NOTES LIST

notes of lecture 1 Course Overview & the shell

- `echo` only print para following,treat it like string
- `cat` link file and redirection output stream

  ```
  # ther is a file named code which contains the shell script
  echo code # the script won't execute
  code # this commond will just print "code"
  cat code # the script will execute and print results on terminal
  # (code results here)
  ```

- `sudo` do as 'super user' or 'root'
  - `sudo su`change to a root user & `exit`logout root user and change to an ordinary user
- blankspace is a delimiter of commands and arguments

notes of lecture 2 Shell Tools and Scripting

- the difference between single quotion and double quotion:Strings delimited with `'` are literal strings and will not substitute variable values whereas `"` delimited strings will. eg:

```
foo=bar
echo "$foo"
# prints bar (substitude variable values)
echo '$foo'
# prints $foo (just literal string)
```

- command substitution $(cmd) and process substitution <(cmd)
- globbings(通配符):`*` and `?`.Notice their difference.One expands to any characters,the other just expands to single one character but null.
- introduced function and it's arguments $0,$1~$9,$$ and so on...
- exit code:the same as before:`true` means 1,`false` means 0;but constrast with previous knowledge,when it is ok or true in an **expression** the return code is 0;if not,it is 1. eg:

```
false || echo "Oops, fail"
# Oops, fail
true || echo "Will not be printed"
#
```

- find递归与不递归:find命令是默认递归遍历文件夹的

```
find . -name “*.txt”
# 当前路径下递归查找以.txt结尾的文件夹
find . -name “*.txt” -maxdepth 1
# 当前路径下不递归查找以.txt结尾的文件夹
```

notes of lecture 3 Editors (Vim)

- mainly introduced how to use vim which i'v touched befor,so recode something useful in the following notes.

lectur 4:data wrangle

some commands useful :

1. **sed**(stream edit):

    1. regular expression:

        1. **.** any charater
        2. ***** 0 or more of the preceding match(匹配之前 *preceding* 的字符零次或者多次匹配)
        3. **+** 1 or more of the preceding match
        4. **?** 0 or 1 of the preceding match(which can be use in Non-greedy matching 'cause **.*** or **.+** is always greedy matching)
        5. **[abc]** any one character of a, b, and c
        6. **(RX1|RX2)** either something that matches RX1 or RX2
        7. **^** the start of the line
        8. **$** the end of the line

    2. capture goup:

        1. (*patterns*) reference :

        ```
        sed -E 's/.*Disconnected from (invalid |authenticating )?user (.*) [0-9.]+ port [0-9]+( \[preauth\])?$/\2/'
         # -E using special meanings whit out escape
         # s/arg1/arg2 if input stream matches arg1 pattern substitude it with arg2
        \1 # referencing the first group
        \4 # referencing the forth group
        ```

2. **sort**:

    1. sort accoding to what?

3. **uniq**(unique):

    1. -c count and delete repeated lines

4. **wc**(word count):

    1. -l count by line
    2. -n sort by numbers
    3. -k select a column seperated by whitesapce in the input stream.following *number1*.*number2* meaning sorting starts at the *column* number1 to column *number2*

5. **paste**(paste input stream into a line):

    1. -sd, change the delimiter (defualt by whitespace,in this situation it is seperated by ,)

6. **awk**(operating columns):

    1. show one exaple :

```
awk '$1==1 && $2 ~ /^c.*e$/ {print $0}' # mind the form of arguments
 # When the first column equals 1 and the second column matches the
pattern,then print the whole line
```

7. **bc**(berckley calculator):

    1. combine with *paste*

8. **xargs**(takes line of input and turns them into arguments)

9. **tr**(translate):delete or change

    1. single arguments is the function of delete
    2. two means change the argumnets1 in STDIN with arguments2

lectur 5:commandline environment

1. job control: 1.signal 2.process manage
2. Terminal multiplexer:
    1. hierachical structure:
        1. sessions
            1. windows
                1. panes