

Missing Semester

Lecture 1 Course Overview + The Shell

环境变量

```
[jon@xpanse missing-semester]$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/bin:/bin:/opt/intel/vtune_an  
/bin64:/home/jon/bin:/home/jon/.cargo/bin:/home/jon/.npm-global.  
/2.6.0/bin:/home/jon/.cargo-target/release:/usr/lib/jvm/default  
:/usr/bin/vendor_perl:/usr/bin/core_perl:/usr/lib/jvm/default/b
```

cd - 回到之前一次目录

```
[jon@xpanse missing-semester]$ cd -  
/  
[jon@xpanse /]$ cd -  
/home/jon/dev/pdos/classes/missing-semester
```

--help

```
[jon@xpanse missing-semester]$ ls --help  
Usage: ls [OPTION]... [FILE]...  
List information about the FILEs (the current directory by default).  
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.  
  
Mandatory arguments to long options are mandatory for short options too.  
-a, --all  
-A, --almost-all  
--author  
-b, --escape  
--block-size=SIZE  
-B, --ignore-backups  
-c  
-C  
--color[=WHEN]  
-d, --directory  
[01 0:1* 1:0- 2:0.5 3:missing]
```

man + 命令

```
do not list implied entries ending with ~

-c      with -lt: sort by, and show, ctime (time of last modification of file
        status information); with -l: show ctime and sort by name; otherwise:
        sort by ctime, newest first

-C      list entries by columns

--color[=WHEN]
        colorize the output; WHEN can be 'always' (default if omitted),
        'auto', or 'never'; more info below

-d, --directory
        list directories themselves, not their contents

-D, --dired
```



ctrl-L = clear

```
[jon@xpanse missing-semester]$
```

重定向

```
[jon@xpanse missing-semester]$ < file > file^C
[jon@xpanse missing-semester]$ echo hello > hello.txt
[jon@xpanse missing-semester]$ cat hello.txt
hello
[jon@xpanse missing-semester]$ cat < hello.txt
hello
[jon@xpanse missing-semester]$ cat < hello.txt > hello2.txt
[jon@xpanse missing-semester]$ cat hello2.txt
hello
```

>> 追加

```
[jon@xpanse missing-semester]$ cat < hello.txt >> hello2.txt
[jon@xpanse missing-semester]$ cat hello2.txt
hello
hello
[jon@xpanse missing-semester]$
```

管道 (注意 -n 1 表示最后一行)

```
[jon@xpanse missing-semester]$ ls -l / | tail -n1
drwxr-xr-x  1 root root  116 Jan 13 10:57 var
```

超级用户

```
[jon@xpanse intel_backlight]$ echo 500 > brightness
bash: brightness: Permission denied
[jon@xpanse intel_backlight]$ sudo echo 500 > brightness
bash: brightness: Permission denied
[jon@xpanse intel_backlight]$ #^Ccho 1 > /sys/net/ipv4_forward
[jon@xpanse intel_backlight]$ sudo su
[sudo] password for jon:
[root@xpanse intel_backlight]# echo 500 > brightness
[root@xpanse intel_backlight]# exit
[jon@xpanse intel_backlight]$ echo 1060 | sudo tee brightness
1060
```

注意：上图中一开始 `sudo` + 使用重定向失败，是因为后面的打开文件还是用的普通用户。解决法一：整个命令前加 `#`，表示整个命令都是超级用户；法二：`sudo su`；法三：管道后面加 `sudo`，注意 `tee` 命令是写入文件和打印在屏幕上

打开文件

Linux : `xdg-open` ; Mac : `open` (使用适合的软件来打开)

Lecture 2 Shell工具和脚本

变量

```
~/m/tools >>> foo=bar
~/m/tools >>> echo $foo
bar
~/m/tools >>> foo = bar
zsh: command not found: foo
~/m/tools >>> █
```

注意空格

字符串

```
~/m/tools >>> echo "Hello"
Hello
~/m/tools >>> echo 'World'
World
~/m/tools >>> echo "Value is $foo"
Value is bar
~/m/tools >>> echo 'Value is $foo'
Value is $foo
```

单引号和双引号的区别在于对变量的效果

函数 .sh 文件

```
1 mcd () {  
2     mkdir -p "$1"  
3     cd "$1"  
4 }
```

\$1 和 argv[1] 相同

```
~/m/tools >>> vim mcd.sh  
~/m/tools >>> source mcd.sh  
~/m/tools >>> mcd test  
~/m/t/test >>> 
```

source 定义了 mcd 函数，然后就可以使用这个函数

\$

\$? 获取上条命令的错误代码（返回值）

\$_ 获取上条命令的最后一个参数

```
~/m/tools >>> rmdir test  
~/m/tools >>> mkdir test  
~/m/tools >>> cd $_  
~/m/t/test >>> 
```

!!

重新执行上一条

```
~/m/tools >>> mkdir /mnt/new  
mkdir: /mnt/new: Permission denied  
~/m/tools >>> sudo mkdir /mnt/new █
```

!! 会被你刚刚尝试的命令取代

标准错误流？

```
~/m/tools >>> echo "Hello"  
Hello  
~/m/tools >>> echo $?  
0  
~/m/tools >>> grep foobar mcd.sh  
~/m/tools >>> echo $?  
1  
~/m/tools >>> █
```

搜索 foobar 字符串，而它不存在

0表示正确，1表示错误。

```
~/m/tools >>> echo $?
1
~/m/tools >>> true
~/m/tools >>> echo $?
0
~/m/tools >>> false
~/m/tools >>> echo $?
1
~/m/tools >>> false || echo "Oops fail"
Oops fail
~/m/tools >>> true || echo "Will be not be printed"
~/m/tools >>> █
```

上图中 `true` 和 `false` 表示判断，相当于目前标准错误流中的返回值 `== true`。另外逻辑运算采用短路。

```
~/m/tools >>> false ; echo "This will always print"
This will always print
~/m/tools >>> █
```

