GNU 命令行

续本达

清华大学 工程物理系

2024-07-13 清华

软件准备

GNU 环境 课程与作业环境

sed 文本流编辑器

gawk GNU awk 语言，数据驱动程序语言

- Alfred Aho, Peter Weinberger, and Brian Kernighan

coreutils GNU 核心命令

man-db 帮助文档阅读器

info 层级帮助文档阅读器

curl 网络请求和下载器

bc 计算器

file 查看文件类型

bash-doc bash 详细教程

```
apt update
apt install gawk file sed man-db info wget curl bc bash-doc neofetch
```

复习

Git

- 队友分工合作，过去与未来的合作

Python

- 程序结构，数据类型，数组张量，画图

技能学习的特点

- 归纳地学习黑客技术
- 先“不求甚解”再择最重要的“寻根究底”，在两种学习模式间自由跳跃。
  - 在麦克斯韦方程出现之前，已经有光学
  - 不是只有做完实验才能吃饭

Python 程序的支撑

- 程序由 Python 书写，或者 R、C++ 等等任何最佳工具
- 透明数据的格式和可复现的结果图。

如何把它们组织起来?

- 如何管理 GB、TB 乃至 PB 级的输入输出数据?
- 如何高效地在不同组输入数据上重复运行亿万次?
- 四大基本原则在此依然适用。

GNU 命令行界面：神秘的黑色魔法世界

- 我们已经接触的命令：`ls`, `cd`, `rm`, `git`, `less`, `man`, `cat`, `pwd`

为什么学习命令行

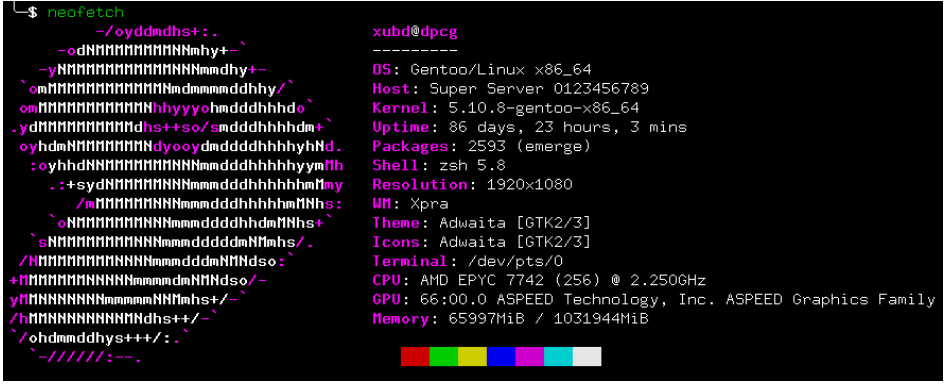
- 复现 原则的保证
  - Python 程序，有明确的功能，定义了输入和输出。
  - 需要高效且可复现地整合协调各类程序，处理数据。
- 体现 透明 原则
  - 数据的中间结果，可以用命令行工具查看 `cat`, `less`, `h5dump` …
  - 注：对应的图形工具也存在，可互相搭配。
- 体现 最佳工具 原则
  - GNU 命令行目前是文件和数据处理流程管理的最佳工具
  - 人操纵计算机最高效的方式：快捷键

参考书

Jeroen Janssens, Data Science at the Command Line

两周之后，你可能会成为：

ゆき Installs Gentoo



数字时代：如何更本质地与世界相连？

- 我们购买设备，使用设备。
  - 如果可以理解设备，改造设备、修理设备和创造设备，生活会有什么变化？
- 程序和命令行是掌控自己数字世界的重要一步
  - 如果你不掌控自己的世界，别人就会代替你成为主人
  - 菜单、窗口是他们设计好给我们用的，程序、命令是自己组合给自己用的。

手机例子

- Gentoo on Android

命令行迫使我们思考

对任何人下达明确的指令，都是困难的。对机器也是。除非通过思考，我们无法让指令明确
而命令行会非常诚实，任何不明确的指令都会被惩罚（报错）

命令行的易用性

- 命令行是天然的 REPL
- 命令行有最丰富的文件操作工具
- 介于程序设计语言与图形界面之间
- 是计算机系统的“母语”，用户与开发者的重叠更大
  - 科学计算中，不是任何时候都有现成的工具可用，经常需要自己开发
    - 对于科学探索来说，地表最强的产品经理也没用用
  - 依赖命令行可以缩小开发者与用户的鸿沟

