.changeSets) {
55.             // For each change set, iterate over the entries and get author name
56.             for (entry in changeLogSet.getItems()) {
57.               email = entry.authorEmail
58.               break  // Stop once the author name is found
59.             }
60.             if (email != "") {
61.               break  // Stop the outer loop once the author name is found
62.             }
63.           }
64.
65.           echo "Author␣Email␣${email}"
66.         }
```

```
67.
68.
69.         script{
70.           withCredentials([sshUserPrivateKey(
71.           credentialsId: "${SSH_CREDENTIALS_ID}",
72.           keyFileVariable: 'identity',
73.           passphraseVariable: '',
74.           usernameVariable: 'pi'
75.           )]){
76.             def remote =[:]
77.             remote.name = 'ranger'
78.             remote.allowAnyHosts = true
79.             remote.port = SSH_REMOTE_PORT.toInteger()
80.             remote.host = "${SSH_REMOTE_HOST}"
81.             remote.user = 'pi'
82.             remote.identityFile = identity
83.             try{
84.               sshPut remote: remote, from: 'checkout.sh', into: '/tmp'
85.               sshCommand remote: remote, sudo: false, command: "cd␣/home/pi/gen2-icont
                    -unified/LTKC/iController␣&&␣rm␣-f␣build.log"
86.               sshCommand remote: remote, sudo: false, command: "chmod␣+x␣/tmp/checkout
                    .sh;␣chown␣pi␣/tmp/checkout.sh;␣chgrp␣pi␣/tmp/checkout.sh;␣cd␣/home/
                    pi/gen2-icont-unified;␣cp␣/tmp/checkout.sh␣.;␣dos2unix␣checkout.sh;␣
                    ./checkout.sh"
87.               sshCommand remote: remote, sudo: false, command: "cd␣/home/pi␣&&␣python3
                    ␣gen2-icont-unified/utilities/scripts/compile_ranger.py"
88.             }
89.             catch( err){
90.               //sshGet remote: remote, from: '~/gen2-icont-unified/LTKC/iController/
                    build.log', into: 'ranger_build.log', override: true
91.               sshGet remote: remote, from: '/home/pi/gen2-icont-unified/LTKC/
                    iController/build.log', into: 'ranger_build.log', override: true
92.               throw err
93.             }
94.             sshGet remote: remote, from: '/home/pi/gen2-icont-unified/LTKC/iController
                    /build.log', into: 'ranger_build.log', override: true
95.           }
96.         }
97.
98.       }
99.     }
100.
101.    stage('Build␣old␣Pathfinder') {
102.      when {
103.        expression {
104.          !skipRemainingStages
105.        }
106.      }
107.      steps {
108.        echo 'Build␣old␣Pathfinder.....'
109.
110.        script{
111.          sh 'rm␣-f␣LTKC/iController/build19.log'
112.          docker.image('nexus.apptricity.com/repository/apptricity/iot/u18owrt19:18.19
                ').inside('--user␣root') {
113.            sh 'chmod␣-R␣o+rw␣.␣&&␣su␣dev␣-c␣"␣git␣config␣--global␣--add␣safe.
```

```
                directory␣"$(pwd)"␣&&␣touch␣LTKC/iController/insideDocker␣&&␣cd␣LTKC/
                iController␣&&␣python3␣../../utilities/scripts/compile_pathfinder19.py␣
                &&␣rm␣-f␣LTKC/iController/insideDocker␣"␣'
            }
          }
        }
      }

      stage('Build␣new␣Pathfinder') {
        when {
          expression {
            !skipRemainingStages
          }
        }
        steps {
          echo 'Build␣new␣Pathfinder.........'

          script{
            sh 'rm␣-f␣LTKC/iController/build22.log'
            docker.image('nexus.apptricity.com/repository/apptricity/iot/u22owrt22:22.22
                ').inside('--user␣root') {
              sh 'chmod␣-R␣o+rw␣.␣&&␣su␣dev␣-c␣"␣git␣config␣--global␣--add␣safe.
                  directory␣"$(pwd)"␣&&␣touch␣LTKC/iController/insideDocker␣&&␣cd␣LTKC/
                  iController␣&&␣python3␣../../utilities/scripts/compile_pathfinder22.py␣
                  &&␣rm␣-f␣LTKC/iController/insideDocker␣"␣'
            }
          }
        }
      }
// 1) from linux, when you go into the docker, the /var/lib/jenkins/branch will be
    mapped into the same directory inside the docker.
//2) but the user is 134:145, (jenkins outside, no name inside docker)
//3) use root log in, then add other user to make /var/lib/jenkins/branch accessable
    by other user(including dev)
//4) change to dev, don't use su - dev, it will going into the another log in shell.
    (must make sure all comand inside one shell, )
//5) that is why we use -c make all continues commands inside one shell.
//6) git config make dev and jenkins can works properly.
//7) why we need to switch to dev, because inside docker, some cross platform compile
    is made by dev
//8) there are three user, dev inside docker, root inside docker, jenkins outside
    docker. The source code outside of docker owner is jenkins.

      stage('Test') {
        when {
          expression {
            !skipRemainingStages
          }
        }
        steps {
          echo 'Test'
        }
      }
    }