它就会始终被打印出来

使用分号始终可以成功

变量进阶

```
~/m/tools >>> foo=$(pwd)
~/m/tools >>> echo $foo
/Users/josejavier/missing-semester/tools
~/m/tools >>> █
```

```
~/m/tools >>> echo "We are in $(pwd)"
We are in /Users/josejavier/missing-semester/tools
~/m/tools >>> █
```

过程替换

```
~/m/tools >>> cat <(ls) <(ls ..) █
```

括号内的指令输出会存到临时文件，上述指令可以打印本目录和上一目录

例子

```
1 #!/bin/bash
2
3 echo "Starting program at $(date)" # Date will be substituted
4
5 echo "Running program $0 with $# arguments with pid $$"
6
7 for file in "$@"; do
8     grep foobar "$file" > /dev/null 2> /dev/null
9     # When pattern is not found, grep has exit status 1
10    # We redirect STDOUT and STDERR to a null register since we do not care
11    # about them
12    if [[ "$?" -ne 0 ]]; then
13        echo "File $file does not have any foobar, adding one"
14        echo "# foobar" >> "$file"
15    fi
16 done
```



`#$` 表示给定的参数个数, `$$` 是命令的进程PID, `$@` 展开成所有参数

重定向到 `/dev/null` 的内容都会被丢弃; `2>` 表示重定向标准错误流

`-ne` 比较选项, `Not Equal`

通配

```
example.sh image.png mcd.sh  
~/m/tools >>> ls *.sh  
example.sh mcd.sh  
~/m/tools >>> mkdir project42  
~/m/tools >>> ls project?
```

`*` 任意多字符, `?` 单个字符

`{}`

```
~/m/tools >>> convert image.png image.jpg  
~/m/tools >>> convert image.{png,jpg}
```

```
~/m/tools >>> touch foo{,1,2,10}
```

```
~/m/tools >>> touch foo foo1 foo2 foo10
```

```
~/m/tools >>> touch project{1,2}/src/test/test{1,2,3}.py
```

`..`

```
~/m/tools >>> touch {foo,bar}/{a..j}
```

shebang #!

```
1 #!/usr/local/bin/python *这个单词源于这行以 #! 作为开头。# 是 sharp, ! 是 bang
2 import sys
3 for arg in reversed(sys.argv[1:]):
4     print(arg)
5 # foobar
```

script.py

开始时那神奇的一行叫做 shebang[*]

```
~/m/tools >>> python script.py a b c
c
b
a
~/m/tools >>> ./script.py a b c
```

Shell 是用首行识别到

shell通过首行识别出它如何运行

```
1 #!/usr/bin/env python
```

env 会在这个目录寻找后面的指令

shellcheck

```
~/m/tools >>> shellcheck mcd.sh                               master ■  
In mcd.sh line 1:  
mcd () {  
^-- SC2148: Tips depend on target shell and yours is unknown. Add a shebang.
```

给出warning和语法错误

tldr

提供深入浅出的命令示例

寻找文件

```
~/m/tools >>> find . -name src -type d  
. ./project1/src  
. ./project2/src  
~/m/tools >>> █
```

上图意为：在当前目录寻找名为 src 的文件夹

```
~/m/tools >>> find . -name src -type d  
. ./project1/src  
. ./project2/src  
~/m/tools >>> find . -path '**/test/*.py' -type f █  
| ~/m/tools >>> find . -mtime -1 █
```

上图查找最近一天修改过的文件

```
~/m/tools >>> find . -name "*.tmp" -exec rm {} \;  
~/m/tools >>> echo $?  
0  
~/m/tools >>> █
```

但我们有 0 错误代码，就是有事发生

上图寻找到文件后又执行删除。注意有错误代码表示有事发生。

```
~/m/tools >>> find . -name "*tmp"  
~/m/tools >>> fd ".*py"
```

而且默认使用正则表达式[*]

fd 文件更短，且默认使用正则表达式、忽略 gitfile

locate

```
~/m/tools >>> updatedb  
*cron 是 UNIX 下一个基于时间的任务管理系统，可以运行定期任务。
```

通常由 cron 定期执行来更新数据库。[*]

updatedb 更新数据库

grep (升级版 ripgrep 和 ack 和 ag)

```
~/m/tools >>> grep -R foobar .
./example.sh:      grep foobar "$file" > /dev/null 2> /dev/n
./example.sh:          echo "File $file does not have any fo
./example.sh:          echo "# foobar" >> "$file"
./mcd.sh:# foobar
./script.py:# foobar
~/m/tools >>>
```

-R 递归文件夹寻找

```
~/m/tools >>> rg -u --files-without-match "^#\!$" -t sh
```

-u 包括隐藏文件，--files-without-match 显示不符合match规则（为了下述检验），"^#\!" 正则表达式匹配 shebang，-t sh 表示只搜索后缀为 sh 的文件。

```
rg "import requests" -t py -C 5 --stats ~/scratch
```

-c 5 显示上下5行，--stats 显示详细查找数据如下：

```
12 matches
12 matched lines
12 files contained matches
168 files searched
3088 bytes printed
1332182 bytes searched
0.002277 seconds spent searching
0.023811 seconds
```

查找执行过的命令

history

```
~/m/tools >>> history
1458 find . -mtime -1
1460 find . -name "*.tmp" -exec rm {} \;
1461 echo $?
1462 find . -name "*.tmp"
1463 fd ".*py"
1464 locate tmp
1465 locate missing
1466 locate missing-semester
1467 grep foobar mcd.sh
1468 grep -R foobar mcd.sh
1469 grep -R foobar
1470 grep -R foobar .
1471 rg "import requests" -t py ~/scratch
1472 rg "import requests" -t py -C 5 ~/scratch
1473 rg -u --files-without-match "^#\!" -t sh
1474 rg "import requests" -t py -C 5 --stats ~/scratch
```