命令行的整合性和扩展性

- 所有的语言工具都可以从命令行调用
  - 新的语言写成的命令行工具不断涌现，许多工具非常适合用于数据处理
- 命令行是天然的“胶水”，整合各类语言协同工作
  - 鼓励小巧精悍工具的组合，避免又长又难理解的马拉松程序出现
  - 不论流行的语言 Perl、Python、Scala 如何变化，命令环境基本不变

命令行的易用性

- 命令行是天然的 REPL
- 命令行有最丰富的文件操作工具
- 介于程序设计语言与图形界面之间
- 是计算机系统的“母语”，用户与开发者的重叠更大
  - 科学计算中，不是任何时候都有现成的工具可用，经常需要自己开发
    - 对于科学探索来说，地表最强的产品经理也没用用
  - 依赖命令行可以缩小开发者与用户的鸿沟

命令行的整合性和扩展性

- 所有的语言工具都可以从命令行调用
  - 新的语言写成的命令行工具不断涌现，许多工具非常适合用于数据处理
- 命令行是天然的“胶水”，整合各类语言协同工作
  - 鼓励小巧精悍工具的组合，避免又长又难理解的马拉松程序出现
  - 不论流行的语言 Perl、Python、Scala 如何变化，命令环境基本不变

命令行的自动性

- 使用的命令可以轻易组成脚本，重复使用
  - 图形界面不易录制操作
- 命令的传递方式是字符，字符可以被清晰地表述处理，因此可充分自动。

命令行的普适性

- GNU/Linux, 各种 Unix，macOS，Microsoft Windows WSL.
- 世界顶级的超级计算机，云计算主机，智能手机，物联网设备都可用同一套命令行工具操作
- 已有约 50 年历史，一定会继续存在
  - 字符的编码标准起源于电报机时代，一个多世纪以来不变。
  - 当今的计算机“终端”“控制台”“命令行”“外壳”（名字太多，统一不了）仍在模拟一个电报机（teleport）。

命令行的自动性

- 使用的命令可以轻易组成脚本，重复使用
  - 图形界面不易录制操作
- 命令的传递方式是字符，字符可以被清晰地表述处理，因此可充分自动。

命令行的普适性

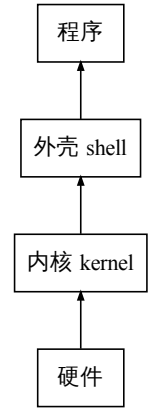
- GNU/Linux, 各种 Unix，macOS，Microsoft Windows WSL.
- 世界顶级的超级计算机，云计算主机，智能手机，物联网设备都可用同一套命令行工具操作
- 已有约 50 年历史，一定会继续存在
  - 字符的编码标准起源于电报机时代，一个多世纪以来不变。
  - 当今的计算机“终端”“控制台”“命令行”“外壳”（名字太多，统一不了）仍在模拟一个电报机（teleport）。

外壳 shell

- 外壳 shell 是相对于操作系统内核 kernel 而言，与人类交互的部分
- 有命令行 (Command Line Interface, CLI) 和图形 (Graphical User Interface, GUI) 两种
- Python 在 shell 之上运行

实际上 Python 也可以成为外壳环境

- Python 与 bash 平等，都是“用户态”程序 userspace
- bash GNU Bourne-Again SHell, 目前使用最为广泛，方便进行文件操作和调用命令
  - `man bash` 可阅读在线文档
  - `info bash` 获得详细教程



命令的类型

- 可执行程序，例如 `ls`

```
type ls
```

```
ls is hashed (/bin/ls)
```

- 脚本，例如 Python 程序

```
type vitables
file $(realpath /usr/bin/vitables)
```

```
vitables is /usr/bin/vitables
/usr/bin/vitables: Python script, ASCII text executable
```

- Shell 内建命令，例如 `cd`

```
type cd
type type
```

```
cd is a shell builtin
type is a shell builtin
```

命令的类型（二）

- Shell 函数

```
function hello { :; }
type hello
```

```
hello is a function
hello ()
{
    :
}
```

- 别名 alias

```
alias vt='vitables -m r'
type vt
```

```
vt is aliased to `vitables -m r'
```