    post {
```

```
158.     always {
159.       script {
160.         def file1Content = fileExists('ranger_build.log') ? readFile('ranger_build.log
                 ') : null
161.         def file2Content = fileExists('LTKC/iController/build19.log') ? readFile('LTKC
                 /iController/build19.log') : null
162.         def file3Content = fileExists('LTKC/iController/build22.log') ? readFile('LTKC
                 /iController/build22.log') : null
163. 
164.         // Combine the contents
165.         def emailBody =
166.         """
167. ␣␣␣␣␣␣␣␣<html>
168. ␣␣␣␣␣␣␣␣<body>
169. ␣␣␣␣␣␣␣␣Check␣Jenkins␣console␣output␣is␣<a␣href="${BUILD_URL}">here␣</a>.␣Jenkins␣is␣
    working␣on␣commit␣id:␣<b>␣${env.GIT_COMMIT}␣</b>,␣compare␣this␣commit␣id␣with␣
    another␣commit␣id␣in␣below␣ranger/pathfinder␣compilation␣output,␣two␣ids␣should␣be␣
    identical.
170. ␣␣␣␣␣␣␣␣<h4>Ranger␣compilation␣output:</h4>
171. ␣␣␣␣␣␣␣␣<pre>
172. ␣␣␣␣␣␣␣␣
173. ␣␣␣␣␣␣␣␣${file1Content}
174. ␣␣␣␣␣␣␣␣</pre>
175. ␣␣␣␣␣␣␣␣
176. ␣␣␣␣␣␣␣␣<h4>Old␣Pathfinder␣compilation␣output:␣</h4>
177. ␣␣␣␣␣␣␣␣<pre>
178. ␣␣␣␣␣␣␣␣${file2Content}
179. ␣␣␣␣␣␣␣␣</pre>
180. ␣␣␣␣␣␣␣␣
181. ␣␣␣␣␣␣␣␣<h4>New␣Pathfinder␣compilation␣output:␣</h4>
182. ␣␣␣␣␣␣␣␣<pre>
183. ␣␣␣␣␣␣␣␣${file3Content}
184. ␣␣␣␣␣␣␣␣</pre>
185. ␣␣␣␣␣␣␣␣
186. ␣␣␣␣␣␣␣␣</body>
187. ␣␣␣␣␣␣␣␣</html>
188. ␣␣␣␣␣␣␣␣"""
189. 
190.         emailext (
191. 
192.         body: emailBody,
193.         mimeType: 'text/html',
194.         subject: '$DEFAULT_SUBJECT',
195.         to: email
196.         )
197.       }
198.     }
199.   }
200. 
201. }
```

3.4 Docker

build my own image and upload to docker hub.

1. sudo docker pull ubuntu:22.04, then sudo docker images check if you download this image
2. sudo docker run -it ubuntu:22.04, create a container, and come into this container.
3. run all the command you want to config this container,
4. inside docker, you are root, you can use `$ adduser -disabled-password -gecos "" dev ↵` and `$ su - dev ↵` to add and switch to dev account.
5. inside docker, make everything ready. then exit to host `$ sudo docker ps -a ↵` list container id
6. `$ sudo docker commit container_id image_name:tag ↵` then use `$ sudo docker images ↵` to check all the existing images and its size.
7. `$ sudo docker save -o u22.tar u22owrt22:22.22 ↵` save it to tar file to avoid lost.
8. Before push, you need to login first, then use tag to specify the directory in the docker hub website. In the end, run push.

```
1. sudo docker tag u22owrt22:22.22 nexus.***.com/repository/apptricity/iot/u22owrt22
   :22.22
2. yan@yan-VMware-Virtual-Platform:~$ sudo docker push nexus.***.com/repository/
   apptricity/iot/u22owrt22:22.22
```

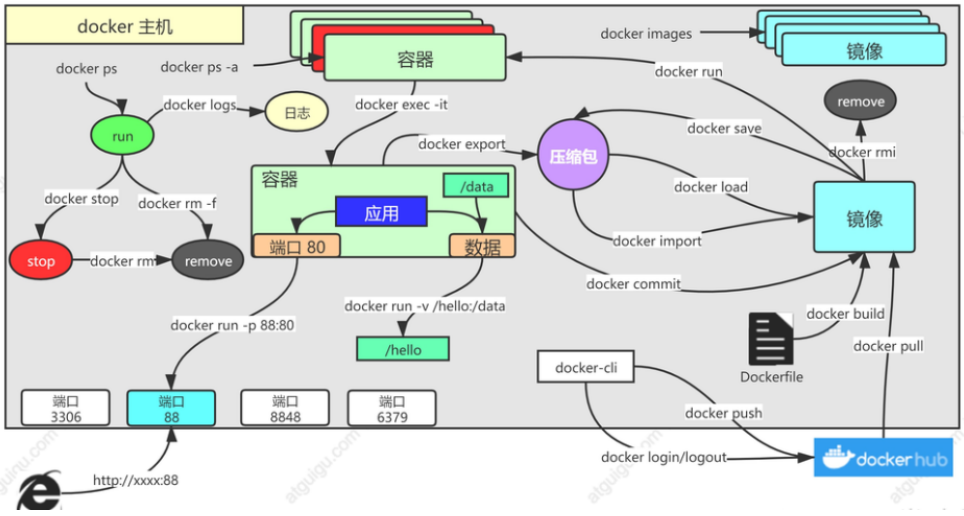
docker only support same architecture. If guest os is arm, then host os can't be x86, unless you use emul.

container do not have any system services running. so a container will not have an SSH server.

curl http://localhost:8080/

save, then load, if import then export. They are different, you can google them. save from image, so it includes layer information, but export from container.

if you want to push, you need to tag it first, such as ubuntu:18.04, 18.04 is a kind of tag, don't forget these number. Once you use this tag name to push, use the same tag name to pull for docker server, such as docker hub.



difference between docker rm and rmi, docker run and docker exec

sudo docker login nexus.***.com sudo docker pull nexus.***.com/repository/***/iot/u18owrt19:18.19 sudo docker push ??

3.5 Clang

This is command to output llvm code.

```
clang++ -S -emit-llvm main.cpp -o out.bc
```

3.6 gcc

3.6.1 gcc basic

You can use md5sum to judge if two exes are the same. but if you use gcc, when you compile a.c with -g, it will include directory name, so the two binary executable on the different two computers maybe are different(due to the different directly, maybe different time stamp).

`$ 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.

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. But sometimes, these warning 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.

Higher levels of optimization can restrict debug visibility and increase compile times. It is usual to use -O0 for debugging, and -O2 for finished code. When using the above optimization options with the -g (debug) switch, it can be difficult to see what is happening. The optimizations can change the order of statements or remove (or add) temporary variables among other things. But an understanding of the kinds of things the compiler will do means that satisfactory debug is normally still possible with -O2 -g.

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.

Check lib:

1. Use $ ldd ↵ can see what libs does an application depends on. ldd works on dynamically linked executables (which depend on .so libraries).
2. objdump: Another way to check dependencies (more detailed, especially for statically linked executables). $ objdump -x /path/to/executable | grep NEEDED ↵
3. readelf $ readelf -d /path/to/exe ↵

$ 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".

3.6.2 include

Default, 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. -E is to stop after preprocess. -x C++ is to specify c++ language. -v is to show all command g++ called. another dash is to institute filename with stdin. The usage is like $ gzip | tar -. ↵

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.

3.6.3 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.

3.6.4 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 declaration.
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. `$ gcc hello.c -fPIC -shared -Wl,-soname,libhello.so.0 -o libhello.so.` ↵ Why do we need three names? how to understand -soname? That is a long story, let me explain below:

1. The first number is primary version, should change when API change(which make thing is not compatible).
2. The second number is sub version, when you adding API(back compatible)
3. The the third number is minor version. fix bug or imporve speed(compatible)

SONAME is used at compilation time by linker to determine from the library file what actual target library version. gcc -lNAME will seek for libNAME.so link or file then capture its SONAME that will certainly be more specific (ex libnuke.so links to libnuke.so.0.1.4 that contains SONAME libnuke.so.0).

'SONAME' of library can be seen with 'objdump -p file |grep SONAME'

I think that not providing a soname is a bad practice since renaming of file will change its behavior.

When you have libhello.so.0.0.1. run `$ ldconfig -n .` ↵, It will produce a soft link libhello.so.0

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

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.

nm -g libhello.so will list all symbol in a .so file

.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.

3.6.5 load .so

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.