从头打印，加个1

```
~/m/tools >>> history 1 | grep convert
1 convert foto.jpg -resize 15% foto_small.jpg
2 convert foto.jpg -resize 30% foto_small.jpg
3 convert foto.jpg -resize 50% foto_small.jpg
4 convert foto.jpg -resize 40% foto_small.jpg
5 convert foto.jpg -resize 35% foto_small.jpg
7 convert foto2.jpg -resize 35% foto2_small.jpg
8 convert foto3.jpg -resize 35% foto3_small.jpg
9 convert foto3.jpg -resize 25% foto3_small.jpg
10 convert foto3.jpg -resize 30% foto3_small.jpg
685 convert -resize x64 favicon.png -background transparent favicon.png
688 convert -resize x32 favicon.png -background transparent favicon.png
778 convert
779 convert favicon.png -define icon:auto-resize=64,48,32,16 favicon.png
1279 find . -name '*.png' -exec convert {} {}.jpg \; \n
1280 find . -name '*.png' -exec convert {} "{}.jpg" \; \n
1366 tldr convert
1384 history | grep convert
1450 tldr convert
```

所有匹配上这个子字符串的

配合grep使用

ctrl+R：倒序查找命令

```
~/m/tools >>> history 1 | grep convert
```

bck-i-search: convert_

连续按 ctrl+R 往前找

fzf 实时搜索、模糊匹配

```
~/m/tools >>> cat example.sh | fzf
```

然后我们用管道连到 fzf 上

```
echo "File $file does not have any foobar, adding one"
grep foobar "$file" > /dev/null 2> /dev/null
>     echo "# foobar" >> "$file"
3/15
> foobar
```

然后可以实时地输入要找的字符串

它会绑定 ctrl+R

历史记录子串查找

fish 和 zsh shell 都有支持。右箭头选中之前的命令

列出目录结构

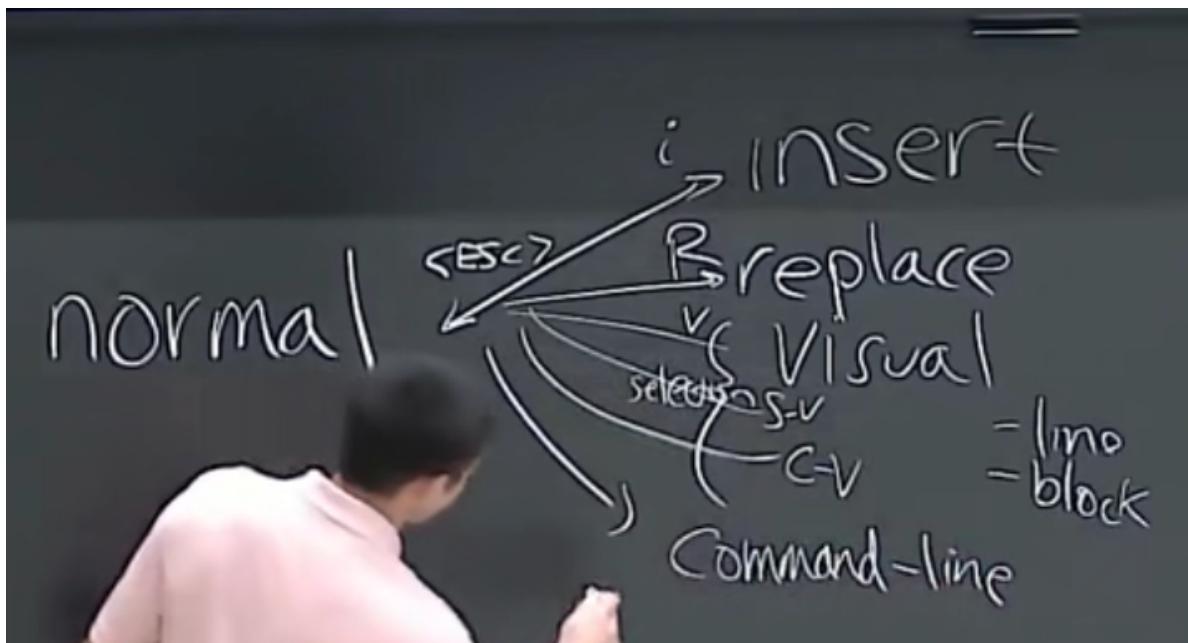
tree broot nnn (Nautilus 和 mac 的访达)

Lecture 3 Vim

ctrl+V的几种写法

```
^V  
Ctrl-V  
<C-V>
```

模式切换



i: insert

r: replace

v: visual

shift+v: visual -line

ctrl+V: visual -block

:是命令行模式

命令行

:quit :q 退出当前window

:qa 退出所有window

:w =write 保存

help 如 help :w, 注意不加: 表示 Normal 状态下的按键

:sp 多窗口

光标在分屏间切换快捷键

- 光标在各个分屏间循环切换

ctrl + w + w

- 光标切换到左侧分屏

ctrl + w + h

- 光标切换到下侧分屏

ctrl + w + j

- 光标切换到上侧分屏

ctrl + w + k

- 光标切换到右侧分屏

ctrl + w + l

`:tabnew` 新的空分页

`ctrl + pagedown/up` 切换分页

关闭与离开窗口

有4种关闭窗口的方式，分别是：离开 (quit) 、关闭 (close) 、隐藏 (hide) 、关闭其他窗口

^代表Ctrl键

`^Wq`, 离开当前窗口

`^Wc`, 关闭当前的窗口

`^Wo`, 关闭当前窗口以外的所有窗口

移动

`hjk1` 移动

`w/b` 向后/前移动一个单词

`e` 移动到单词末尾

`0/$/^` 行首/尾/行首第一个非空白字符

`ctrl + u/d` 上下翻页

G 大写：最后一行 **gg** 第一行

H M L 大写：当前页的上中下页

f* 如 **fo**，查找后面第一个字母 *****，光标跳到它前面（**t** 前两个）

F* 如 **fo**，查找前面第一个字母 *****，光标跳到它前面

编辑

i insert

o open，下面新开一行 **O** 上面新开

d delete，需要配合上述移动按键，如 **dw** 删除一个词

u undo取消 **ctrl+R** redo

c change类似delete，但删完后会进入insert模式

dd 整行删 **cc** 删完进入insert

x 删除字符

r* replace，用*代替此处的字符

~ 改变大小写

a append，追加

复制粘贴

y 复制，**p** 粘贴。需要范围

yy 复制整行

Visual

v 进入 选择之后的内容

V 选择一行文字

ctrl + v 矩形块 但是和粘贴重复

Count

4j 向下4行

7dw 删除7个word

修饰符

a around (包含括号)

i inside

例如 **ci[** 改变方括号内的东西

% 在成对的符号间转移

搜索

/** 反斜杠后面加你要搜索的东西

n 搜搜下一个

其他

. 重复刚刚的命令

Lecture 4 Data Wrangling

less 更人性化显示内容

正则表达式?