概念

- 标准输入 standard input `stdin` (file descriptor 0)
- 标准输出 standard input `stdout` (fd 1)
- 标准报错 standard error `stderr` (fd 2)

管道

- 把前一个程序的标准输出和后一个标准输入连接起来
- 无限串联，每个命令各司其职

复习 Python 的标准输入和输出

- `input()` 标准输入
- `print()` 标准输出
- `print(..., file=sys.stderr)` 标准报错

输出

```
echo "I am using the command line!"
```

```
I am using the command line!
```

管道注入 wc

```
echo "I am using the command line!" | wc -c
```

```
29
```

管道用于筛选

```
seq 5
```

```
1
2
3
4
5
```

```
seq 30 | grep 7
```

```
7
17
27
```

```
seq 100 | grep 7 | wc -c
```

```
56
```

保存管道的中间结果

- tee

```
seq 100 | grep 7 | tee number-seven.txt | wc -c
```

```
56
```

```
cat number-seven.txt
```

```
7
17
27
37
47
57
67
70
71
72
73
74
75
76
77
```

重定向

- 标准输入和输出都可以被重定向到文件

```
seq 100 > s100
cat s100 | head
```

```
1
2
3
4
5
6
7
8
9
10
```

```
wc -l < s100
```

```
100
```

自由管道 FIFO named pipe

```
mkfifo filter.pipe
seq 100 > filter.pipe &
grep 7 < filter.pipe
```

```
[1] 42881
7
17
27
37
47
57
67
70
71
72
73
74
75
76
77
78
79
```

任务控制

- bash 环境中 & 代表置于后台运行。

```
sleep 1m &
sleep 2m &
jobs
```

```
[1] 43068
[2] 43069
[1]-  Running                 sleep 1m &
[2]+  Running                 sleep 2m &
```

- fg 将任务拉至前台
- bg 将任务转至后台
- kill 将任务终结
- ^z （Ctrl-z），挂起前台任务
- ^c 终结前台任务

计算器的四种写法

- awk 派，使用 awk 语言

```
cat s100 | awk '{total+=$NF} END {print total}'
```

5050

- Python 派？Python 无法写成一行
- paste 派

```
cat s100 | paste -s -d +
cat s100 | paste -s -d + | bc
```

1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20+21+22+23+24+25+26+27+28+29+30+31+32+3
5050

帮助

- manual page `manpage`

```
man cat
man paste
```

  - Emacs `M-x woman`
- info page

```
info cat
info paste
```

  - Emacs `C-h i`
- 命令 `--help -h`

帮助（二）

```
uname --help
```

```
Usage: uname [OPTION]...
Print certain system information.  With no OPTION, same as -s.

  -a, --all                print all information, in the following order,
                             except omit -p and -i if unknown:
  -s, --kernel-name        print the kernel name
  -n, --nodename           print the network node hostname
  -r, --kernel-release     print the kernel release
  -v, --kernel-version     print the kernel version
  -m, --machine            print the machine hardware name
  -p, --processor          print the processor type (non-portable)
  -i, --hardware-platform  print the hardware platform (non-portable)
  -o, --operating-system   print the operating system
      --help     display this help and exit
      --version  output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/uname>
or available locally via: info '(coreutils) uname invocation'
```

本机信息

```
hostname
uname -a
```

```
dpcg
Linux dpcg 5.10.8-gentoo-x86_64 #1 SMP Sun Aug 15 14:46:49 CST 2021 x86_64 AMD EPYC 7742 64-
Core Processor AuthenticAMD GNU/Linux
```

```
id
date
```

```
uid=1003(xubd) gid=1003(xubd) groups=1003(xubd),10(wheel),103(vboxusers),250(portage),999(xpra
Mon Jul 25 10:53:06 AM CST 2022
```

本机信息

```
uptime
dmesg | tail
```

```
 10:53:35 up 86 days, 23:32,  7 users,  load average: 1.05, 0.59, 0.36
[6943834.870848] nfs: server jpd not responding, timed out
[6943847.158500] nfs: server jpd not responding, timed out
[6943852.790312] nfs: server jpd not responding, timed out
[6943871.733748] nfs: server jpd not responding, timed out
[6943888.645236] nfs: server jpd not responding, timed out
[6943894.261019] nfs: server jpd not responding, timed out
[6943900.404864] nfs: server jpd not responding, timed out
[6943906.420662] nfs: server jpd not responding, timed out
[7345744.319918] xpra[197935]: segfault at 1 ip 00007f687e955f89 sp 00007fff68363050 error 4 i
3.so.0.2404.26[7f687e901000+5d000]
[7345744.319934] Code: 00 00 48 89 44 24 08 31 c0 e8 53 be fc ff 48 85 c0 74 1d 0f b6 50 4c 6
```

