易汉博 | 领先的大数据与健康解决方案
Leading solutions for big data and health

Linux 学习

易生信

www.ehbio.com/Training

train@ehbio.com

2018-04-01

Contents

1 Linux 基础内容

视频课见 http://bioinfo.ke.qq.com。

1.1 Linux 系统简介和目录理解

1.1.1 为什么要用 Linux 系统

个人认为，Linux 操作系统和类 Linux 操作系统的命令行界面是最适合进行生物信息分析的操作系统。原因有三点：

- 长期运行的稳定性
- 多数软件只有 Linux 版本
- 强大的 Bash 命令简化繁琐的操作，尤其是大大简化重复性工作

但对于初学者来说，接触和理解 Linux 操作系统需要一些时间和摸索。陡然从可视化点选操作的 Windows 进入到只有命令行界面的 Linux，最大的陌生感是不知道做什么，不知道文件在哪？

我们这篇教程就带大家学习、熟悉、体会 Linux 系统的使用。

1.1.2 Linux 系统简介

- Linux 是一种多用户、多任务的操作系统。最开始是由 Linus Torvalds 与 1991 年发布，后由社区维护，产生了不同的分发版。
- 常见版本有 `Centos`, `Ubuntu`, `RedHat`, `Debian` 等。服务器多用 `Centos` 系统，免费，稳定，但更新慢。`Ubuntu` 系统更新快，注重界面的体验，适合自己笔记本安装。有面向中国的“麒麟”系统。其它两个没用过，`Centos` 与 `RedHat`, `Debian` 与 `Ubuntu` 同宗，命令行操作起来很相似。

1.1.3 如何获取 Linux 系统

- 如果自己的单位有共有服务器，可以尝试申请账号。
- 自己的电脑安装双系统或虚拟机。
- 使用gitforwindows在 windows 下模拟使用 Linux 命令。
- 购买一块云服务器
- 试验下在线学习平台实验楼 https://www.shiyanlou.com (里面也有不少 Linux 教程，任意点一个进去，双击桌面的 `Xfce` 图标，都可以启动 Linux 终端)

1.1.4 Linux 系统登录

登录服务器的 IP 是：192.168.1.107; 端口是：22；用户名是每个人的姓名全拼，如陈同为 chentong (全小写，无空格)；密码是 yishengxin。

Figure 1.1: 配置 Xshell 登录服务器 1。

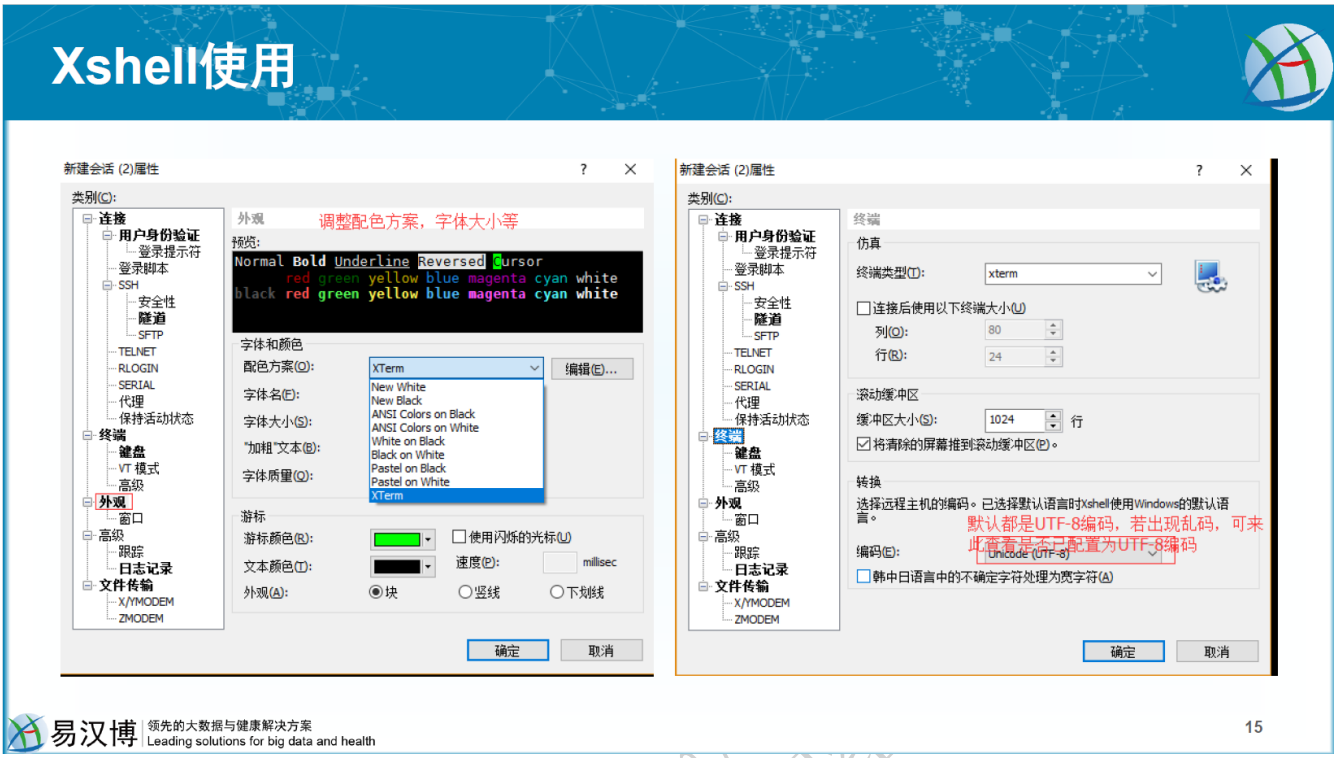

Figure 1.2: 配置 Xshell 登录服务器 2。

1.1.5 初识 Linux 系统

既然用 Linux，我们就摒弃界面操作，体验其命令行的魅力和庞大。后续操作都是在命令行下进行的，主要靠键盘，少数靠鼠标。

登录 Linux 系统后，呈现在眼前的是这样一个界面:

```
Last login: Mon Jun  5 16:56:56 2017 from 239.241.208.209

Welcome to aliyun Elastic Compute Service!

ct@ehbio:~$
```