Lecture 5 Command-line Environment

Job control

sleep

```
5 #!/usr/bin/env python
4 import signal, time
3
2 def handler(signum, time):
1     print("\nI got a SIGINT, but I am not stopping")
5
1 signal.signal(signal.SIGINT, handler)
2 i = 0
3 while True:
4     time.sleep(.1)
5     print("\r{}".format(i), end="")
6     i += 1
```

ctrl+c 停止 SIGINT (发送一个信号, 可以通过 man signal 查看)

ctrl+\ 表示 SIGQUIT

上图程序表示 ctrl+c 时不会停止, 而 ctrl+\ 会停止。因为 ctrl+c 发送信号之后程序重新计数

软件无法捕捉的信号是 SIGKILL, ssh中断的时候信号是 SIGHUP 挂断

ctrl+z 暂停 (SIGSTOP)

```
~/cmd >>> nohup sleep 2000 &
```

& 表示在后台运行

```
~/cmd >>> sleep 1000
^Z
[1] + 63286 suspended sleep 1000
~/cmd >>> nohup sleep 2000 &
[2] 63567
Appending output to nohup.out
~/cmd >>> jobs
[1] + suspended sleep 1000
[2] - running nohup sleep 2000
~/cmd >>> █
```

```
~/cmd >>> bg %1
[1] - 63286 continued sleep 1000
~/cmd >>> jobs
[1] - running sleep 1000
[2] + running nohup sleep 2000
~/cmd >>> █
```

`bg %1` 在后台继续任务一, `fg` 恢复到前台, 重新连接标准输出

`kill -STOP %1` 可接各种signal

```
~/cmd >>> kill -HUP %1
[1] + 63286 hangup sleep 1000
~/cmd >>> jobs
[2] + running nohup sleep 2000
~/cmd >>> kill -HUP %2
~/cmd >>> jobs
[2] + running nohup sleep 2000
~/cmd >>> kill -KILL %2
[2] + 63567 killed nohup sleep 2000
~/cmd >>> jobs █ 这样就完成了工作
~/cmd >>> █
```

使用 `nohup` 的任务, 当你发送 `-HUP` 并不会停止

tmux : Sessions Windows Panes

terminal

`tmux` 打开另一个 `terminal`

`ctrl + a + d` 回到原来, 命令 `tmux a` 回到正在进行的 `tmux terminal`

```
Attached (from session 0)
~/cmd >>> tmux new -t foobar █
```

创建新 `terminal` 并指定名字

```
~/cmd >>> tmux ls
0: 1 windows (created Tue Jan 21 14:21:23 2020)
foobar-1: 1 windows (created Tue Jan 21 14:24:40 2020) (gro
```

```
tmux ls
```

window

`ctrl + a + c` (create) 创建新窗口, `ctrl + a + p` (previous) / `n` (next) 窗口切换。或者数字如 `ctrl + a + 1`

`ctrl + a + "` 上下分成两屏 `ctrl + a + %` 左右分 (不同的window做不同的事情)。`ctrl + a +` 方向键 移动

`ctrl + a + 空格` 等距显示

The screenshot shows a tmux session with three panes. The left pane displays a file listing with permissions, owner, modification date, and file names. The right pane shows a 'top' command output with columns for PID, USER, PRI, NI, VIRT, RES, SHR, %CPU, and %MEM. The bottom pane shows a terminal session with the command `~/cmd >>> python sigint.py`. A status bar at the bottom indicates the current window is 1 of 1, the time is 14:28, and the date is 18.

`ctrl + a + z` 缩放

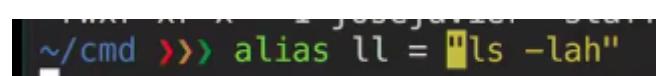
Dotfiles

alias别名

```
~/cmd >>> alias ll="ls -lah"
~/cmd >>> ll
total 120
drwxr-xr-x 10 josejavier staff 320B Jan 21 14:15 .
drwxr-xr-x@ 28 josejavier staff 896B Jan 21 11:12 ..
-rw-r--r-- 1 josejavier staff 521B Jan 21 12:11 aliases.sh
-rw-r--r-- 1 josejavier staff 367B Jan 21 10:45 case.sh
-rw-r--r-- 1 josejavier staff 120B Jan 21 13:29 config
-rw----- 1 josejavier staff 0B Jan 21 14:15 nohup.out
-rw-r--r--@ 1 josejavier staff 34K Jan 21 12:12 notes.html
-rw-r--r--@ 1 josejavier staff 3.7K Jan 21 12:12 notes.md
-rw-r--r-- 1 josejavier staff 40B Jan 21 11:02 path.sh
-rw xr-xr-x 1 josejavier staff 246B Jan 21 08:43 sigint.py
```

它正在执行该命令，而无需我键入整个命令。

注意：不能有空格如下图



```
~/cmd >>> alias mv="mv -i"
~/cmd >>> mv aliases.sh case.sh
overwrite case.sh? (y/n [n])
```

“move”已扩展为此“move -i”

查看别名：

```
~/cmd >>> alias mv
mv='mv -i'
~/cmd >>> alias ll
ll='ls -lah'
~/cmd >>>
```