```
1. The LD_DEBUG environment varaible.
2. http://www.bnikolic.co.uk/blog/linux-ld-debug.html
3.
4. 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

```
1. #include <stdio.h>
2. #include <dlfcn.h>
3. int main(int argc, char *argv[]){
4.   void *dl = NULL;
5.   int (*add)(int a, int b);
6.   dl = dlopen( "./libtest.so", RTLD_LAZY);
7.   if( dl == NULL ){
8.     printf("so loading error.\n");
9.     return 1;
10.  }
11.    add = (int(*)(int, int))dlsym(dl, "add");
12.  if( dlerror() != NULL ){
13.        printf("fun load error.\n");
14.    return 1;
15.  }
16.  printf("%d\n", add(1, 2));
17.  return 0;
```

```
18. }
```

3.7 gdb

3.7.1 start

-g and -O are different, even -O3 will also include debug information, you can use strip to delete all the debug information.

`$ 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.

3.7.2 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.

3.7.3 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` ↵

3.7.4 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.

3.7.5 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.

3.8 Automaticly Build

3.8.1 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.

A key idea, what perform in makefile? and what perform in shell?

```
Target: prerequisites
 command #this perform in shell
 VAR=abc #VAR will not become valid after this line,
         #I did makefile refactoring in company.

ifeq ($(MAKEGOALS) , target)  #this will perform inside make file script
VAR=abc
endif
```

will list all the command it will perform when give a target.

```
make -n -f makefile grammar.o
```

make file has implicit rule. how to config it?

```
%.c : %.y #disable all the implicit rule from *.y to *.c, yacc example
```

The basic part of Makefile is:

```
Target: prerequisites
  tab command
```

Build branch in makefile, only use ifeq in makefile script. If you want to use a command after tab, you have to set a target, it's not very convenient.

```
OWRT := ../../../owrt19075
OWRT_EXISTS :=$(shell [ -d $(OWRT) ] && echo yes || echo no)
PATH_DIR := $(shell echo $$PATH)

ifeq ($(OWRT_EXISTS),yes)
STAGING_DIR_MAK = ~/owrt19075/openwrt/staging_dir
TOOL_CHAIN = toolchain-arm_cortex-a5+vfpv4_gcc-7.5.0_musl_eabi
else
```

```
STAGING_DIR_MAK = ~/owrt22033/openwrt/staging_dir
TOOL_CHAIN = toolchain-arm_cortex-a5+vfpv4_gcc-11.2.0_musl_eabi
endif

LIB_DIR  = $(STAGING_DIR_MAK)/target-arm_cortex-a5+vfpv4_musl_eabi/usr/lib/
INCLUDE_DIR = $(STAGING_DIR_MAK)/target-arm_cortex-a5+vfpv4_musl_eabi/usr/include/
INCLUDE_ZLIB = $(STAGING_DIR_MAK)/host/include/

export PATH=$(PATH_DIR):$(STAGING_DIR_MAK)/$(TOOL_CHAIN)/bin
export STAGING_DIR=$(STAGING_DIR_MAK)/$(TOOL_CHAIN)
```

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 – 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.

```
1. all: all_1
2.   rm a.txt
3.
4. all_1:
5.   @echo "no go to here"
6.   -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"
```

```
48. # should execute the content of their build rules,
49. #even if a newer file named "all", "clean", or "distclean" exists.
50. .PHONY: all clean distclean
51.
52.
53. tags:
54.   echo "$(program_CXX_SRCS)" > cscope.files
55.   rm cscope.out cscope.in.out cscope.po.out
56.     cscope -Rbq
57.   rm tags
58.   ctags $(program_C_SRCS) $(program_CXX_SRCS)
59.
60. # This is first build rule in the makefile,
61. # "make" and executing "make all" are the same.
62. # The target simply depends on $(program_NAME),
63. # which expands to "myprogram", and that target is given below:
64. all: $(program_NAME)
65.
66.
67. # The program depends on the object files
68. #(which are automatically built using the predefined build rules...
69. #nothing needs to be given explicitly for them).
70.
71. # The build rule $(LINK.cc) is used to link the object files
72. # and output a file with the same name as the program.
73. # Note that LINK.cc makes use of CXX, CXXFLAGS, LDFLAGS, etc.
74. # On my own system LINK.cc is defined as:
75. # $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH),
76. # so if CXXFLAGS, CPPFLAGS, LDFLAGS, and TARGET_ARCH are undefined,
77. # but CXX is g++, expand to g++ $(program_OBJS) -o $(program_NAME).
78.
79. $(program_NAME): $(program_OBJS)
80.     $(LINK.cc) $(program_OBJS) -o $(program_NAME)
81.
82. #
83. # This target removes the built program and the generated object files.
84. # The @ symbol indicates that the line should be run silently,
85. # and the - symbol indicates that errors should be ignored
86. # (i.e., if the file already doesn't exist, we don't really care,
87. # and we should continue executing subsequent commands)
88.
89. clean:
90.     @- $(RM) $(program_NAME)
91.     @- $(RM) $(program_OBJS)
92.
93. #
94. # The distclean target depends on the clean
95. #target (so executing distclean will cause clean to be executed),
96. # but we don't add anything else.
97.
```

```
98. distclean: clean
```

3.8.2 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.

3.9 putty

load, change something, then come back to Session part to save.

In putty, highlight is just copy and right click is just paste.

putty can input user@host.com directly, you can avoid inputting user name too.

in github, there is solarized_dark.reg, go to your reg editor, find

```
[HKEY_CURRENT_USER\Software\SimonTatham\PuTTY\Sessions\session_name],
```

then export , in exported file, use solarized_dark.reg color0 to color24 replace color0 to color24, then run reg import exported file. In this way, when you open putty, the interface will become solarized.

By now, VS code, putty and texstudio are become solarized theme.

3.10 ssh

Three common used pattern when use ssh:

- local to access server(board). ssh-keygen on local, then ssh-copy-id to server.
- local to access github. From desktop, copy private key to new computer, then modify conif
- Jenkins to access server, copy private key and paste it to Jenkin credentials.

ssh is Secure Shell

The basic steps:

1. The first step, make sure on server side, run `$ sudo service sshd status ↵`, it shows that ssh service is running correctly.
2. on you local, run `$ ssh-keygen ↵`
3. copy public key file to the server by using `$ ssh-copy-id remot_username@remote_server_ip_address ↵`. If you want to use ssh-copy-id, you can go into the Git Bash console. There is ssh-copy-id is OK. In most of Linux, it's pre-installed. But in windows, you can use below in power shell cmd.