首先解释下出现的这几个字母和符号:

- ct: 用户名
- ehbio：如果是登录的远程服务器，则为宿主机的名字；若是本地电脑，则为自己电脑的名字。
- ~: 代表家目录, 在我们进入新的目录后，这个地方会跟着改变
- $: 用来指示普通用户输入命令的地方；对根用户来说一般是 #
- http://bashrcgenerator.com/可视化定制不同的显示方式。
- 个人习惯的展示：`PS1=\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\u@\h:\w\$`

1.1.6 我的电脑在哪？

打开 Windows，首先看到的是桌面；不爱整理文件的我，桌面的东西已经多到需要 2 个屏幕才能显示的完。另外一个常用的就是我的电脑，然后打开 D 盘，依次点开对应的文件夹，然后点开文件。

Linux 的文件系统组织方式与 Windows 略有不同，登录进去就是家目录，可视为 Windows 下的桌面 [^Linux 的家目录严格来说可能类似于 Windows 下的 'C:\\Users\\ct']。在这个目录下，我们可以新建文件、新建文件夹，就像在桌面上的操作一样。

而 Linux 的完整目录结构如下：

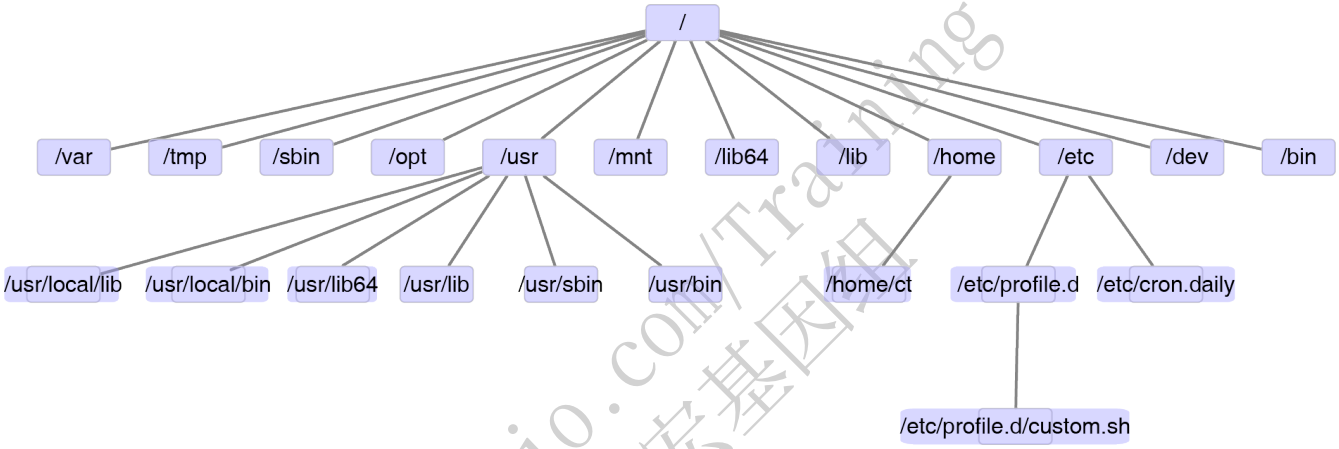

Figure 1.3: Linux 目录层级结构。

```
# 若提示命令找不到，运行下面语句安装 tree
# 需要有根用户权限
# yum install tree.x86_64
ct@ehbio:~$ tree -d -L 2 /
```

作为一个普通用户，通常只在`/home/usr`, `/tmp` 下有可写的权限，其它目录最多是可读、可执行，部分目录连读的权限都没有。这种权限管理方式是 Linux 能成为真正多用户系统的一个原因。后面我们会讲解如何查看并修改这些权限。

1.1.7 硬件信息查看

看完目录结构了，来看一下硬盘有多大，有多少可用空间，只需要运行 `df -h` 命令。

```
ct@ehbio:~$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda2             193G   61G  122G  34% /
tmpfs                 127G  344K  127G   1% /dev/shm
/dev/sda1             190M   77M  103M  43% /boot
```

Table 1.1: Linux 下目录简介

Path	Description
/	根目录
/bin	常用软件如 ls, mkdir, top 等的存放地
/dev	硬件相关
/etc	存放系统管理和配置相关文件
/etc/cron*	与定时任务相关的文件夹，可执行程序放置到对应文件夹就可以定时执行
/etc/profile.d	目录下存放 Bash 相关的配置文件，相当于全局的.bashrc
/etc/profile.d/custom.sh	我在配置全局环境时，一般写入这个文件；如果不存在，可以新建。
/home	家目录，默认新建用户的个人家目录都在此文件夹下
/home/ct	用户名为 ct 的用户的家目录
/lib -> usr/lib	存放动态库的目录 (library)，安装软件时碰到依赖的动态库一般存储于此
/lib64 -> usr/lib64	64 位软件动态库，-> 表示软连接，等同于快捷方式
/mnt	文件系统挂载，一般插入 U 盘会显示在这。
/opt	部分额外安装的软件会置于此
/root	根用户的家目录
/sbin -> usr/sbin	根用户的管理命令
/tmp	临时目录，会定时清空，常用于存放中间文件
/usr	存放系统应用的目录，前面有几个目录都是该目录下子目录的软链
/usr/bin	大部分应用程序安装于此
/usr/sbin	根用户的管理命令
/usr/lib	存放动态库的目录 (library)，安装软件时碰到依赖的动态库一般存储于此
/usr/lib64	64 位软件动态库
/usr/local/bin	存放本地安装的命令
/usr/local/lib	存放本地安装的库
/var	存放各服务的日志文件。若装有网络服务，一般在/var/www/html 下。

```
/dev/mapper/a          37T   12T   25T  32% /ehbio1
/dev/mapper/b          37T   28T  8.8T  76% /ehbio2
/dev/mapper/c          37T   15T   23T  40% /ehbio3
```