注意：当关闭shell时所有alias都消失，所以可以通过点文件来保存配置

~/.bashrc

```
~/cmd >>> vim ~/.bashrc
```

```
1 alias sl=ls
```

上图 PS1 修改最前面的提示符

```
PS1="\w > "
```

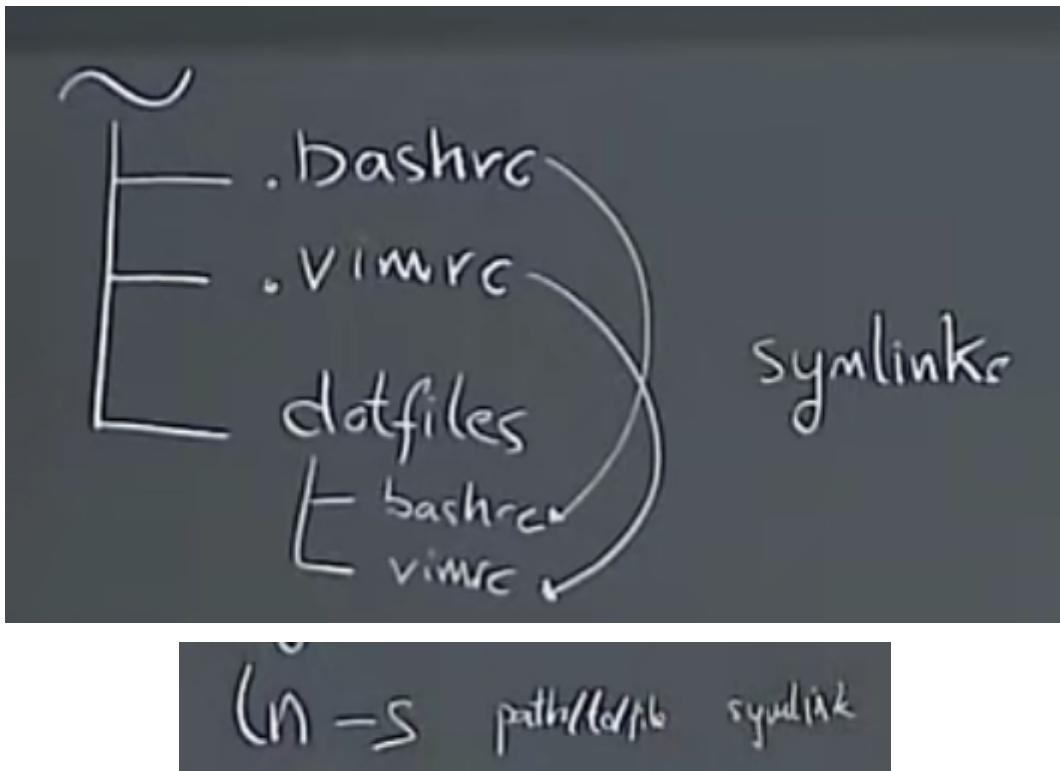
上图 \w 表示当前工作目录

其他点文件

```
~/.vimrc
```

```
.config/alacritty 修改字体
```

GNU stow



```
(ln -s path/to/file symlink)
```

使用符号链接

Remote Machines

```
~/cmd >>> ssh jjgo@192.168.246.142
~/cmd >>> ssh jjgo@foobar.mit.edu
```

SSH密钥

ssh-keygen

```
~/cmd >>> cat ~/.ssh/id_ed25519.pub | ssh jjgo@192.168.246.142 tee .ssh/authorized_keys
```

或者

```
Connection to 192.168.246.142 closed.
~/cmd >>> ssh-copy-id jjgo@192.168.246.142
```

复制

scp :

```
~/cmd >>> scp notes.md jjgo@192.168.246.142:foobar.md
notes.md                                         100% 3802      3.3MB/s   00:00
~/cmd >>>
```

rsync (适合多文件) :

```
~/cmd >>> rsync -avP jjgo@192.168.246.142:cmd
```

~/.ssh/config

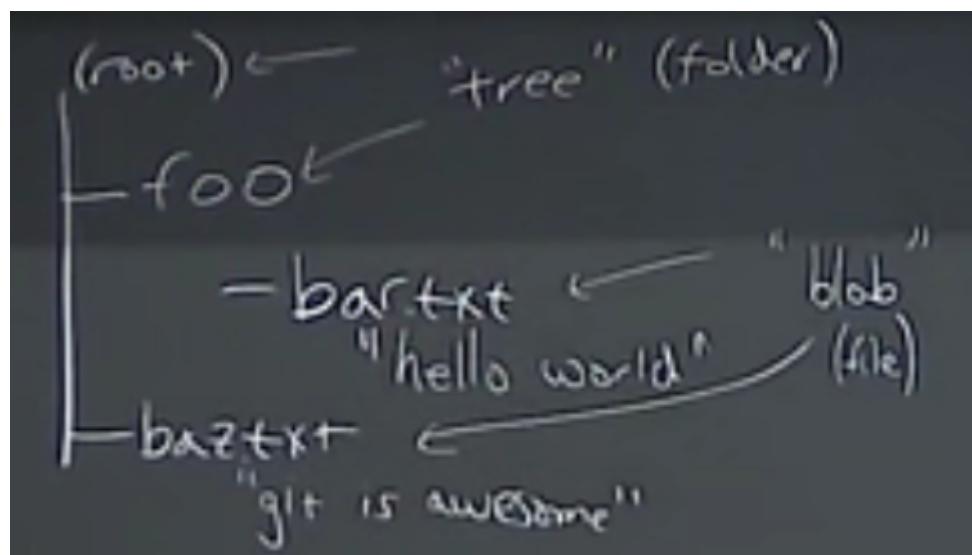
```
3 Host vm
2   User jjgo
1   HostName 192.168.246.142
4   IdentityFile ~/.ssh/id_ed25519
1   RemoteForward 9999 localhost:8888
```

