Lecture 2 Shell Tools and Scripting

1. 注意space的使用(变量命名) [[ ]]需要使用空格间隔以识别变量
2. source fileName <=>. fileName 当前环境下读取并执行fileName中的命令,区别与sh fileName

, 并不建立新的shell执行fileName, 可以通过变量测试两种方式的不同；

1. $_ 变量保存的是上一次bash操作的"$1"(自己试的)
2. !! -> 前一次执行的命令
3.

$?	**显示最后命令的退出状态，0表示没有错误，其他表示有错误$? 是显示最后命令的退出状态，0表示没有错误，1表示有错，其他表示有错误**
$#	脚本参数个数
$@	传给脚本参数的参数列表
$!	Shell最后运行的后台Process的PID(后台运行的最后一个进程的进程ID号)
$$	程序的pid

4. || 或，从左到右，成立则停止；&& 从左到右，有不成立则停止；`;`分段执行
5. foo=$(pwd) 重命名

```
#!/bin/bash
echo "$(date)时执行程序"
echo "运行程序$0,带有$#个参数，进程号$$"
for file in "$@"; do
#从参数中查找foobar,将标准输出和错误输出重定向到/dev/null中
    grep foobar "$file" > ./null 2>./null
    if [["$?" -ne 0]]; then
        echo "File $file does not have any foobar, adding one"
        echo "# foobar" >> "$file"
    fi
done
```

> file

7.

字符	**作用**
-eq	等于 equal
-ne	不等于 not equal
-gt	大于 （greater ）
-lt	小于 （less）
-ge	大于等于 greater & equal
-le	小于等于 less & equal

8. [[ ]]命令和[]之间必须有空格，[] 等同于test,是表达式； { } 限定变量名称范围，内部的程序在当前的shell中执行， () 在另外的shell中执行，括号内的变量不会作用与当前的shell，$(ls) 会在另外的shell执行command，但输出放到本输出中；
9. convert 转变图片格式
10. { } 正则表达式的扩展功能，{a..z}意味着a,b,...,z {a,z} <=> a z
11. diff <(ls foo) <(ls bar)

```
#!/usr/local/bin/python
# or #!/usr/bin/env python
import sys
for arg in reversed(sys.argv[1:]):
    print(arg)
```

13. shellcheck *.sh 检测bash问题
14. mv -i 覆盖前提示 rg 递归搜索当前目录，寻找匹配模式的行
15. 小工具 tldr <=> `Too long; didn't read` TLDR 页的 GitHub 仓库将其描述为简化的、社区驱动的手册页集合。在实际示例的帮助下，努力让使用手册页的体验变得更简单。https://github.com/raylee/tldr
16. ffmpeg 从视频中截取图片，处理图片的工具
17. find . -name src -type d find . -path "*/test/*.py" -type f find . -mtime -1 find . -name "*.tmp" -exec rm {} \;
18. fd 一个简约版的find, fd ".*py"
19. grep -R 在当前目录的所有中递归寻找
20. ripgrep 全文搜索 rg

Lecture 3 Editors(vim)

1. normal insert visual mode

Lecture 4 Data Wrangling

1. pipe | 将一个进程的数据块送给另一个； 在方括号中的字符集不关心顺序。
2. sed stream editor

   Replace the first occurrence of a regular expression in each line of a file, and print the result:
   sed 's/regex/replace/' filename sed 's/.*Disconnected from//'

   `sed 's/[]/replace/g' #迭代递归更接近与正则表达式[]的含义`
3. wc Count lines, words, or bytes. -l --lines -w --words -c --characters(bytes)
4. sort Sort lines of text files.
5. uniq 配合sort使用，输出不重复的行。Display number of occurrences of each line along with that line:
   sort file | uniq -c
6. gnuplot A graph plotter that outputs in several formats.
7. xargs. Execute a command with piped arguments coming from another command, a file, etc. The input is treated as a single block of text and split into separate pieces on spaces, tabs, newlines and end-of-file.
8. rustup Rust toolchain installer.
   Install, manage, and update Rust toolchains.

9. ffmpeg 视频处理

Quickly extract a single frame from a video at time mm:ss and save it as a 128x128 resolution image:
ffmpeg -ss mm:ss -i video.mp4 -frames 1 -s 128x128 -f image2 image.png

10. feh Lightweight image viewing utility. View images locally or using a URL:
    feh path/to/images

Lecture 5 Command-line environment

Job control

1. sleep 等待n s h m时间
2. siginal sigint例子 [todo] SIGINT HUP ...