日常

- 创建空文件

```
touch nf
ls -l nf
```

```
-rw-r--r-- 1 xubd xubd 0 Jul 25 11:01 nf
```

- 输出信息

```
echo "Hi!"
```

```
Hi!
```

cut 选择列

```
sudo dmesg | cut -c-15 | tail
```

```
[370615.920192]
[370616.151878]
[370616.389111]
[370619.345380]
[370619.345382]
[370619.365730]
[370619.365751]
[476022.060361]
[476109.862666]
[476204.332049]
```

```
ls -1 */* | cut -d/ -f 1 | uniq
```

```
__pycache__
_minted-e1
_minted-e2
_minted-e3
_minted-e4
_minted-e5
_minted-e6
_minted-e7
_minted-e8
_minted-e9
_minted-ea
_minted-eb
_minted-ec
_minted-ed
_minted-ee
_minted-ef
_minted-p2-ein
_minted-p2
dataframe-practice
```

文件信息

- 寻找 dev/waveform-compress 路径之下所有 R 程序文件

```
find dev/waveform-compress -name "*.R"
```

```
dev/waveform-compress/compress_comp.R
dev/waveform-compress/compress_core.R
dev/waveform-compress/compress_one.R
dev/waveform-compress/getH.R
dev/waveform-compress/wdlib.R
```

- 识别文件类型

```
file dev/waveform-compress/compress_one.R
```

```
dev/waveform-compress/compress_one.R: a /usr/bin/env Rscript script, ASCII text executab
```

定义

- 通配符可以按规则匹配，用于构造简单匹配模式
  - * 匹配任意多个任意字符
  - ? 匹配一个任意字符

```
cd physics_data
ls *.html
```

```
Command-Line.slides.html     Python-Basics.slides.html
Data-Formats.slides.html     Python-Constructs.slides.html
Data-Frame-RPy.slides.html   Python-Functions.slides.html
Data-Frame.slides.html      Python-Modules.slides.html
Pandas-RDB.html      Python-SQLite.slides.html
Pandas-RDB.slides.html      Shell-Scripts.slides.html
```

```
echo *.html
```

```
Command-Line.slides.html Data-Formats.slides.html Data-Frame-RPy.slides.html Data-
Frame.slides.html Pandas-RDB.html Pandas-RDB.slides.html Python-Basics.slides.html
Constructs.slides.html Python-Functions.slides.html Python-Modules.slides.html Pyth
SQLite.slides.html Shell-Scripts.slides.html
```

例子

```
ls Python-*.html
```

```
Python-Basics.slides.html       Python-Modules.slides.html
Python-Constructs.slides.html   Python-SQLite.slides.html
Python-Functions.slides.html
```

```
ls *.?pynb
```

```
Command-Line.ipynb      Pandas-RDB.ipynb
Data-Formats.ipynb      Python-Basics.ipynb
DataFrame.ipynb     Python-Constructs_Functions.ipynb
Data-Frame-RPy.ipynb      Python-Functions.ipynb
GLM.ipynb     Python-IO_Survival.ipynb
GLM-Python.ipynb      Python-SQLite.ipynb
LIGO-intro.ipynb      Relational-Algebra.ipynb
LOSC_Event_tutorial.ipynb   Relational-SQL.ipynb
Make-Pipeline.ipynb      Shell-Practice.ipynb
Merge-GroupBy.ipynb      Shell-Scripts.ipynb
NumPy.ipynb      Visualization.ipynb
NumPy-SciPy.ipynb
```

交互模式

- ~~REPL~~

脚本模式

- 把命令集合起来，逐条执行
- 一边文件名以 .sh 结尾

```
echo "cat s100 | paste -s -d + | bc" > add-up-100.sh
bash add-up-100.sh
```

```
5050
```

让脚本可执行

- 在第一行填加脚本解释器 `#!/bin/bash` ，告知内核本脚本要由 `/bin/bash` 解释执行。 “#!”也称为“shebang”，意思是“sharp”（#）和“bang”（!）。

```
sed '1i#!/bin/bash' -i add-up-100.sh
cat add-up-100.sh
```

```
#!/bin/bash
cat s100 | paste -s -d + | bc
```