   ```
   type %USERPROFILE%\\.ssh\\id_rsa.pub | ssh user@host "mkdir -p ~/.ssh && cat >> ~/.ssh/
   ```

4. modify .ssh/config. After this step, you can ssh from any cmd and vs code now.

   ```
   Host 192.168.0.182
    HostName 192.168.0.182
    IdentityFile ~/.ssh/id_rsa
    User pi
   ```

5. Run ssh -T user@server.com(or ip) to make sure your ssh has been configured properly.

You should **NEVER** save the file with its contents starting with —–BEGIN RSA PRIVATE KEY—– on the server(which you want to access, such as github or remote board.), that is your private key. Instead, you must put the public key into the /.ssh/authorized_keys file. This public key has the .pub extension when generated using ssh-keygen and its contents begin with *ssh-rsa AAAAB3*.

The permissions of /.ssh on the server should be 700. The file /.ssh/authorized_keys (on the server) is supposed to have a mode of 600. The permissions of the (private) key on the client-side should be 600.

There are three ssh related tools: Winscp and putty and vs code. The main purpose of ssh is to avoid input password again and again when you use ssh log in the remote server.

winscp can save password directly. It also support edit file directly, but is not as well as vs code.

putty can also avoid password.

```
in local: ssh-keygen -t rsa -b 4096 # without -b, it will use 2048.

in server: Or, ssh-copy-id -i ~/.ssh/id_rsa.pub remote-user@remote-host in the terminal.
Or, completely manually step-by-step:

Create a directory (if it doesn't exist already) named .ssh in the home directory of the remo
In that directory, create a file named authorized_keys (if it doesn't exist already).

In case your remote umask is more liberal than usual, make the file not group-writable: chmod

Finally, somehow copy (append) the contents of your local public key (~/.ssh/id_rsa.pub) into


in local: puTTY cannot use keys in OpenSSH format.

You need to convert your key to .ppk format first. For that, use PuTTYgen from PuTTY package.
```

```
Run PuTTYgen;
Press Load to load the private key in OpenSSH format;
Press Save private key to save the private key in .ppk format
```

After all above steps, putty can access remote server without input password.

After you did above, you vs code can open remote directly. install remote-ssh extension in vs code.

3.11 texstudio

there is texstudio.ini file in github, copy it to `C:\Users\yzhao\AppData\Roaming\texstudio`. before you copy there, copy old texstudio.ini in case of crash. it use Solarized Dark theme. VS studio also support Solarized dark theme.

3.12 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. In Mac, you can find this file here: /Users/yan.zhao/Library/Preferences/doublecmd/doublecmd.xml, but dark mode doesn't work well on Mac, solarized theme doesn't work.

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.

input letter, it will pop up filter windows to help you find file very quickly.

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
ctrl+.	show hide
alt+enter	show property
alt+0,1,2	tab switch
alt+down	dir history
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

3.13 IDE

3.13.1 VSCode

copilot

VS can install copilot extension. There is yzhao@mycompany and job pw.
accuracy decrease according to this order.

1. ghost text
2. ctrl+i(existing prompt /fix with context)
3. ctrl+i(your own prompt,with context,selected)
4. ctrl+i(your own prompt, without context)

Code completion and code suggestion are related, but they are not exactly the same.
code suggestion (ghost code):

1. tab :accept
2. ctrl + -> : accept part
3. esc : not accept,
4. disable, need to disable whole copilot

code suggestion sometimes is a little eager, there are two options, 1) just keep typing and don't type tab unless you want to accept it. 2) totally disable copilot.

Inline suggestions are great at autocompleting a section of code. But since most coding activity is editing existing code, it's a natural evolution of Copilot code completions to also help with edits, both at the cursor and further away. Edits are often not made in isolation - there's a logical flow of what edits need to be made in different scenarios. Copilot Next Edit Suggestions (Copilot NES) is this evolution.

To enable or disable code completions, select the Copilot menu in the Status Bar, and then check or uncheck the options to enable or disable code completions.

When you highlight some code or maybe you can see whenever there's a red squiggle due to a compiler error. Let's use AI to fix a coding error.

press Ctrl+I on your keyboard to bring up editor inline chat.

press Ctrl+alt+ I will open copilot panel on right. While the Chat view is great for keeping a conversation going, editor inline chat is optimized for situations where you want to ask Copilot about the code you're actively working on in the editor.

config for C++ and python

VS can install support python, install python first, then you can use venv directly.

VS can also use C++, install C++ extension. and you can also install cl.exe. use developer command prompt, then run "code" command, then you can press F5 to compile your program directly. Google "Configure VS Code for Microsoft C++" or ask directly in chatgpt. You need to run code command in "Developer PowerShell for VS prompt", then just ctrl+f5 will run your c++ programme. The first time you run your program, the C++ extension creates tasks.json, which you'll find in your project's .vscode folder. tasks.json stores build configurations.

A common config is add below to "args":

```
1. "/std:c++latest",
2. "${workspaceFolder}\\*.cpp" instead of "${file}"
```

When you debug with the play button or F5, the C++ extension creates a dynamic debug configuration on the fly. There are cases where you'd want to customize your debug configuration, such as specifying arguments to pass to the program at runtime. You can define custom debug configurations in a launch.json file. Change the stopAtEntry value to true to cause the debugger to stop on the main method when you start debugging.

```
1. "stopAtEntry": false,
```

tips

move cursor, cmd+left arrow (windows home) line begning. option(window ctrl) +left arrow, move to begining of word. cmd+shift+ jump to match bracket. come back to edit (ctrl+K, ctrl+q) and come back to last cursor(ctrl +u) are two most important move commands.

about cursor jump, another big topic is symbol jump. crtl+shift +O will list all symbol, f12 go to implemenation(C++). click a function, then right click mouse, you can see a list of useful command. That is also very important.

some command has no keyboard short cut, but you can ctrl+shift+p call out command palete, then search relateated command, then run it. so command palete is very important conception in vs code.

ctrl + F2 add multi cursor fucntion, it's very useful for refactor, such as change variable name.

Once you find a command in command palete is useful, you can bind a keyshort to it. just input any combination, if this combination has been used, the vs code will tell you this key has been used by another command.