如上设置后，可以如下ssh链接

```
~/cmd >>> ssh vm
```

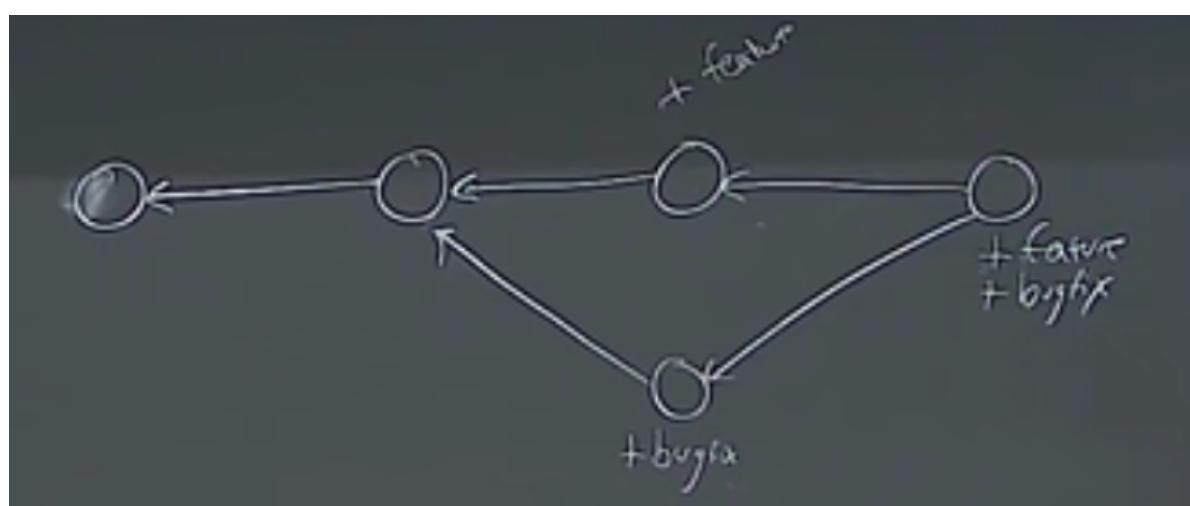
Lecture 6 Version Control(Git)

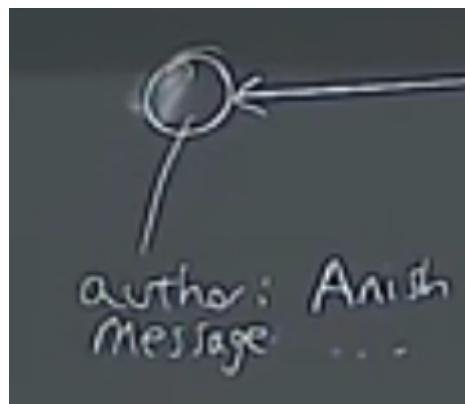
底层逻辑



文件夹folder称为tree，而文件file被称为blob

git使用有向无环图的结构，分支历史fork





每个节点还有自己的元数据

伪代码

```
type blob = array<byte>
type tree = map<String, tree | blob>
type Commit = Struct {
    parents: array<commit>
    author: String
    message: String
    snapshot: tree
}
```

blob是字节数组

tree是从字符串到tree或blob的映射

commit包含父节点commit数组（因为可能有多个父节点）、元数据（如作者、message）以及具体数据snapshot (tree)

type object = blob | tree | commit

objects = map<String, object>

```
def store(o):
    id = Sha1(o)
    objects[id] = o
```

object包含blob或tree或commit

存储数据是通过hash object成一个id然后存储id对应的object

```
def load(id):
    return objects[id]
```

读取数据是通过id在数组中寻找

实际上上述数据类型基本都以hash之后的id存储，类似于指针

除了上述数据类型，git还维护了一个reference用以存储hash之后的乱码id，用人类易读的名字来指代各个节点

references = map<String, String>

注意不能修改已形成的图，所以只能增加新的节点

git命令

git init 创建一个git

git help 后面加上命令，如 git help init

git status 查看当前的git状态

git add + 文件名 将文件提交到 staging area 暂存区

git commit 会弹出文本编辑器让你进行commit message书写

git log 可以可视化的显示出提交图（更有效的命令 git log --all --graph --decorate）

git cat-file -p hash值 可以看到这次snapshot的文件，再输入下图tree的hash值又能看到tree里面的内容

```
.../demo > git cat-file -p 42fb7a2
tree 68aba62e560c0ebc3396e8ae9335232cd93a3f60
author Anish Athalye <me@anishathalye.com> 1579721830 -0500
committer Anish Athalye <me@anishathalye.com> 1579721830 -0500

Add hello.txt
```

head和master

head 指向当前节点的指针

master 自动生成的branch

git checkout hash值（只需打前几个值）用来修改 head 指针，hash值也可以用其他别名如指着它的 master等

git checkout 文件名 放弃修改

git diff 可以查看自从上次snapshot之后的更改。还可以加上branch值查看从这个branch开始产生的diff，默认是head。当你加上两个branch就可以查看两个时间之间的改变

```
.../demo > git diff hello.txt
diff --git a/hello.txt b/hello.txt
index fdff486..2071a40 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1,2 +1,3 @@
 hello world
 another line
+aoisdjf
.../demo > [0:2.1] 1:missing- 2:playground与您的项目git diff也可以
```

branch和merge

git branch 列出当前的branch。-vv 显示更全面的信息

git branch 分支名 创建新分支

git checkout 分支名 head指向新的分支

git log --all --graph --decorate --oneline 更紧凑的显示图

```
.../demo > git checkout -b dog
Switched to a new branch 'dog'
```

上图 git checkout -b 分支名 等同于 git branch 分支名; git checkout 分支名

git merge 分支名 将当前的分支和指定的分支merge

Merge Conflict

git mergetool，一般使用 vimdiff

git merge --abort 暂停merge。再 git merge，然后使用 vim 解决冲突，然后 --continue

Git Remotes

上传端

`git remote` 列出当前的remote

`git remote add <remote>` 创建remote, 可以是网上仓库也可以是本地的一个位置

`git push <remote> <local branch>:<remote branch>` 推送

`git clone <url> <folder name>` 从某个网站 (也可以是本地地址) 克隆在某个文件夹的位置

不需要指定推送的remote方法 (直接推送到remote同名的branch)