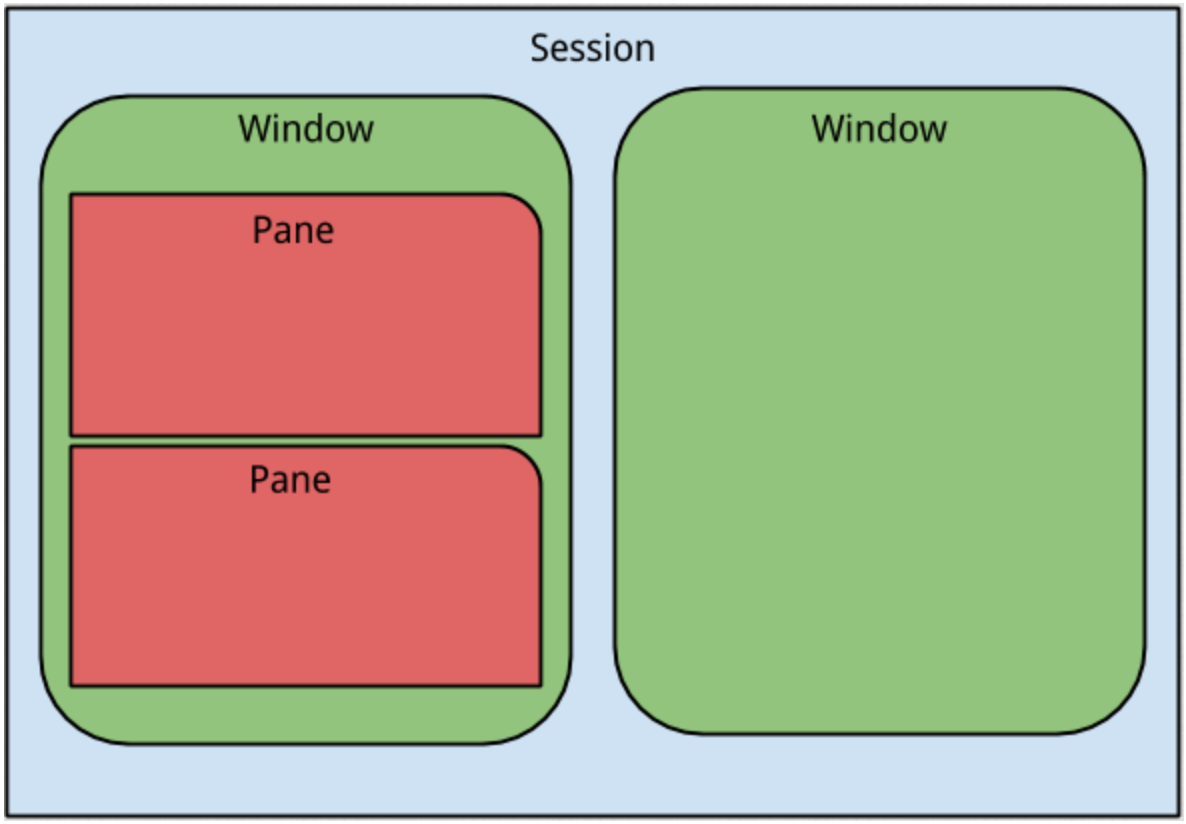
```
#!/usr/bin/env python
import signal, time

def handler(signum, time):
    print("\nI got a SIGINT, but I am not stopping")
signal.signal(signal.SIGINT, handler)
i=0
while True:
    time.sleep(.1)
    print("\r{}".format(i), end="")
    i += 1
```

3. nohup no hang up 不挂断运行，并非后台运行， & 后台运行
4. jobs 查看当前终端后台执行任务，ps查看所有终端的瞬时进程。
5. bg <=> Resumes jobs that have been suspended (e.g. using `Ctrl + Z`), and keeps them running in the background; fg <=> Run jobs in foreground.
    - Bring most recently suspended background job to foreground:
      fg
    - Bring a specific job to foreground:
      fg job_id
6. `%1` <=> A single % (with no
   Simply naming a job can be used to bring it into the foreground: %1 is a synonym for `''fg %1''`, bringing job 1 from the background into the foreground. Similarly, `"%1 &''` resumes job 1 in the background, equivalent to ``bg %1''.

Terminal

1. sessions, windows and panes; tmux 管理会话，新建pane, `Ctrl` + `b` 进入快捷键模式



C-b Send the prefix key (C-b) through to the application.
C-o Rotate the panes in the current window forwards.
C-z Suspend the tmux client.
! Break the current pane out of the window.
" Split the current pane into two, top and bottom.

List all paste buffers.

$ Rename the current session.