The common key shortcut:

```
C+b, C+j, open or close panel and view.
C+` , open or close terminal

F12, go to definition.
S+F12, go to reference.
Conmmand+1, 2,3 , go to the split group, then ctrl+tab switch document
ctrl + 1, 2, 3 switch document(tab)

F1, then @, it will pop up all symbols. that is very useful.
C+S+O, it will pop up all symbols.

commnd +k, c  comment
commnd +K, u  uncomment.
command +shift +k delet line

alt +upper arrow,  move line up
shift+alt+upper, copy line up

command +upper, beginning of file
home, beginning of line

command(ctrl) + f2, delete multi symbol

ctrl+-, ctrl+shift+-  , go back/forward.
command k, q  go back to last edit.

for texstudio,  alt+left/right is go back/forward, and ctrl+h go back to last edit.
```

ctrl+shift+p configure task, will generate task.json. crtl+shift+D, click create a launch.json.

usually, you only need to do it in Linux, In window, you can use vs studio directly. The detail can be google. It's very common.

ctrl+shift+o: list all symbol, ctrl+p list all files.

There are three json files you need to know. With c_cpp_properties.json, you can resolve includes can't find header files. F12 also can jump into the definition in header files. You don't need compileCommands if you can build your own task.json

```
//c_cpp_properties.json
{
    "configurations": [
        {
            "name": "Linux",
            "includePath": [
                "${workspaceFolder}/**"
            ],
            "defines": [],
            "compilerPath": "/usr/bin/gcc",
            "cStandard": "gnu17",
            "cppStandard": "gnu++14",
            "intelliSenseMode": "linux-gcc-x64",
            "compileCommands": "/home/yan/build/debug-cpp11/compile_commands.json"
        }
    ],
    "version": 4
}
```

task.json. You can use C+S+B to run this task.json. You need to switch back to the correct C/C++ file. No any project need this task.json. If you have Cmake build system outside. you can go to the terminal and run cmake or make there. YOu don't need use task.json at all. But you still need to launch.json if you want to debug the exe. if you wanto debut the exe, you should use DCMAKE_BUILD_TYPE=Debug to run cmake. to make exe has debug information.

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: cpp build active file",
            "command": "/usr/bin/g++",
            "args": [
                "-g",
                "${file}",
                "-L",
                "/home/yan/build/debug-cpp11/lib",
                "-lmuduo_net",
                "-lmuduo_base",
                "-lmuduo_http",
                "-lmuduo_inspect",
                "-lmuduo_pubsub",
                "-lpthread",
                "-I",
                "/home/yan/muduo-master",
                "-DCHECK_PTHREAD_RETURN_VALUE -D_FILE_OFFSET_BITS=64 -Wall -Wextra -
                    Werror -Wconversion -Wno-unused-parameter -Wold-style-cast -
                    Woverloaded-virtual -Wpointer-arith -Wshadow -Wwrite-strings -march
                    =native -std=c++11 -rdynamic ",
                "-o",
                "${fileDirname}/${fileBasenameNoExtension}"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: /usr/bin/cpp"
        },
    ]
}
```

launch.json. in the Run and Debug view, you can click add launch.json link, it will add a template, then in this template, you can click add configuration, then select C/C++, it will produce the most of item below. you only need to change "program",

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
```

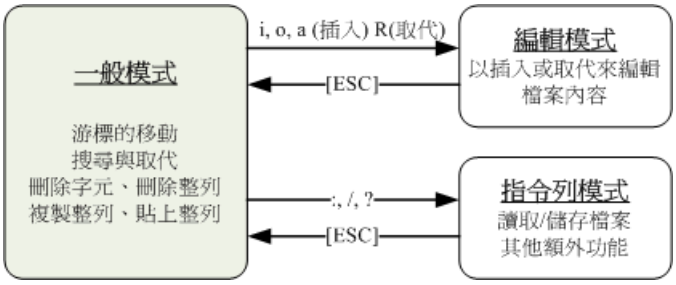
```
            "name": "(gdb) Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "/home/yan/build/debug-cpp11/bin/twisted_finger07",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${fileDirname}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                },
                {
                    "description":  "Set Disassembly Flavor to Intel",
                    "text": "-gdb-set disassembly-flavor intel",
                    "ignoreFailures": true
                }
            ]
        }

    ]
}
```

Chapter 4

Vim

4.1 basic knowledge

There are three basic modes inside the vim. You can also think that visual is another mode.



4.1.1 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: In insert mode, there are some useful commands you can use 1) ctrl+r 2) ctrl+x, 3) ctrl+h,u,w, 4) ctrl+o. 5) ctrl+t,d. The first two can be used for insert, the third can be used deleted, and the last one can be used temperably come back to normal mode.

Rule6: 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).

Rule7: Learn to use dot command more, detail can be seen below dot section.

Rule8: 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. If there is a repeated task, you can use three methods:

1. dot formual(one move , one dot)
2. Macro
3. Ex command, use normal or global apply commands or macro on a range.

Rule 9: move you cursor quicker 1) use H,M,L or zt, zb or ctrl+f,d. 2) use marks more to jump back. 3)‘. ‘^(gi) to jump back insert point.

Rule 10: Know operator, operator+motion and operator pending state:

```
c Change
d Delete
y Yank into register
g~ Swap case
gu Make lowercase
gU Make uppercase
> Shift right
< Shift left
= Autoindent
! Filter {motion} lines through an external program
// https://vimways.org/2019/vim-and-the-shell/ This is a good reference
// to introduce ! Filter.
```

Rule 11: use more autocomplete in insert mode 1) Ctrl+x 2) Ctrl+e 3) Ctrl+j 4)Ctrl+l(YCM)

Rule 12: Know basic regex and
v and
V, they can be used in /, ? and s command and also vimgrep

Rule 13: To know more Ex command to deal with more large scale tasks. s, normal and global should be used more.

```
:[range]delete [x]  jDelete specified lines [into register x]
:[range]yank [x]    Yank specified lines [into register x]
:[line]put [x]      Put the text from register x after the specified line
:[range]copy {address} Copy the specified lines to below the line specified by {address}
:[range]move {address} Move the specified lines to below the line specified by {address}
:[range]join        Join the specified lines
:[range]normal {commands} Execute Normal mode {commands} on each speci- fied line
:[range]substitute/{pattern}/{string}/[flags] Replace occurrences of {pattern} with {string}
:[range]global/{pattern}/[cmd]
```

Rule 14: now v, V and ctrl+V. gv is select the content again and o is jump between begin and end of selection.You should know how to use A command to append after a ragged visual block.

Rule15: 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, More practice to form muscle memory.

Rule16: 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>.

4.1.2 Basic style

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>.