除了看硬盘，还想看下 CPU、内存、操作系统呢？

```
# serverInfo.sh 是我写的一个脚本，这个脚本怎么实现的会是一个考核题目。
ct@ehbio:~$ serverInfo.sh


Hostname is localhost.localdomain,Ip address is 192.168.1.30.

The 64 bit operating system is  CentOS release 6.9 (Final),

Nuclear info is 2.6.32-696.10.1.el6.x86_64.

The CPU is Intel(R) Xeon(R) CPU E9-5799 v2 @ 3.90GHz.

There are 8 physical cpu, each physical cpu has 0 cores, 0 threads.

There are 96 logical cpu.

The memory of this server is 252G.
```

1.1.8 目录内容查看

通常登陆后直接进入家目录，下面大部分操作也是在家目录下完成的。如果想查看当前目录下都有什么内容，输入命令 `ls`，回车即可 (ls 可以理解为单词 list 的缩写)。当前目录下什么也没有，所以没有任何输出。

```
ct@ehbio:~$ ls
```

如果错把 `l` 看成了 `i`，输入了 `is`，则会出现下面的提示未找到命令。如果输入的是 Linux 基本命令，出现这个提示，基本可以判定是命令输入错了，瞪大眼睛仔细看就是了。在敲完命令回车后，注意查看终端的输出，以判断是否有问题。

```
ct@ehbio:~$ is
-bash: is: 未找到命令
# 大小写敏感
ct@ehbio:~$ lS
-bash: lS: 未找到命令
```

当前目录下只有一个文件，看不出效果，我们可以新建几个文件和文件夹。

1.1.9 新建目录

`mkdir` 是新建一个目录 (make a directory)；`data` 是目录的名字。如果目录存在，则会出现提示，“无法创建已存在的目录”。这时可以使用参数`-p` 忽略这个错误。

```
ct@ehbio:~$ mkdir data
ct@ehbio:~$ ls
data
ct@ehbio:~$ mkdir data
mkdir: 无法创建目录"data" : 文件已存在

# -p: no error if existing, make parent directories as needed
ct@ehbio:~$ mkdir -p data
```

`cat` 是一个命令，主要用来查看文件；在这与 `<<END` 连用用于读入大段数据。输入 `cat <<END` 之后，回车，会看到终端出现一个大于号，大于号后面可以输入内容，再回车，继续输入内容，直到我们输入 `END` (大写的，与上面一致)，输入过程结束，我们输入的内容都显示在了屏幕上。

```
ct@ehbio:~$ mkdir data
ct@ehbio:~$ cat <<END
a
bc
END
a
bc
```

如果我们想把这些内容写入文件，就需要使用 `command > filename` 格式。

`>` 是一个重定向符号，即把前面命令的输出写入到 `>` 后面的文件中。如下所示，新建了一个 `Fasta` 格式的文件。

`ls -l` 列出文件的详细信息；`-l` 表示命令行参数，是程序预留的一些选项，保证在不更改程序的情况下获得更灵活的操作。可使用 `man ls` 查看 `ls` 所有的命令行参数, 上下箭头翻页，按 `q` 退出查看。(man: manual, 手册)

```
ct@ehbio:~$ cat <<END >data/test.fa
>SOX2
ACGTCGGCGGAGGGTGGSCGGGGGGGGAGAGGT
ACGATGAGGAGTAGGAGAGAGGAGG
>OCT4
ACGTAGGATGGAGGAGAGGGAGGGGGGAGGAGAGGAA
AGAGTAGAGAGA
>NANOG
ACGATGCGATGCAGCGTTTTTTTTTGGTTGGATCT
CAGGTAGGAGCGAGGAGGCAGCGGCGGATGCAGGCA
ACGGTAGCGAGTC
```

```
>mYC HAHA
ACGGAGCGAGCTAGTGCAGCGAGGAGCTGAGTCGAGC
CAGGACAGGAGCTA
end
END

## 注意命令和参数之间的空格
ct@ehbio:~/data$ ls-l
-bash: ls-l: 未找到命令

ct@ehbio:~$ ls -l
总用量 4
## d: dir; 表示 data 是个目录
## rwx：表示目录的权限，暂时忽略，或自己在线搜索
drwxrwxr-x 2 ct ct 4096 6 月   8 14:52 data

ct@ehbio:~$ ls -l data
总用量 4
## 开头的`-`表示 test.fa 是个文件
-rw-rw-r-- 1 ct ct 284 6 月   8 14:48 test.fa
```

1.1.10 访问文件

查看写入的文件的内容，cat 文件名；需要注意的是文件所在的目录，默认是当前目录；如下面第一个命令，会提示 cat: test.fa: 没有那个文件或目录，是因为当前目录下不存在文件 test.fa。(注意文件末尾的 end)

```
# 这个应该是最常见的错误之一，程序不可能知道你的输入文件在什么地方，
# 需要人为指定。
# 如果未指定路径，表示当前目录
ct@ehbio:~$ cat test.fa
cat: test.fa: 没有那个文件或目录

ct@ehbio:~$ cat data/test.fa
>SOX2
ACGTCGGCGGAGGGTGGSCGGGGGGGGAGAGGT
ACGATGAGGAGTAGGAGAGAGGAGG
>OCT4
ACGTAGGATGGAGGAGAGGGAGGGGGGAGGAGAGGAA
AGAGTAGAGAGA
>NANOG
ACGATGCGATGCAGCGTTTTTTTTTGGTTGGATCT
CAGGTAGGAGCGAGGAGGCAGCGGCGGATGCAGGCA
ACGGTAGCGAGTC
>mYC HAHA
```

```
ACGGAGCGAGCTAGTGCAGCGAGGAGCTGAGTCGAGC
CAGGACAGGAGCTA
end
```

`test.fa` 在目录 `data` 下，可以先进入 `data` 目录，然后再查看文件。类比于 Windows 下先点开一个文件夹，再点开下面的文件。

这个例子中文件 `test.fa` 在当前目录的子目录 `data` 里面，在当前目录下直接查看 `test.fa` 就像在我的电脑里面不进入 C 盘，就像打开 `Program file` 文件夹。这属于隔空打牛的境界，不是一般人能练就的。起码 Linux 下不可以。