DotFiles .file

1. alias <=> typedef 将一个程序块重命名 alias gs="git status"
2. PS1 变量
3. github mathiasbynens/dotfiles

Remote Machine

1. ssh host@ip
2. ssh-keygen
3. scp
4. rsync
5. .ssh/config

Lecture 6 git

1. .git 包括 branches/ config description HEAD hooks/ info/ objects/info,pack refs/heads,tags
2. git cat-file -p "hash"
3. git snapshot
4. git checkout
5. git merge 配合 git log --graph
6. git fetch
7. git config <=> .gitconfig

Lecture 7 Debugging and Profiling(量化描述)

Debugging

1. logger && log show
2. `python -m ipdb <name>.py` `ipdb` pip install ipdb ipdb Python的调试器，可以在Python代码内部插入断点，查看变量等。flag -m module-name Searches sys.path for the named module and runs the corresponding .py file as a script. ipdb l <=> list; s <=> step, c <=> continue, p 'value' <=> print special value, {locals()}; q <=> quit, b <=> breakpoint
3. gdb
   - 调试未执行程序，命令行

```
# 无参数程序
gdb program
# 进入后输入
run(r)
# 有参数程序
gdb program
#输入参数
run arg1 arg2 ...
```

   - 调试运行中的程序，需要预先查看进程号(PID), 可以用`ps` or `pidof`

```
gdb
attach pid
# or 直接进入多线程模式的进程
gdb -p pid
# 显示当前进程中的所有线程
info thread
# 启动特定线程调试 tid thread identity
thread tid
# '-O2 -c foo.c'作为gcc的
gdb --args gcc -O2 -c foo.c
```

   - 应用发生core dump，即核心转储时的调试。这个时候内核会将应用程序在崩溃发生时的内存数据、程序调用堆栈等核心信息转存到磁盘。

```
# core dump 是程序异常退出时的内存快照，是异常发生后对程序进行现场还原和故障排查的关键线索
ulimit -c # 查看和指定core文件的大小
# 对某进程产生的core文件/data/core/xxx 分析调试
gdb program/data/core/xxx
# or
gdb -c /data/core/xxx
# backtrace (bt)显示程序异常退出时刻的函数堆栈情况
# frame, print, up, down, where等命令
```

   - 断点设置和查看

命令	解释
break	执行下一条指令时由GDB暂停程序
break +offset_num	再向下执行 offset-num 的行号
break -offset_num	向下执行到指定的行号

- 观察点设置和查看

命令	形式
rwatch var	写观察点
awatch var	读观察点
watch var	

4. strace

```
import time

def foo():
    return 42
for foo in range(5):
    print(foo)
bar = 1
bar *= 0.2
time.sleep(60)
print(baz)
```

6. pyflakes
7. mypy
8. writegood 检查书写错误

Profiling

2. Python -m cProfile 性能分析模块
3. tac Print and concatenate files in reverse (last line first).
4. kernprof
5. memory_profiler, python module
6. sudo perf stat, record, report 记录CPU系统资源使用
7. Flame Graph
8. htop 显示当前的系统资源调用
9. du, ->Disk usage: estimate and summarize file and directory space usage.Disk usage: estimate and summarize file and directory space usage.
10. lsof, -> Lists open files and the corresponding processes.
11. hyperfine, ->

Lecture 8 Metaprogramming

1. make Makefile.txt

Makefiles

rules

target ... : prerequisites ... 目标：提前条件... 目标可以是执行或者对象文件，也可以是执行的操作如 clean recipe prerequisites 是生成目标的输入文件，recipe是 make执行的操作, recipe前需要一个 ... tab区分

...

...

```
# or like follow method
targets : prerequisites ; recipe
    recipe
    ...
```

```
edit : main.o kbd.o commaedit : main.o kbd.o command.o display.o \
               insert.o search.o files.o utils.o
        cc -o edit main.o kbd.o command.o display.o \
                   insert.o search.o files.o utils.o
```

使用变量替代一长串的内容

```
object = main.o kbd.o command.o display.o \
          insert.o search.o files.o utils.o
```

另一种组织方式，将object文件放在一个

```
#example
objects = main.o kbd.o command.o display.o \
          insert.o search.o files.o utils.o
edit : $(objects)
    cc -o edit $(objects)
$(objects) : defs.h
kdb.o command.o files.o : command.h
display.o insert.o search.o files.o : buffer.h
```

clean 的方法

```
clean :
    rm edit $(objects)

# 更实用的方式 防止和一个实际名为clean的文件使得编译器混乱，
.PHONY : clean
clean :
    -rm edit $(objects)
```

make如何处理Makefile

- 默认从第一个target file开始，可以通过修改变量`.DEFAULT_GOAL`执行默认开始处理的target
- 解析规则前，需要先处理prerequisites,这个过程迭代，到prerequisites不必再解析其他文件。
- 没有依赖项(prerequisites)的规则先不进行处理，除非命令行指定，如 **make clean**.