4.1.3 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.

dot command is related with last, last content, last command, last jump

4.2 Basic command

4.2.1 Motion command

Move in insert mode

This part maybe a little obsolete, in insert mode, you can use some existing commands, not create your own command. 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

[*I* will open a preview windows and list all these references. and [*i* just peek it in the bottom. This command only search in current .cpp and .h file. It will not search all the .cpp file. If you want to search in all .cpp file, you can use EasyGrep.

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 ^ $ 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), ; $,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 recorded 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-^> 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

Basic command is:

1. mark: ma
2. go to line 'a
3. go to the exact postion ‘a

Other points about mark command.

1. :marks command will show you all the marks list.
2. 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.
3. 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.
4. `. 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.)
5. 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: The basic command of position list

```
:lv word %  # % is current file
:lopen - Open the location list window.
:lcl or :lclose - Close the location list window.
```

```
:lnext - Go to the next item on the list.
:lprev - Go to the previous item on the list.
:lfirst - Go to the first item on the list.
```

position list is associated with current window, and quickfix list is global.

You can use :lne :lw :lpr to manipuate position list windows. You can use :cne :cw :cpr to manipuate quickfix windows.

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.

You can use # and * command, you don't need to input word which you want to search.

If you use / or ?, you can use <C-r> <C-w> to paste current word to command windows. or you can input manually.

Press F4 to toggle highlight research result.

regex in search command:

1. **vimgrep is super slow in big project, So I mainly use below methods: 1) :GrepOptions, then use c command change grep command from vimgrep to grep. 2) Then, use r to change it to recursive. 3) check pwd and extension configuration(for C++ big project), 4) For whole word, use \vV. 5) for "abc.h", use :Grep \<abc\.h\>**
2. When you use :Grep to search word boundaries, use "\b[a-z]+\b" or "\<[a-z]+\>", when use vim, use "\v<[a-z]+>".
3. **You can use :Grep with regex directly, \vv is just :Grep $***.

   ```
   :Grep \<abc\.h\> will find all abc.h head file. Then you can use
   :GrepOptions, to config, 1) which command 2) which file.
   3) recursive. and so on.
   ```

4. A few points which you need to know 1) egrep just equal grep -E. 2) -E use extended regex. The difference between basic and extended is if a few special character has special meaning. For example. '?', '+', parentheses, braces ('{}'), and '|'. extended version has special maning. so "a+b" will match aab, not a+b.
5. just like previous, vim has the same idea, we use \v to represent "magic", magic means that a few special character has specail meaning. so "\va+b" will match aab, not a+b.
6. **Most of time, I just use -E in grep and \v in vim. don't need to know all the regex syntax.**.
7. \vV is more usefule than \vv. it will only match the whole word. Both \vv and \vV has a problem, it can't used to search "abc.h", because \vv only pick up the word from current cursor, "abc.h" is not word. In this way, we can use :Grep command directly.
8. vimgrep use // to enclose pattern, but when you use / it's a little different, /abc/e search abc and put cursor to end.
9. Know how to search and substitude the last search.
10. vimgrep is slow, You can use :grep to use external grep to search, it's much faster. But in neovim, you have to use -r and –color=never flag. Otherwise, result includes color code and can't be shown properly in neovim. I have config grep in .vimrc to make –color=never.
11. patten example 1) escape html 2) search duplicate word.
12. There are two subtopics, One is regex syntax is different between grep, python and vim. The detail can be see below link. And a good regex example can be found here:

https://remram44.github.io/regex-cheatsheet/regex.html#syntax-basics
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 :jumps :reg :change are four useful commands.
Summary:

1. Three useful search commands:

```
/\v^abc\c  # search "abc" in the beginning of line, \c is case insentive. \v is
    magic.
# / is command, NO SPACE AFTER /

:lv /\vabc\c/ %  #MUST use pair of / include pattern. \v and \c has some meaning.
#% is current file.

:%s/\vabc\c/www/g #MUST use pair of / include pattern.  g is gloable, otherwise,
#it only replace the first word. NO SPACE AFTER s command.
```

4.2.2 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>	dˆ or cˆ
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. **c,d,y can add "a(register) in front of the command. p command can also follow "a(register name)**
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 dz, then input the two letters to specify the position, to give exact position, Or you can use df, than use dot command to repeat it.**

4.2.3 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).
Summary:

1. H,M,L: move cursor.
2. Ctrl+u,d,e,y: move documents.
3. zz,zt,zb: move both

4.2.4 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.

4.2.5 Buffer

Manage buffer:

Action	insert mode	normal mode
edit a file in a new buffer		:e file
next, previous buffer	RR	:bn :bp , <C-^>
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.

4.2.6 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.**

4.2.7 Register

You can specify a register by prefixing the command with " followed by the register name:

Command	Meaning
`"ayy`	Yank a line into register `a`
`"ap`	Paste from register `a`
`"adw`	Delete a word into register `a`
`"0p`	Paste the last yanked text
`"+y` / `"+p`	Yank/paste from the system clipboard 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.

Default register is ", so p command is equal ""p . " will change when y and delete, and 0 will save y. and 1 9 save large block delete.

". 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 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.

4.2.8 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

4.2.9 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.

4.2.10 macro

All command need run in normal mode.

1. qa : recording to register a
2. q: stop
3. @a: apply macro

4.2.11 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.

4.3 Text Block

4.3.1 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("
head	(a,i)(t,<)	%(matchit), dst, cst<h1>, 1) phh(Emmet) 2)select,S, (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".

4.4 Configure a new computer

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
```

1. Run `$ git clone https://github.com/VundleVim/Vundle.vim.git  /.vim/bundle/Vundle.vim` ↵
2. 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. The new configration is vimrc_fedora. **You should use vimrc_fedora and change it to .vimrc**. The below contents are based on new vimrc. so please download vimrc_fodera and change it to .vimrc. The other one is for old reference purpose.
3. Go to my github and download vim/ycm_extra_conf.py and rename to .vim/.ycm_extra_conf.py. YCM will use it.
4. Go to my github and download vim. Then copy three child directories "after", "plugin" and "macros" to .vim directory.
5. Open a vim and then run :PluginInstall
6. Go to YCM homepage to see how to intall YCM. In a new version Linux, this can be done very easy.
7. 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. Download Gnome tweak, then you can swap Esc and Cap there.
8. Use which command to see if xclip, ctags and cscope have been installed. use sudo apt-get install xclip.
9. sudo apt-get install exuberant-ctags
10. sudo apt install cscope
11. if you use neovim, you need to install python2 pip to install pynvim, it will make YCM work. below code is how to install python2 pip