- 赋予可执行权限

```
chmod +x add-up-100.sh
./add-up-100.sh
```

```
5050
```

脚本程序

- 世界上本没有脚本，打的命令多了，聚在一起，就形成了脚本
- 脚本可以当作命令使用

例子：=bash= 脚本

```
cat arguments.sh
```

```
#!/bin/sh

echo The first argument is $1
echo The second argument is $2
echo all the arguments are $@
```

```
chmod +x arguments.sh
./arguments.sh 1 输入 2 参数
```

```
The first argument is 1输入
The second argument is 2参数
all the arguments are 1输入 2参数
```

Shebang 和脚本命令

- `#!` 是一对有特殊含义的字符，它处在第一行开头在被执行时，内核会寻找其后的解释器来运行其后的脚本。
- `chmod` 改变文件的权限，分别为 r 读 w 写 x 执行。
- `chmod +x arguments.sh` 给 `arguments.sh` 可执行的权限。
- 执行时，系统使用 `#!/bin/sh` 指定的 `/bin/sh` 执行程序。
  - `./arguments.sh` 等价于 `/bin/sh arguments.sh` 。
  - 注意 `/bin/sh` 很可能不是 `bash` 。需要 `bash` 时，直接使用 `/bin/bash`

```
realpath /bin/sh # dash 是不同的 shell，功能少速度快
```

```
/usr/bin/dash
```

- 如果可执行的脚本在 `PATH` 指定的路径里，就可以被 shell 找到，例如 `egrep` 。

```
file $(which egrep)
```

```
/bin/egrep: POSIX shell script, ASCII text executable
```

脚本的参数

- 脚本调用时的参数，在脚本中使用 ‘$1‘ ‘$2‘
  - 相当于 Python 的 ‘sys.argv[1]‘, ‘sys.argv[2]‘
- 所有的参数是 ‘$@‘

```
cat /bin/egrep
```

```
#!/bin/sh
exec /bin/grep -E "$@"
```

- bash 脚本例子

```
file /usr/bin/* | grep "Bourne-Again shell" | head
```

```
/usr/bin/AddOrReplaceReadGroups:                Bourne-Again shell script, ASCII text
/usr/bin/ant:                                   Bourne-Again shell script, ASCII text
/usr/bin/antlr:                                 Bourne-Again shell script, ASCII text
/usr/bin/antlr3:                                Bourne-Again shell script, ASCII text
/usr/bin/any2djvu:                              Bourne-Again shell script, ASCII text
/usr/bin/apache-rat:                            Bourne-Again shell script, ASCII text
/usr/bin/BamIndexStats:                         Bourne-Again shell script, ASCII text
/usr/bin/BamToBfq:                              Bourne-Again shell script, ASCII text
```

bash 变量

- 在 bash 中，一切皆字符串
- 变量需要额外的 $ 来指定， $a 和 ${a} 代表变量 a 的值
  - 类比 make ，使用 $(a) 代表变量取值
- 赋值号“=”的前后都不能有空格，与 Python 的格式建议相反！

```
a=1
echo $a
b=我是谁
echo ${a}${b}
```

```
1
1我是谁
```

```
echo "${a}${b}, 我从哪里来" # 双引号中变量是被替换的
```

```
1我是谁，我从哪里来
```

变量的引用

- 无引号时，空格起作用
- 单引号字符串中的字符保持不变
- 双引号时，字符串中的变量被替换

```
echo '${a}${b}，我从哪里来' # 单引号保持不变
```

```
${a}${b}，我从哪里来
```

引用命令的执行结果

- 把命令运行的标准输出赋给变量
- 与 make 类似，换行分隔的列表被转成由空格分隔

```
n_list=$(seq 9)
echo ${n_list}
```

```
1 2 3 4 5 6 7 8 9
```

```
n233=$(seq 10000 | egrep ^23+$)
echo ${n233}
echo ${#n233} # 字符个数
```

```
23 233 2333
11
```

bash 数组

- 虽然取名为数组 array ，但与 Python 的 列表更接近。
  - 可以存储任意元素。

```
busket=()
busket+=( 香蕉 弥猴桃 牛油果 )
busket+=( 西瓜 荔枝 )
echo ${busket} # 按普通变量取值，只有第一个元素
echo ${busket[@]}
echo ${busket[3]}
```

```
香蕉
香蕉 弥猴桃 牛油果 西瓜 荔枝
西瓜
```

```
echo ${#busket[@]} # 元素个数
```