- 当一个c源文件单独生成对应的.o文件时，可以省略.c的书写，默认对应更新，也就是说 main.o : defs.h main.c \n\t cc -c main.o 可以简写为 main.o : def.h，同理，command.o : defs.h command.h command.c \n\t cc -c command.o 省略为 command.o: defs.h command.h

makefiles包含五种：显示规则、隐式规则、变量定义、评论和指令

directives

- 包含其他的makefile, 文件名可以以bash方式扩展，第一个不应该包含\t(tab)

```
include filenames
# example
include foo *.mk $(bar)
# <=>
include foo a.mk b.mk c.mk bish bash
# 其中*.mk 匹配a.mk b.mk c.mk 变量bar="bish bash"
```

头文件寻找路径，首先在以**-I or --include-dir**的路径下寻找，找不到则再从*prefix/include* <=> /usr/local/include, /usr/gnu/include, /usr/local/include, /usr/include下寻找。

忽略一个makefile文件，由于不需要重新编译或者不存在

```
# replace include with -include
-include filenames
```

- 变量 MAKEFILES

如果变量MAKEFILES定义了，则这个变量被解读为一些其他的makefiles，并且在其他include filename前被加载，这个变量仅有当前定义起作用，不能从其他的makefile中读取并加载

- % 匹配任意target
- make 解读Makefile

  两段解读

> a. 加载所有的makefile文件，将变量内在化并赋值，解读所有的显式和隐式规则，根据所有的>target和prerequisites形成一张关系表
> b. 根据第一阶段形成的表，判断哪个target需要更新然后运行规则更新target

- **Makefile**解析方式

a. 读取所有的逻辑行，包括带有 "****" 的行

b.移除评论行

c. 如果读取行是以 规则前置符开头的(默认是/t,tab)，则将该行加入当前的recipe中，并读取下一行。

d. 扩展immediate expand context行中的元素

e. 扫描行中的分隔符，像 ':' '=',判断是宏定义还是规则式

- Pattern of Makefile

character	meaning
::	[todo]
$@	the target file (target: ...)
$<	the source file (such *.c)

Lecture 9 Security and Cryptography 密码术

####

1. Entropy 熵 log2
2. Hash function 非数字性的，同样的数据，运算后结果不同；冲突抵抗性
3. sha1sum <file or line>
4. Key Derivation Function
5. Symmetric key cryptography, -
    - keygen() -> key
    - encrypt(plaintext, key) -> ciphertext
    - decrypt(ciphertext,key) -> plaintext
6. Asymmetric key cryptography
    - keygen() -> (public key, private key)
    - encrypt(p,public key)-> c
    - decrypt(c,private key) ->p

Lecture 10 Potpourri 集锦

1. keyboard, remap keys,

key	remapped	reason
Caps Lock	Ctrl/Escape	很少用
...	...	...

2. daemons 守护进程 systemd(the system daemon) systemctl status列出当前运行中的 daemons，enable, disable, start, stop, restart
3. fuse
4. Backups 在同一块磁盘上的复制不是一个备份，装置外的备份才有实际意义。
5. APIs 一般放在 api.service.com 例如 api.weather.gov/, IFTTT工具网站
6. Windows 管理工具 `tiling`, pane(框)管理工具`tmux`
7. VPNs be all the rage 十分流行
8. Hammer 锤子 Hammerspoon 锤子
9. VMs, Vagrant 代码描述机器配置信息，然后用vagrant up 加载到虚拟机中， Docker容器技术。

Lecture 11 Q&A

1. useful data wrangling tools —— jq or pup which are specialized parsers for JSON and HTML data respectively. The Perl programming language is another good tool for more advanced data wrangling pipelines. Another trick is the `column -t` command that can be used to convert whitespace text (not necessarily aligned) into properly column aligned text.
2. vim
    - Marks - In vim, you can set a mark doing m for some letter X. You can then go back to that mark doing '. This let's you quickly navigate to specific locations within a file or even across files. 行首标记 s (start)：ms，行尾标记 e (end)：me，:marks 列出所有标记，:' 标记行首 :`标记光标位置
    - Navigation - Ctrl+O and Ctrl+I move you backward and forward respectively through your recently visited locations.
    - Undo with time - The :earlier and `:later` commands will let you navigate the files using time references instead of one change at a time.