提到目录，Linux 下有绝对路径和相对路径的概念。

- 绝对路径：以/开头的路径为绝对路径，如`/home/ct`, `/usr/bin`, `/home/ct/data` 等。需要注意的是 `~/data` 等同于`/home/ct/data`, 多数情况下可以等同于绝对路径，但在一个情况下例外，软件安装时用于`--prefix` 后的路径必须是/开头的绝对路径。
- 相对路径: 不以/和 `~` 开头的路径都是相对路径，如 `data` 表示当前目录下的 data 目录，等同于`./data` (`.` 为当前目录), `../data` 表示当前目录的上一层目录下的 data 目录 (`../`表示上层目录)。

`pwd` (print working directory) 获取当前工作目录。

`cd` (change dir) 切换目录。若 `cd` 后没有指定切换到那个目录，则调回家目录。特别地，`cd -`表示返回到最近的 `cd` 操作前所在目录，相当于回撤到上一个工作目录。

`head` 查看文件最开始的几行，默认为 10 行，可使用`-n 6` 指定查看前 6 行。

```
# 记住输出
ct@ehbio:~$ pwd
/home/ct

ct@ehbio:~$ cd data

# 注意输出变化
ct@ehbio:~$ pwd
/home/ct/data
ct@ehbio:~/data$ head -n 6 test.fa
>SOX2
ACGTCGGCGGAGGGTGGSCGGGGGGGGAGAGGT
ACGATGAGGAGTAGGAGAGAGGAGG
>OCT4
ACGTAGGATGGAGGAGAGGGAGGGGGGAGGAGAGGAA
AGAGTAGAGAGA
```

另外 `less` 和 `more` 也可以用来查看文件，尤其是文件内容特别多的时候。

```
ct@ehbio:~/data$ less test.fa
# q: 退出
# 上下箭头、空格翻页
```

1.1.11 获取可用命令行参数

前面使用的命令，有几个用到了参数如 `ls -l`, `head -n 6` 等，需要注意的是命令跟参数之间要有空格。

终端运行 `man ls` 可以查看 `ls` 所有可用的参数，上下箭头翻页，按 `q` 退出查看。(man: manual, 手册)

```
ct@ehbio:~/data$ man ls
NAME
 ls - list directory contents

SYNOPSIS
 ls [OPTION]... [FILE]...

DESCRIPTION
 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
  do not ignore entries starting with .

 -A, --almost-all
  do not list implied . and ..

 --author
  with -l, print the author of each file

 -b, --escape
  print C-style escapes for nongraphic characters
 ....
```

1.1.12 小结

1. Linux 是多用户操作系统。这一点可以从我们每个人能同时登录到同一台 Linux 电脑各自进行操作而不相互干扰可以体会出来。大家可以尝试下是否可以看到其它人家目录下的东西。我们后期在权限管理部分会涉及如何开放或限制自

己的文件被他人访问。

2. Linux 下所有目录都在根目录下。根目录某种程度上可类比于 Windows 的我的电脑，第一级子目录类比于 `C` 盘，`D` 盘等 (等我们熟练了，就忘记这个拙劣的类比吧)。
3. 使用 `mkdir` 新建目录，`cd` 切换目录，`pwd` 获取当前工作目录，`ls` 查看目录下的内容, `cat` 查看文件，`man ls` 查看 ls 命令的使用。
4. 访问一个文件需要指定这个文件的路径，当前目录下的文件可省略其路径`./`，其它目录下则需要指定全路径 (可以是相对路径，也可以是绝对路径)。

1.1.13 练习题

1. 在家目录下新建文件夹 `bin` 和 `soft`。
2. 在 `bin` 和 `soft` 目录下各自新建一个文件，名字都是 `README`。
3. 在 `bin/README` 文件中写入内容：This folder is used to save executable files。
4. 在 `soft/README` 文件中写入内容：This folder is used to save software source files。
5. 查看两个 `README` 文件的大小。

1.2 Linux 下文件操作

1.2.1 文件上下翻转和左右翻转

两个有意思的命令，`tac`: 文件翻转，第一行变为最后一行，第二行变为倒数第二行；`rev` 每列反转，第一个字符变为最后一个字符，第二个字符变为倒数第二个字符。

```
ct@ehbio:~/data$ cat <<END | tac
> first
> second
> third
> END
third
second
first

ct@ehbio:~/data$ cat <<END | rev
> abcde
> xyz
> END
edcba
zyx
```

1.2.2 其它新建文件的方式

`nano` 类似于 Windows 下记事本的功能，`nano filename` 就可以新建一个文件，并在里面写内容；`ctrl+x` 退出，根据提示按 `Y` 保存。

`vim` 功能更强大的文本编辑器。`vim filename` 就可以新建一个文件, 敲击键盘字母 `i`，进入写作模式。写完后，敲击键盘 `Esc`, 退出写作模式，然后输入`:w` (会显示在屏幕左下角)，回车保存。`vim` 的常用方法，后面会有单独介绍。

1.2.3 文件拷贝、移动、重命名

常用的文件操作有移动文件到另一个文件夹、复制文件到另一个文件夹、文件重命名等。

`cp` (copy): 拷贝文件或文件夹 (`cp -r` 拷贝文件夹时的参数，递归拷贝).