5

bash 字典

- 又称关联数组 associative array ，类比 Python 字典

```
declare -A cargo # 声明 cargo 是字典
cargo[banana]=3
cargo[apple]=4
echo ${cargo[banana]}
echo ${!cargo[@]} # 取词
echo ${cargo[@]} # 取值
```

```
3
apple banana
4 3
```

bash 算术

- bash 一等公民是字符串，算术运算用于辅助和便利
  - 比用管道 和 `$(...)` 简洁

```
echo $((1+3)) $((3*5))
echo $(echo 1+3 | bc) $(echo 3*5 | bc)
```

```
4 15
4 15
```

- 但是计算能力有限，只能做简单运算

```
echo $((2**100))
echo $(echo "2^100" | bc)
```

```
0
1267650600228229401496703205376
```

选择结构

```
if [ 3 -gt 2 ]; then
    echo "3>2"
else
    echo "3<=2"
fi
```

3>2

判断语句

- `[...]` 是一个程序，正式名字是 `test`
- 真假判断来自该程序执行的返回值。
  - 0 为真：执行成功
  - 非 0 为假：执行失败
  - 注意与 Python 正好相反！

文档

- `man [` 、 `info test`

真假判断

- $? 或 ${?} 变量值是前一条命令的返回值

```
[ 1 = 2 ] # 假，返回非 0
echo $?
[ 2 != 3 ] # 真，返回 0
echo $?
```

```
1
0
```

- [INTEGER1 -eq INTEGER2] 相等
- [INTEGER1 -ge INTEGER2] 大于等于
- [INTEGER1 -lt INTEGER2] 小于
- [INTEGER1 -ne INTEGER2] 不等
- [! EXPRESSION] 取否
- [EXPRESSION1 -a EXPRESSION2] 取和
- [EXPRESSION1 -o EXPRESSION2] 取或

bash 内建真假判断

- bash 提供了 [[...]] 的内建命令，兼容 [...] 语法，有更多功能
- 内建命令，调用速度更快
- 命令层次的逻辑运算
  - command1 && command2 取和
  - command1 || command2 取或
  - ! command 取否
- 逻辑运算替代选择结构，以下两者等价

```
if [ 3 -gt 2 ]; then
    echo "3>2"
else
    echo "3<=2"
fi
# 逻辑运算替代选择结构，可读性更强
[[ 3 -gt 2 ]] && echo "3>2" || echo "3<=2"
```

```
3>2
3>2
```

循环结构

- for 循环，计算 Fabonacci 数
- 从 seq 10 结果中，取得 10 个元素，用变量 i 遍历
  - 类比： Python for 循环，迭代器

```
x=1
y=1
for i in $(seq 10); do
    s=$(($x+$y))
    x=${y}
    y=${s}
done

echo ${y}
```

144

while 循环

- 与 `seq 1000 | egrep ^23+$` 等价

```
seq 1000 | while read line
do
    echo $line | egrep -q '^23+$' && echo $line
done
```

```
23
233
```

until 循环

- 与 while 循环类似，条件相反

```
seq 1000 | until ! read line
do
    echo $line | egrep -q '^23+$' && echo $line
done
```

循环控制命令

- break 终止循环、 continue 跳入下一轮循环

```
seq 1000 | until ! read line
do
    if echo $line | egrep -q '^23+$'; then
        echo $line
        break
    fi
done
```

函数定义

- bash 函数是五种命令之一

```
function greet() {
    echo "Hello, $1"
}

greet 大佬
```

```
Hello, 大佬
```

函数递归

- 递归计算 Fabonacci 数
- `$(command)` 从 `command` 的标准输出取得结果
- `return` 给返回值，只能用 `$?` 捕捉或者用于逻辑运算
  - 与 Python 函数完全不同，因为命令成功与失败的判断更重要

```
function fib() {
    if [ $1 -le 2 ]; then
        echo 1
    else
        echo $(($(fib $(($1-1))) + $(fib $(($1-2)))))
    fi
}

fib 11
```