`git branch --set-upstream-to=origin/master` 将当前master分支和remote的master分支绑定

`git push`

接收端

`git fetch` 查找是否远端是否有更新

`git pull=git fetch; git merge` 将本地的branch里链接上远端的branch

其他

命令 `git config` 或者修改 `~/.gitconfig`

`git clone --shallow` 快速复制但没有版本信息

`git diff` 查看修改, 然后 `git add -p 文件名` 可以手动交互式决定修改内容, `git diff --cached` 会显示会被提交的更改有哪些, 提交之后再来 `git checkout 文件名` 就可以放弃该修改

`git blame 文件名` 查看代码是由谁写的, 然后 `git show 代码hash` 获得详细信息

`git stash` 将文件回复到上次提交, 实际上是隐藏起来了; `git stash pop` 重新显示出来

`git bisect` 二分查找? 检查何时代码不能通过

`.gitignore` 修改当前目录下的 `.gitignore` 文件来避免上传二进制文件等, 如文件名、`*.o` 等

shell集成和vim插件 (如 `Gblame`) 可以更容易了解仓库变化

Lecture 7 Debugging and Profiling

Debug

日记记录

显示颜色

```
~/deb >>> printf "\e[38;2;255;0;0m This is red      \e[0m"
This is red %
```

系统日志放在`/var/log`文件夹里

```
~/deb >>> logger "Hello Logs"
~/deb >>> log show --last 1m | grep Hello
2020-01-23 14:15:56.951889-0500 0xd663c3 Default 0x0
061 0 logger: Hello Logs
~/deb >>>
0:zsh* | 1 我们找到了条目。我们刚刚创建的条目是 "Hello Logs" 23/01 14:16 / 11
```

logger 创建条目 log show 显示

Debugger (python)

python -m ipdb xxx.py 开始debug

l 列出代码

s 单步执行

restart 重启

c continue

p + 变量 打印

q 退出

b + 行数 设置断点

GDB (较底层)

```
jjgo@lion ~ >>> gdb --args sleep 20
```

run 执行

Python静态分析工具

pyflakes xxx.py

mypy xxx.py

甚至有英文的静态分析工具如 writegood

Profiling

主要是CPU profiler，一种是tracing profiler（每次进入函数都记录下来），另一种是sampling profiler（每隔一段时间会暂停程序来分析）

Python版本CPU Profiler

```
~/p/m/deb >>> python -m cProfile -s tottime grep.py 1000 '^import|def[^,]*$' *.py
```

```
~/p/m/deb >>> python -m cProfile -s tottime urls.py | tac
```

注：TAC与CAT相反，反向输出

上面的profiler显示的函数的运行的时间，可以用line profiler看更人性化的提示

```
~/p/m/deb >>> kernprof -l -v urls.py
```

```
@profile
def get_urls():
    response = requests.get('https://missing.csail.mit.edu')
    s = BeautifulSoup(response.content, 'lxml')
    urls = []
    for url in s.find_all('a'):
        urls.append(url['href'])
```

在最前面加一个`@profile`然后再次输入上面那个语句

Python版本Memory Profiler

Line #	Mem usage	Increment	Line Contents
1	39.578 MiB	39.578 MiB	@profile
2			def my_func():
3	47.211 MiB	7.633 MiB	a = [1] * (10 ** 6)
4	199.801 MiB	152.590 MiB	b = [2] * (2 * 10 ** 7)
5	47.211 MiB	0.000 MiB	del b
6	47.211 MiB	0.000 MiB	return a

perf

`stat`统计CPU信息

```
> sudo perf stat stress -c 1
```

`stress`是一个程序，如果`ctrl+c`会显示一些信息

`record`和`report`

```
jjgo@lion ~/pwndbg >>> sudo perf record stress -c 1
stress: info: [24913] dispatching hogs: 1 cpu, 0 io, 0 vm, 0 hdd
^C[ perf record: Woken up 3 times to write data ]
[ perf record: Captured and wrote 0.758 MB perf.data (19344 samples) ]

jjgo@lion ~/pwndbg >>> sudo perf report
```

Flame Graph/call graph

其他

`htop`

`du` (disk usage)

```
jjgo@lion ~/videos >>> du -h videos/
```

`ncdu`是交互版

```

ncdu 1.11 ~ Use the arrow keys to navigate, press ? for help
--- /home/jjgo/videos/videos ---
/..
8.6 GiB [#####] MIT-Missing-Semester-le...1-1400-Adhoc Section_1.mp4
8.6 GiB [#####] MIT-Missing-Semester-le...1-1400-Adhoc Section_3.mp4
8.6 GiB [#####] MIT-Missing-Semester-le...1-1400-Adhoc Section_4.mp4
8.6 GiB [#####] MIT-Missing-Semester-le...1-1400-Adhoc Section_2.mp4
434.0 MiB [ fed_lec5.mp4
403.3 MiB [ MIT-Missing-Semester-le...1400-Adhoc%20Section_1.mp4
2.2 MiB [ ./mypy_cache
304.0 KiB [ output.jpg
164.0 KiB [ ./git
12.0 KiB [ .lec3.py.swp
12.0 KiB [ /__pycache__
4.0 KiB [ lib.py
4.0 KiB [ lec5.py
4.0 KiB [ lec3.py
4.0 KiB [ lec1.py
4.0 KiB [ lec2.py
Total disk usage: 35.2 GiB Apparent size: 35.2 GiB Items: 108
0: perf 1: top- 2:ncdu* 3:lsof 4:strace 5:[tmux] 23/01 14:53:54
3: tti 所以NCDU是一个交互式版本，可以让我浏览文件夹，我很快就能看到

```



lsof

```
jjgo@lion ~ >>> sudo lsof
```

显示谁在使用这个文件或监听

```
jjgo@lion ~ >>> sudo lsof | grep ":4444 .LISTEN"
python    25031      jjgo    4u        IPv4          26630769      0t0
TCP *:4444 (LISTEN)
```