`cp source1 source2 ... target_dir` 将一个或多个源文件或者目录复制到已经存在的目标目录。

`cp` 常用参数

-r: 递归拷贝

-f: 强制覆盖

-i: 覆盖前先询问

-p: 保留文件或目录的属性，主要是时间戳

-b: 备份复制，若目标文件存在，先备份之前的，再把新的覆盖过去

-u: 更新复制，若源文件和目标文件都存在，只在源文件的修改时间比较新时才复制

```
# 列出当前目录下有的文件和文件夹
ct@ehbio:~$ ls
data

# 新建一个文件夹
ct@ehbio:~$ mkdir ehbio_project
```

```
# 列出当前目录下有的文件和文件夹，及其子文件夹的内容
# data 目录下有一个文件，ehbio_project 目录下无文件
ct@ehbio:~$ ls *
data:
test.fa

ehbio_project:

# 拷贝 data 目录下的文件 test.fa 到 ehbio_project 目录下
ct@ehbio:~$ cp data/test.fa ehbio_project/

# 列出当前目录下有的文件和文件夹，及其子文件夹的内容
# data 目录下有一个文件，ehbio_project 目录下无文件
ct@ehbio:~$ ls *
data:
test.fa

ehbio_project:
test.fa
```

mv (move): 移动文件或文件夹

mv source target, 常用参数有

-f: 强制覆盖

-i: 覆盖前询问

-u: 更新移动

```
# 重命名 data 目录下的文件 test.fa 为 first.fa
# mv 除了可以移动文件，也可以做单个文件的重命名
ct@ehbio:~$ mv data/test.fa data/first.fa

# 列出当前目录下有的文件和文件夹，  及其子文件夹的内容
ct@ehbio:~$ ls *
data:
first.fa

ehbio_project:
test.fa
```

`rename`: 文件重命名 (常用于批量重命名，不同的系统可能用法略有不同，使用前先 `man rename` 查看使用方法)

```
# 进入另一个目录
ct@ehbio:~$ cd ehbio_project/
ct@ehbio:~/ehbio_project$ ls
test.fa

# 给文件做一份拷贝
ct@ehbio:~/ehbio_project$ cp test.fa second.fa
ct@ehbio:~/ehbio_project$ ls
second.fa  test.fa

# 给文件多拷贝几次，无聊的操作，就是为了给 rename 提供发挥作用的机会
ct@ehbio:~/ehbio_project$ cp test.fa test2.fa
ct@ehbio:~/ehbio_project$ cp test.fa test3.fa
ct@ehbio:~/ehbio_project$ cp test.fa test4.fa

# cp 后面需要 2 个参数，被拷贝的文件和要被拷贝到的目录或文件
# 出现下面的错误，表示缺少目标路径或文件
ct@ehbio:~/ehbio_project$ cp ehbio.fa
cp: 在" ehbio.fa"  后缺少了要操作的目标文件
Try 'cp --help' for more information.

ct@ehbio:~/ehbio_project$ ls
second.fa  test2.fa  test3.fa  test4.fa  test.fa

# 用 rename 进行文件批量重命名
ct@ehbio:~/ehbio_project$ rename 'test' 'ehbio' test*.fa
ct@ehbio:~/ehbio_project$ ls
ehbio2.fa  ehbio3.fa  ehbio4.fa  ehbio.fa  second.fa
```

`ln` (link): 给文件建立快捷方式 (`ln -s source_file target` 创建软连接)。

在建立软连接时，原文件要使用全路径。全路径指以/开头的路径。如果希望软链可以让不同的用户访问，不要使用 `~`。

建立软连接，是为了在不增加硬盘存储的情况下，简化文件访问方式的一个办法。把其它文件夹下的文件链接到当前目录，使用时只需要写文件的名字就可以了，不需要再写长串的目录了。

`ln` 命令常用参数

-s: 软连接

-f: 强制创建

../: 表示上一层目录；../../: 表示上面两层目录

pwd (print current/working directory): 输出当前所在的目录

\``为键盘 Esc 下第一个按键 （与家目录 ~‘符号同一个键)，写在反引号内的命令会被运行，运行结果会放置在反引号所在的位置

```
# 建立软连接，把当前目录下的 ehbio2.fa，链接到上一层目录的 data 下面

# 这是一个无效的软连接，
ct@ehbio:~/ehbio_project$ ln -s ehbio2.fa ../data

# 在使用 ls 查看时，无效的软连接的文件名下面是黑色的背景。
# 不同的终端配色方案会有不同，一般一直闪烁表示是失效的链接，
# 另外是否失效以能否使用为最终判断标准。
ct@ehbio:~/ehbio_project$ ls -l ../data/
总用量 4
lrwxrwxrwx 1 ct ct   9 6 月   9 17:55 ehbio2.fa -> ehbio2.fa
-rw-rw-r-- 1 ct ct 284 6 月   8 14:48 first.fa

# 输出当前所在的目录
ct@ehbio:~/ehbio_project$ pwd
/home/ct/ehbio_project

# 建立软连接时，原始文件一定使用全路径。全路径指以/开头的路径。
ct@ehbio:~/ehbio_project$ ln -s /home/ct/ehbio_project/ehbio2.fa ../data
ln: 无法创建符号链接" ../data/ehbio2.fa" : 文件已存在

# 上面的错误信息时，已经存在这么一个链接了（虽然是无效的），但再建新的链接时还会提示
# 使用 `-f` (force) 强制覆盖已有的链接
ct@ehbio:~/ehbio_project$ ln -fs /home/ct/ehbio_project/ehbio2.fa ../data

# 再次查看时，就正常了。文件名下面没有了恐怖的背景色，并且有个右箭头指向原始文件
# `lrwxrwxrwx`中的 `l`表示软连接。
ct@ehbio:~/ehbio_project$ ls -l ../data/
总用量 4
lrwxrwxrwx 1 ct ct  32 6 月   9 17:56 ehbio2.fa -> /home/ct/ehbio_project/ehbio2.fa
-rw-rw-r-- 1 ct ct 284 6 月   8 14:48 first.fa

# 通常为了简化写法，使用 `pwd`代替全路径
# `为键盘 Esc 下面的按键，写在反引号内的命令会被运行，运行结果会放置在反引号所在的位置
ct@ehbio:~/ehbio_project$ ln -s `pwd`/ehbio2.fa ../data
ln: 无法创建符号链接" ../data/ehbio2.fa" : 文件已存在
ct@ehbio:~/ehbio_project$ ln -fs `pwd`/ehbio2.fa ../data
ct@ehbio:~/ehbio_project$ ls -l ../data/
总用量 4
```

```
lrwxrwxrwx 1 ct ct  32 6 月   9 17:56 ehbio2.fa -> /home/ct/ehbio_project/ehbio2.fa
-rw-rw-r-- 1 ct ct 284 6 月   8 14:48 first.fa
```