89

bash 的代码块与子进程

- `{...}` 代表一个代码块，把多行代码汇总成一个单元。
- `(...)` 使用复刻 (fork) 一个子 bash 运行代码。
  - 子 bash 中的变量设置不影响外层 bash 。

```
s=1
( s=2; )
echo $s
```

```
1
```

  - 类比： `$(seq 100)` 代表复刻后运行 seq 提取标准输出。

bash 调试

交互法 在 bash 环境试验获得想要的语句

屏幕输出法 用 `echo ... >&2` 输出到标准错误 stderr

额外调试信息输出法 `set -xv` 开启调试输出

- `set +xv` 关闭调试输出
- `bash -xv script.sh` 以调试模式执行脚本

bash(1)

- `-v` Print shell input lines as they are read.
- `-x` Print commands and their arguments as they are executed.

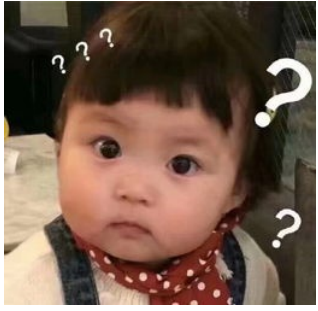
使用 bash 的原因

- bash 的列表、字典，形似 Python，计算能力不如 bc ，为什么值得使用？
- 最佳工具：使用 bash 处理文件时，有时需要列表、字典和算术
  - 但不需要太多。需要更专业的操作时，意味着更高级的工具胜任
  - “信手拈来”的境界是创新的温床，重大的全新突破都在有了足够积累后不经意间出现
    - 不经意的前提是：放松、随意
  - 不让大脑等待双手是最佳工具原则的最高等级
    - 人机交互，打字，扔掉鼠标
    - 避免不必要的环境切换

sl

```
sl
```

```
bash: sl: command not found
```



```
apt install sl
```

matrix 可操作动画

- 安装

```
apt install cmatrix
```

- 文档

```
man cmatrix
```

ASCII Art

```
apt install sysvbanner bsdgames
```

- worm 贪食蛇
- worms 模拟蚯蚓

```
banner physics
```

```
######   #     #  #     #   #####   ###   #####    #####
#     #  #     #   #   #    #     #   #   #     #  #     #
#     #  #     #    # #     #         #   #        #
######   #######     #      #####    #   #         #####
#        #     #     #           #   #   #              #
#        #     #     #     #     #   #   #     #  #     #
#        #     #     #      #####   ###   #####    #####
```

ASCII Art: cowsay

```
apt install cowsay-off
```

```
cowsay -e xx -T W "parity?"
```

```
 ----------
< parity? >
 ----------
        \   ^__^
         \  (xx)\_______
            (__)\       )\/\
             W ||----w |
                ||     ||
```

cowsay II

```
cowsay -f meow "parity?"
```

```
 ----------
< parity? >
 ----------
  \
   \ ,     _ ___.--'''`--''//-,-_--_.
      \`"' ` || \\ \ \\/ / // / ,-\\`,_
     /'`  \ \ || Y  | \|/ / // / - |__ `-,
    /@"\  ` \ `\ |  | ||/ // | \/  \  `-._`-,_.,
   /  _.-. `.-\,___/\ _/|_/_\_\/|_/ |     `-._._)
   `-'``/  /  |  // \__/\__  /  \__/ \
        `-'  /-\/  | -|   \__ \   |-' |
          __/\ / _/ \/ __,-'   ) ,' _|'
         (((__/(((_.' ((___..-'((__,'
```

- 其它动物

```
ls /usr/share/cowsay/cows
```

管道练习

```
apt install fortune-zh lolcat
```

```
fortune | cowsay | lolcat -a
```

```
 ------------------------------------------------------------------------
/ 谷神不死，是谓玄牝。 玄牝之门，是谓天地根。 绵绵若存，用之不勤。   \
\ --《道德经》                                                           /
 ------------------------------------------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

ASCII Art Anime

```
apt install bb
bb
```

ASCII Art color Anime

```
apt install caca-utils mpv
cacafire
```

```
mpv "http://hep.tsinghua.edu.cn/~orv/teaching/Yuki_Installs_Gentoo.mp4" -vo caca
```

再来一打

- 小玩具类

```
apt-cache depends games-toys
```

- 所有类

```
apt-cache depends games-all
```

- 欢迎大家以 merge request 的形式补充更多的游戏
  - ~~尤其是能在校园网内联网的!~~