```
sudo dnf install python2
....
$ python2 -m ensurepip --no-default-pip
...
$ pip2 --version
```

By now, I begin to use neovim, so please google how to install neovim in you linux platform.

put init.vim to /.config/nvim.

The newest vimrc name is vimrc_fedora, please copy it to your home directory and rename it as .vimrc

go to the /vim/bundle/Visual-Mark/plugin/, change the visualmark file. There is space after mm keymap, On some system, F2 will bring "Q" into the input, then it will trigger VIM EX command mode. If you want to use visual mark plug in, please go to /home/yzhao/.vim/bundle/Visual-Mark/plugin/ and edit visualmark.vim. change F2 and Ctrl+F2 to [m and]m. Because for some linux platform, F2 and F3 doesn't work properly.

I didn't use easymotion anymore, because it's slow for big document, by now I use hop.vim and I have new configuration in init.vim

New version

1. google how to install vim 8
2. after install vim 8, (it supports + register).
3. run PluginUpdate
4. go to .vim/bundle/YouCompleteMe, and run .install.py –clang-completer

Ubuntu

Sometime, you need to install powerline font to make airline show properly. sudo apt-get install fonts-powerline
for Fedora,
sudo dnf install powerline-fonts

if you don't want to start YCM add, below to your .vimrc
let g:loaded_youcompleteme = 1

How to install new version nvim? You have to install new version, otherwise, hop plugin doesn't work.

```
sudo apt-get install python-dev python-pip python3-dev python3-pip

$ sudo add-apt-repository ppa:neovim-ppa/unstable
Update and install

$ sudo apt-get update
$ sudo apt-get install neovim
```

4.5 Plug in

4.5.1 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.

4.5.2 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
```

4.5.3 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**

4.5.4 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   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\+if\)\@!\>,'
```

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

4.5.5 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 will add word1 word2 word3
- 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><p1>". 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.

4.5.6 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>,
- 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 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>".

4.6 Programmer tips

4.6.1 Basic

- Build a IDE-like Vim, You need below plugins:
  1. File explore: NerdTree and CtrlP. In NerdTree, you can use mark functions more for some files you keep working, such as documents. In CtrlP, you can use MRU function more.
  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-^> 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. It's only for current open files. If you want to use Ctrl+], you need to use ctags command to produce tags file. You can see in ctags section.

6. If you want to search withing specific directory, you can use easyGrep 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.

4.6.2 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. I map it to F9 now.

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

4.6.3 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 texdraw
 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 install YCM, for cpp file, :Errors will not work anymore. When you reach error line, error message will be shown in the bottom automatically.

4.6.4 Code navigation

- gd and gD are just text search base methods. 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 direction, 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, use F2 and F3 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**
- **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 is not a general-purpose tool for managing tags files. It only creates the tags it needs on-the-fly in-memory without creating any files.
- 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 ctags 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.

ctags

- Need to produce a /.ctags file as below:

```
--exclude=.git
--exclude=.svn
--exclude=*.js
--exclude=*.html
--exclude=*.css
--exclude=*.swp
--exclude=*.bak
--exclude=*.pyc
--exclude=*.class
--exclude=*.sln
--exclude=*.csproj
--exclude=*.csproj.user
--exclude=*.cache
--exclude=*.dll
--exclude=*.pdb  !!! not write as --exclude=/*.pdb
```

- go to your project, then run below command. –**exclude must follow whole directory**

```
ctags -R --exclude=@/home/yzhao/.ctags --c++-kinds=+p+l+x+c+d+e+f+g+m+n+s+t+u
c       classes)
d       (macro definitions)
e       (enumerators)
f       (function definitions)
g       (enumeration names)
l       (local variables)
m       (class, struct, and union members)
n       (namespaces)
p       (function prototypes) default no extract
s       (structure names)
t       (typedefs)
u       (union names)
v       (variable definitions)
x       (external and forward variable declarations) default no extract
```

- use Ctrl+] to jump.
- Default we don't extract local variable, but you can use /gd to jump to there.

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' -o -name '*.h' > 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)easygrep
<C-o>, <C-i>	jumt back	

- If you don't want to jump [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. Use captive V, it will only match the whole word.
- These command which don't support semantic. They 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

4.6.5 Other Edit commands

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.**
- shift+F2 sometimes doesn't work, I change it to F3.

4.6.6 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

4.6.7 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.

4.6.8 AutoComplete

YoucompleteMe

- In order to make YCM "understand your code, you need to add YCM_extra_conf.py. You can use YCM generator to produce it automatically. For docker compiling, you can download one and add your own configuration. The most import item is add -I flag. Open a file, then you can use :YCMDegubInfo to show the compiler options. clangd will use these compiler options "understand" your code and generate some useful informtion.
- If you didn't provice correct -I flag in YCM_extra_conf.py, Vim will pop up some error message, then you can add more -I options in YCM_extra_conf.py to correct it.
- **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 clangd. In fedora, you can use below commands.

```
sudo dnf install clang-tools-extra
which clangd
clangd --version
sudo dnf install python3-devel
cd .vim/bundle/YouCompleteMe/
./install.py  --clangd-completer
python3 ./install.py  --clangd-completer
```

- 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-j>'
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 desrciption 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. Right now, I close the preview windows in vimrc.
- 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**
- You need to add some include path directory into .ycm_extra_conf.py file.`g++ -E -x c++ - -v < /dev/` 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
```

- If there are syntactic errors, \gg command maybe doesn't work.
- If there are head include errors, it maybe hide the others error. **You need to pay attention to these two things**

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. For example, in latex, type item, then type Ctrl+a list all options, or type Ctrl+j to insert automatically.
- 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.
- In C++, you can use **Some common use pattern are: main, fun, for inc ndef def, if ife el elif.**
- In latex, you can use **item, enum, center, desc, lst and fig**

incc def #if	include
main	main
fun	fun
td st	typedef struct
vector set map	•
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 autocomplete 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

- **If auto insert text for you, then you can ctrl+p(A just reverse selection) to give up the selection and return back the original input, then input more character to refersh options list**.
- 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, <C+a> list all options
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+?>. ? can be N,P,I,D etc.
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. <C-x><C-]> will insert word in tags files.
9. 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 5

muduo net library

- There are two good project, and I copy them to my github.
- five exposed classes: Buffer, EventLoop, TcpConnect, TcpClient, TcpServer.
- Three and half events: 1) accept and connect 2) close and shutdown (server side) and read return 0,(client side)
- message arrive(That is the most important)
- message sent over(that is haft)
- use VSCode, add

```
muduo::Logger::setLogLevel(muduo::Logger::TRACE);
```

  in main. Then use terminal in VSCode to compile and run command.
- One loop(thread) can support multi server. That is all. very simple. all server share the same thread(loop).

```
main()
{
  EventLoop loop;
  ChargenServer c1(&loop, InetAddress(2019);
  c1.start()
  EchoServer e1(&loop, InetAddress(2007);
  e1.start();
  loop.loop();
}
```