复制、移动或创建软连接时，如果目标已存在，除了使用`-f`强制覆盖外，还可以使用 `rm` 命令删除。

`rm` 可以删除一个或多个文件和目录，也可以递归删除所有子目录，使用时一定要慎重。`rm` 命令删除的文件很难恢复。

`rm -rf *`: 可以删除当前目录下所有文件和文件夹，慎用。

`rm` 命令常见参数：

-f：强制删除

-i: 删除前询问是否删除

-r: 递归删除


```
ct@ehbio:~/ehbio_project$ ln -s `pwd`/ehbio2.fa ../data
ln: 无法创建符号链接" ../data/ehbio2.fa" : 文件已存在

# 删除之前的软连接，源文件不会被删除
ct@ehbio:~/ehbio_project$ rm -f ../data/ehbio2.fa

# 再次新建软连接
ct@ehbio:~/ehbio_project$ ln -s `pwd`/ehbio2.fa ../data
ct@ehbio:~/ehbio_project$ ls -l ../data/
总用量 4
lrwxrwxrwx 1 ct ct  32 6 月   9 17:56 ehbio2.fa -> /home/ct/ehbio_project/ehbio2.fa
-rw-rw-r-- 1 ct ct 284 6 月   8 14:48 first.fa

#
ct@ehbio:~/ehbio_project$ mkdir tmp

# touch filename: 如果文件不存在，则新建一个文件
#                 若存在，则更新其修改和访问时间
#                 在后面的搜索讲解中会有 touch 的一个妙用
ct@ehbio:~/ehbio_project$ touch tmp/a
ct@ehbio:~/ehbio_project$ touch tmp/b

# -r: 递归删除，主要用于删除目录和子目录时
# -f: 强制删除
ct@ehbio:~/ehbio_project$ rm -rf tmp
```

1.2.4 Linux 下命令的一些异常情况

命令不全：在命令没有输入完 (引号或括号没有配对)，就不小心按下了 Enter 键，终端会提示出一个 > 代表命令不完整，这是可以继续输入，也可以 ctrl+c 终止输入，重新再来。(下面 sed 命令使用时，还有另外一种命令不全的问题)

```
ct@ehbio:~/ehbio_project$ rename 'ehbio2
>'
ct@ehbio:~/ehbio_project$ rename 'ehbio2
> ^C
ct@ehbio:~/ehbio_project$
```

文件名输入错误: 多一个字母、少一个字母、大小写问题

```
ct@ehbio:~/ehbio_project$ls
ehbio2.fa  ehbio3.fa  ehbio4.fa  ehbio.fa  second.fa

# 重命名没有生效
ct@ehbio:~/ehbio_project$ rename 'ehbio2' 'ehbio5' ebio2.fa
ct@ehbio:~/ehbio_project$ ls
ehbio2.fa  ehbio3.fa  ehbio4.fa  ehbio.fa  second.fa

# 仔细看是 ehbio2.fa 写成了 ebio2.fa，更正后即可。
Z8vb3e9jtel4m99ss6e7eZ:~/ehbio_project$ rename 'ehbio2' 'ehbio5' ehbio2.fa
ct@ehbio:~/ehbio_project$ ls
ehbio3.fa  ehbio4.fa  ehbio5.fa  ehbio.fa  second.fa
```

所在目录不对: 访问的文件不存在于当前目录，而又没有提供绝对路径, 或软连接失效

```
ct@ehbio:~/ehbio_project$ ls
ehbio3.fa  ehbio4.fa  ehbio5.fa  ehbio6.fa  ehbio.fa  second.fa
ct@ehbio:~/ehbio_project$ ls ../data
ehbio2.fa  first.fa

# 当前目录没有 ehbio2.fa
ct@ehbio:~/ehbio_project$ less ehbio2.fa
ehbio2.fa: 没有那个文件或目录

# ehbio2.fa 在上一层目录的 data 目录下
ct@ehbio:~/ehbio_project$ ls ../data/ehbio2.fa
../data/ehbio2.fa

# 加上路径依然访问不了
ct@ehbio:~/ehbio_project$ less ../data/ehbio2.fa
../data/ehbio2.fa: 没有那个文件或目录
```

```
# 上面的问题是软连接失效，在之前的操作中删掉了原始的 ehbio2.fa，所以快捷方式失效

# 正确的访问
ct@ehbio:~/ehbio_project$ tail -n 3 ../data/first.fa
ACGGAGCGAGCTAGTGCAGCGAGGAGCTGAGTCGAGC
CAGGACAGGAGCTA
end
```

1.2.5 Linux 下文件内容操作

常用的文件内容操作有文件压缩解压缩、文件大小行数统计、文件内容查询等。

`gzip`: 压缩文件; `gunzip`: 解压缩文件

```
# gzip -c 把压缩的文件输出到标准输出（一般是屏幕）
# '>' 输出重定向，输出写入文件

ct@ehbio:~/ehbio_project$ gzip -c ehbio.fa >ehbio.fa.gz

# 多了一个.gz 文件
ct@ehbio:~/ehbio_project$ ls
ehbio3.fa  ehbio4.fa  ehbio5.fa  ehbio.fa  ehbio.fa.gz  second.fa

# 解压缩
ct@ehbio:~/ehbio_project$ gunzip ehbio.fa.gz
gzip: ehbio.fa already exists; do you wish to overwrite (y or n)? y

ct@ehbio:~/ehbio_project$ ls
ehbio3.fa  ehbio4.fa  ehbio5.fa  ehbio.fa  second.fa
```

`wc` (word count): 一般使用 `wc -l` 获取文件的行数。

```
# 输出文件有 14 行
ct@ehbio:~/ehbio_project$ wc -l ehbio.fa
14 ehbio.fa
```