注: fd比find快

```
~/scratch >>> hyperfine --warmup 3 'fd -e jpg' 'find . -iname "*.jpg"'
```

Lecture 8 Metaprogramming

Make

```

paper.pdf: paper.tex plot-data.png
        pdflatex paper.tex

plot-%.png: %.dat plot.py
        ./plot.py -i $*.dat -o $@

```

版本控制

64.0.20190324

8.1.7
↑ ↑ ↑
major minor patch

README.md

[crates.io](#) v0.0.2 [docs](#) 0.0.2 [Azure Pipelines](#) succeeded [codecov](#) 89% [dependencies](#) up to date

Lecture 9 Security and Cryptography

熵

用于检验安全性

Entropy
 $\log_2(\# \text{ possible})$

Hash

sha1-160位

```
> printf 'hello' | sha1sum  
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
```

- non-inversible
- collision resistance

Key derivation functions(KDF)

类似hash函数但是会更慢

对称加密

Symmetric key cryptography

- Keygen() \rightarrow Key
- encrypt(plaintext, Key) \rightarrow ciphertext
- decrypt(ciphertext, Key) \rightarrow plaintext

}

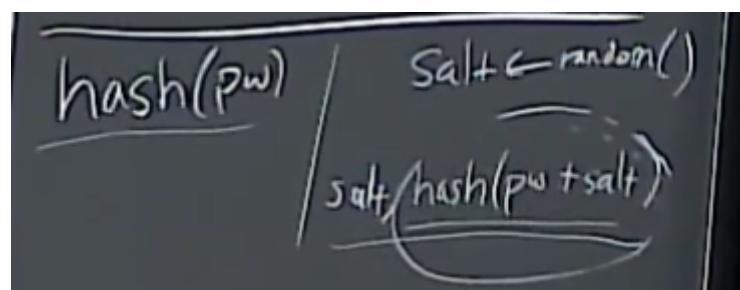
- given ciphertext,
can't figure out plaintext
(without Key)
- decrypt(encrypt(m, k), K) = m

如openssl:

```
> openssl aes-256-cbc -salt -in README.md -out README.enc.md
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
```

```
> openssl aes-256-cbc -d -in README.enc.md -out README.dec.md
enter aes-256-cbc decryption password:
> cmp README.md README.dec.md
> [ ]
```

(-)(Master)
(-)(Master)



salt也加入hash使得密码更随机

非对称加密

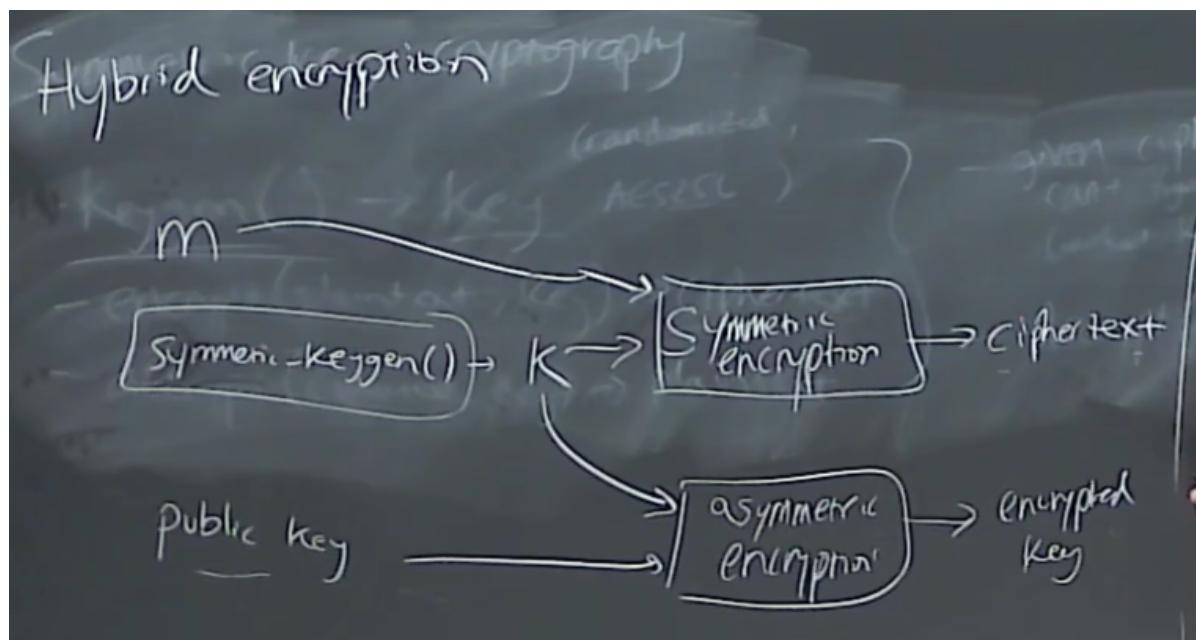
Asymmetric Key cryptography

Keygen() \rightarrow (public key, private key)

encrypt(p, public key) \rightarrow c
decrypt(c, private key) \rightarrow p

Sign (message, private key) \rightarrow signature
Verify (message, sig, public key) \rightarrow ok?
- hard to forge (without private key)
- correct

加密结合



Lecture 9 Potpourri

选项

如果文件名和选项名相同，如删除 -i 的文件

做法：使用两个 -，表示后面所有的东西（中间空格也看作不是flag）不被看作是选项，如 rm ---i

Lua

用来自定义系统功能

虚拟机

vagrant

Lecture 10 QA



Anonymous

What is the difference between `source script.sh` and `./script.sh`

source的话在当前的bash里运行，在当前的shell里定义，而./会实例化一个新的bash来运行，不会在当前的shell定义



Anonymous

What browser plugins do you use?

UBlock origin用来屏蔽广告

stylus

password manager

Anonymous

What are other useful data wrangling tools

pandoc文档格式转换

Anonymous

Any more Vim tips?

`m` 可以mark

`ctrl+o` 向后搜索

`ctrl+i` 向前搜索