- For each server, you need to give two call back functions. onConnection and onMessage. In each ChargenServer, you also need to include TcpServer. In ChargenServer, pass ChargenServer's onConnection and onMessage into TcpServer. Then all main functions are finished in TcpServer.
- Another important component is EventLoop, it includes A poll or Epolle. it just return back a activeChannel then call each channels handleRead call back functions. This channels's handeRead call back come from outside. such as TcpServer
- In the begining, EventLoop has two channels, one is time, the other is wakeup. wakeup for other thread to run something in this poll thread.

```
EventLoop::EventLoop()
  : looping_(false),
    quit_(false),
    callingPendingFunctors_(false),
    threadId_(CurrentThread::tid()),
    poller_(new EPoller(this)),
```

```
    timerQueue_(new TimerQueue(this)),  //ONe is timeQueue
    wakeupFd_(createEventfd()), The another is wakeUpFd.
    wakeupChannel_(new Channel(this, wakeupFd_))
{
  LOG_TRACE << "EventLoop created " << this << " in thread " << threadId_;
  if (t_loopInThisThread)
  {
    LOG_FATAL << "Another EventLoop " << t_loopInThisThread
              << " exists in this thread " << threadId_;
  }
  else
  {
    t_loopInThisThread = this;
  }
  wakeupChannel_->setReadCallback(
      boost::bind(&EventLoop::handleRead, this));
  // we are always reading the wakeupfd
  LOG_TRACE<<"EventLoop::EventLoop()";
  wakeupChannel_->enableReading();
}
```

- You can think timeque is TcpConnection, because it has channel and fd inside it. But it's not build from acceptor.
- Acceptor generate a new TCPConnection, Each TcpSever include one Acceptor and init in it's constructor

```
TcpServer::TcpServer(EventLoop* loop, const InetAddress& listenAddr)
  : loop_(CHECK_NOTNULL(loop)),
    name_(listenAddr.toHostPort()),
    acceptor_(new Acceptor(loop, listenAddr)), //Here.
    threadPool_(new EventLoopThreadPool(loop)),
    started_(false),
    nextConnId_(1)
{
  LOG_INFO << "TcpServier ctor"<<"name_"<<name_;
  acceptor_->setNewConnectionCallback(
      boost::bind(&TcpServer::newConnection, this, _1, _2));
}

Acceptor::Acceptor(EventLoop* loop, const InetAddress& listenAddr)
  : loop_(loop),
    acceptSocket_(sockets::createNonblockingOrDie()),
    acceptChannel_(loop, acceptSocket_.fd()),
    listenning_(false)
{
  acceptSocket_.setReuseAddr(true);
  acceptSocket_.bindAddress(listenAddr);
  acceptChannel_.setReadCallback(
      boost::bind(&Acceptor::handleRead, this));
}
```

- Next, I will introduce TcpServer. Each TcpServer has two channels.

- below is important note: Eventloop only communicate with Poller, more preciously, talk with channels. So you have to to "register" some kind of channels into Eventloop. at the same time.

when something happen in epoll, Eventloop will call back all you logic outside channles. For example, Acceptor generate acceptChannel, then call acceptChannel_.enableReading to update

```
1)enableReading() { events_ |= kReadEvent; update(); }

2)void Channel::update()
{
  loop_->updateChannel(this);
}

3)
void EventLoop::updateChannel(Channel* channel)
{
  assert(channel->ownerLoop() == this);
  assertInLoopThread();
  poller_->updateChannel(channel);
}

4)
void EPollPoller::updateChannel(Channel* channel)
{
  Poller::assertInLoopThread();
  const int index = channel->index();
  LOG_TRACE << "fd = " << channel->fd()
    << " events = " << channel->events() << " index = " << index;
  if (index == kNew || index == kDeleted)
  {
    // a new one, add with EPO
    ......
```

- TcpSever generate one Acceptor, then also pass in newConnection into Acceptor, Acceptor pass this function into it's channel. Once acceptor listion, it will call back this funciton. This funciton will generate one new TcpConnection. One new TcpConnection will also generate a new Channel and call enableReading() to "registor" its fd into poll. But When you greate TcpConnection, you pass TcpServer's OnMessage and OnConnection as it's call back. That's all. Why do we need pass TcpServer's member function as Acceptor's channel call back. Because this call back will generate new TcpConnection, and this new TcpConnection should be put into TcpServer to manage.

```
void TcpServer::newConnection(int sockfd, const InetAddress& peerAddr)
{
  loop_->assertInLoopThread();
  char buf[32];
  snprintf(buf, sizeof buf, "#%d", nextConnId_);
  ++nextConnI
```

Chapter 6

Other Tools

6.1 OS and phone

6.1.1 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.pdf` 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/local/share`to 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)	?

6.1.2 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.

6.1.3 mac

- command is window key if you use a windows keyboard. Ctrl is not used very often in Mac, but command(windows key) is used a lot. such as: command+c and command +v. You need to get used to it.
- 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
```

- common used shortcut for mac. For app, windows and tab three big components. you need to know how to open, switch and close it. It gives you a lot of convenience.

short cut	function
cmd+space, cmd + tab, cmd+Q	new app, switch app, close app
cmd+n, cmd+ , cmd+w(shift)	new windows, switch windows, close windows
f3, ctrl+down	show all apps, show all windows
cmd+T, cmd +num(cmd+tab), cmd+w	new tab, switch tab, close tab
command+option+esc	force app quit(you need to command+tab switch)
cmd +space, ctrl+space, alt+space	spotlight, input, iterm2
cmd+shift +3,4	screen shot, save to desktop
fn +f11	show desktop
cmd+h	hide windows.
cmd+ctrl+q	lock .
cmd+ctrl+(L,C,R)	move windows(customized in keyboard short-cut)

- how to split windows? hover you mouse on the green button.
- check json file with format
  `https://jsoneditoronline.org/`
- fix "Double Commander" can't be opened because Apple cannot check it for malicious software. In the Finder on your Mac, locate the app you want to open. Don't use Launchpad to locate the app. Control-click the app icon, then choose Open from the shortcut menu. Click Open.
- two fingers click is right click.

6.1.4 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

6.1.5 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. ctrl+num to navigate the tabs. and ctrl+click to open a link in a new tab. ctrl+T open a new tab. ctrl+w to close the current tab.