获取文件中包含大于号 (>) 的行, `grep` (print lines matching a pattern，对每一行进行模式匹配)。

`grep` 的用法很多，支持正则表达式匹配，这个后面我们会详细讲解。

```
ct@ehbio:~/ehbio_project$ grep '>' ehbio.fa
>SOX2
>OCT4
>NANOG
>mYC HAHA

# 获取包含> 的行的行数 (-c: count lines)
ct@ehbio:~/ehbio_project$ grep -c '>' ehbio.fa
4

# 是不是还记得当时新建文件时，末尾多了一行 end，删除 end 所在行
ct@ehbio:~/ehbio_project$ less ehbio.fa

# -v: 不输出匹配上的行
ct@ehbio:~/ehbio_project$ grep -v 'end' ehbio.fa >ehbio6.fa
ct@ehbio:~/ehbio_project$ cat ehbio6.fa
>SOX2
ACGTCGGCGGAGGGTGGSCGGGGGGGGAGAGGT
ACGATGAGGAGTAGGAGAGAGGAGG
>OCT4
ACGTAGGATGGAGGAGAGGGAGGGGGGAGGAGAGGAA
AGAGTAGAGAGA
>NANOG
ACGATGCGATGCAGCGTTTTTTTTTGGTTGGATCT
CAGGTAGGAGCGAGGAGGCAGCGGCGGATGCAGGCA
ACGGTAGCGAGTC
>mYC HAHA
ACGGAGCGAGCTAGTGCAGCGAGGAGCTGAGTCGAGC
CAGGACAGGAGCTA
```

替换文件中的字符: sed 是一个功能强大的文件内容编辑工具，常用于替换、取得行号等操作。现在先有个认识，后面会详细介绍。

| 为管道符，在相邻命令之间传递数据流，表示把上一个命令的输出作为下一个命令的输入。

```
# 第一个错误，漏掉了文件名
# 程序静止在这，等待用户的进一步输入
# ctrl+c 杀掉当前命令
ct@ehbio:~/ehbio_project$ sed 's/ HAHA//' | tail -n 3

^C

# 第二个错误，文件名和单引号之间没有空格，使得 sed 判断命令错误

ct@ehbio:~/ehbio_project$ sed 's/ HAHA//'ehbio.fa  | tail -n 3
```

```
sed：-e 表达式 #1，字符 11:"s"的未知选项

# 正确操作，

ct@ehbio:~/ehbio_project$ sed 's/ HAHA//' ehbio.fa  | tail -n 4
>mYC
ACGGAGCGAGCTAGTGCAGCGAGGAGCTGAGTCGAGC
CAGGACAGGAGCTA
end
```

另外一个方式，去除 `HAHA`，使用 `cut` 命令。cut 更适合于矩阵操作，去除其中的一列或者多列。但在处理 FASTA 格式文件时有这么一个妙用。FASTA 文件中序列里面是没有任何符号的，而如果名字比较长，则可以指定相应分隔符就行 `cut`，这样既处理了名字，又保留了序列。

`-f`: 指定取出哪一列，使用方法为`-f 2` (取出第 2 列)，`-f 2-5` (取出第 2-5 列)，`-f 2,5` (取出第 2 和第 5 列)。注意不同符号之间的区别。

`-d`: 设定分隔符, 默认为 TAB 键。如果某一行没有指定的分隔符，整行都为第一列。

```
ct@ehbio:~/ehbio_project$ cut -f 1 -d ' ' ehbio.fa | tail -n 4
>mYC
ACGGAGCGAGCTAGTGCAGCGAGGAGCTGAGTCGAGC
CAGGACAGGAGCTA
end
```

1.2.6 小结和练习

1. Linux 下文件拷贝、移动、重命名、软连接、压缩、替换等操作。
2. Linux 下常见问题

- `ehbio2.fa`: 没有那个文件或目录：这个错误通常由什么引起，应该怎么解决？
- 若文件 `a` 存在，运行 `ln a data/b` 能否成功给 `a` 建立软连接？
- `grep '> ehbio.fa` 的输出是什么？

3. 若目标文件存在时，再运行 `cp`, `mv` 或 `ln` 会有什么提示？
4. 计算某一个 Fasta 序列中序列的个数。

1.3 Linux 终端常用快捷操作

- 命令或文件名自动补全：在输入命令或文件名的前几个字母后，按 Tab 键，系统会自动补全或提示补全
- 上下箭头：使用上下箭头可以回溯之前的命令，增加命令的重用，减少输入工作量
- ！加之前输入过的命令的前几个字母，快速获取前面的命令

```
ct@ehbio:~/ehbio_project$ cut -f 1 -d ' ' ehbio.fa | tail -n 4
>mYC
ACGGAGCGAGCTAGTGCAGCGAGGAGCTGAGTCGAGC
CAGGACAGGAGCTA
end
ct@ehbio:~/ehbio_project$ man cut
# 直接跳到上面运行的 cut 命令，再执行一次
ct@ehbio:~/ehbio_project$ !cut
cut -f 1 -d ' ' ehbio.fa | tail -n 4
>mYC
ACGGAGCGAGCTAGTGCAGCGAGGAGCTGAGTCGAGC
CAGGACAGGAGCTA
end
```

- ctrl+a 回到命令的行首，ctrl+e 到命令行尾，(home 和 end 也有类似功能)，用于修改命令或注释掉命令

```
# 写完下面的命令，突然不想运行了，又不想一个个删掉
ct@ehbio:~/ehbio_project$ cut -f 1 -d ' ' ehbio.fa | tail -n 4

# 按 ctrl+a, 回到行首，再输入 `#`号，回车，命令即被注释掉。
ct@ehbio:~/ehbio_project$ #cut -f 1 -d ' ' ehbio.fa | tail -n 4
```

- !! 表示上一条命令。

```
ct@ehbio:~/ehbio_project$ ls
ehbio3.fa  ehbio4.fa  ehbio5.fa  ehbio6.fa  ehbio.fa  second.fa
ct@ehbio:~/ehbio_project$ !!
ls
ehbio3.fa  ehbio4.fa  ehbio5.fa  ehbio6.fa  ehbio.fa  second.fa
```

- 替换上一个命令中的字符，再运行一遍命令，用于需要对多个文件执行同样的命令，又不想写循环的情况

```
# 输入一个命令
ct@ehbio:~/ehbio_project$ #cut -f 1 -d ' ' ehbio.fa | tail -n 4

# !! 表示上一条命令
```

```
# :gs 表示替换，把上一个命令中全部的 ehbio 替换为 ehbio3; g: global; s: substitute
ct@ehbio:~/ehbio_project$ !!:gs/ehbio/ehbio3
#cut -f 1 -d ' ' ehbio3.fa | tail -n 4

# 替换后效果如上

# 去掉命令前的 # 号
ct@ehbio:~/ehbio_project$ cut -f 1 -d ' ' ehbio3.fa | tail -n 4
>mYC
ACGGAGCGAGCTAGTGCAGCGAGGAGCTGAGTCGAGC
CAGGACAGGAGCTA
end

## 替换 ehbio3 为 ehbio4，直接运行命令
ct@ehbio:~/ehbio_project$ !!:gs/ehbio3/ehbio4
cut -f 1 -d ' ' ehbio4.fa | tail -n 4
>mYC
ACGGAGCGAGCTAGTGCAGCGAGGAGCTGAGTCGAGC
CAGGACAGGAGCTA
end
```

1.4 Linux 下的标准输入、输出、重定向、管道

在 Linux 系统中，有 4 个特殊的符号，<, >, |, -，在我们处理输入和输出时存在重要但具有迷惑性的作用。

默认 Linux 的命令的结果都是输出到标准输出，错误信息 (比如命令未找到或文件格式识别错误等) 输出到标准错误，而标准输出和标准错误默认都会显示到屏幕上。

> 表示重定向标准输出，`> filename` 就是把标准输出存储到文件 filename 里面。标准错误还是会显示在屏幕上。

`2 >&1` 表示把标准错误重定向到标准输出。Linux 终端用 `2` 表示标准错误，`1` 表示标准输出。

- (短横线)：表示标准输入，一般用于 1 个程序需要多个输入的时候。

< 标准输入，后面可以跟可以产生输出的命令，一般用于 1 个程序需要多个输入的时候。相比-适用范围更广。

| 管道符，表示把前一个命令的输出作为后一个命令的输入，前面也有一些展示例子。用于数据在不同的命令之间传输，用途是减少硬盘存取损耗。

下面我们通过一个程序 `stdout_error.sh` 来解释上面的文字 (Bash 脚本写作，后面会有专门介绍)，内容如下

```
#!/bin/bash

echo "I am std output"
# 下面是随便写的一个理论上不存在的命令，理论上会报错的。
unexisted_command
```

运行这个脚本

```
# 标准输出和标准错误默认都会显示到屏幕上
ct@ehbio:~$ bash stdout_error.sh
I am std output
stdout_error.sh: line 5: unexisted_command: command not found
# 上一行的 line 5 表示是第 5 行命令出错了，通常用来调试程序。
# 在这个例子中，bash 脚本第 5 行是故意输入的一个错误命令，用来展示什么是标准错误

# > 把结果输入到了文件；标准错误还显示在屏幕上
ct@ehbio:~$ bash stdout_error.sh >stdout_error.stdout
stdout_error.sh: line 5: unexisted_command: command not found
ct@ehbio:~$ cat stdout_error.stdout
I am std output

# > 把结果输入到了文件; 2> 把标准错误输入到了另一个文件
ct@ehbio:~$ bash stdout_error.sh >stdout_error.stdout 2>stdout_error.stderr
ct@ehbio:~$ cat stdout_error.stderr
stdout_error.sh: line 5: unexisted_command: command not found

# 标准输出和标准错误写入同一个文件
ct@ehbio:~$ bash stdout_error.sh >stdout_error.stdout 2>&1
ct@ehbio:~$ cat stdout_error.stdout
I am std output
stdout_error.sh: line 5: unexisted_command: command not found
```

下面看管道符和标准输入的使用。

```
# 管道符的使用
# 第一个命令的输出作为第二个的输入
# 前面的例子中也有使用
# tr: 是用于替换字符的，把空格替换为换行，文字就从一行变为了一列
ct@ehbio:~$ echo "1 2 3" | tr ' ' '\n'
1
2
3

# cat 命令之前也用过，输出一段文字
# diff 是比较 2 个文件的差异的，需要 2 个参数
```

```
# - （短横线）表示上一个命令的输出，传递给 diff
# < 表示其后的命令的输出，也重定向给 diff
ct@ehbio:~$ cat <<END | diff - <(echo "1 2 3" | tr ' ' '\n')
> 2
> 3
> 4
> END
0a1
> 1
3d3
< 4

# 如果不使用管道和重定向标准输入，程序是这么写的

# 先把第一部分存储为 1 个文件
ct@ehbio:~$ cat <<END >firstfile
2
3
> 4
> END
ct@ehbio:~$ less firstfile

# 再把第二部分存储为 1 个文件
ct@ehbio:~$ echo "1 2 3" | tr ' ' '\n' >secondfile

# 然后比较
ct@ehbio:~$ diff firstfile secondfile
0a1
> 1
3d3
< 4
```

管道符的更多应用

```
ct@ehbio:~$ echo  "actg aaaaa cccccg" | tr ' ' '\n' | wc -l
3

# sed =：先输出行号，再输出每行的内容
ct@ehbio:~$ echo  "a b c" | tr ' ' '\n' | sed =
1
a
2
b
3
c
```

```
# 后面这个命令会略微难理解些
# sed = 同时输出行号
# N: 表示读入下一行；sed 命令每次只读一行，加上 N 之后就是缓存了第 2 行，所有的操作都针对第一行；
# s: 替换；把换行符替换为\t
ct@ehbio:~$ echo  "a b c" | tr ' ' '\n' | sed = | sed 'N;s/\n/\t/'
1   a
2   b
3   c

# 后面这个命令不太好解释
# sed = 同时输出行号
# N: 表示读入下一行；sed 命令每次只读一行，加上 N 之后就是缓存了第 2 行，所有的操作都针对第一行；
# s: 替换；把读取的奇数行行首加一个'>'(偶数行相当于被隐藏了)
ct@ehbio:~$ echo  "a b c" | tr ' ' '\n' | sed = | sed 'N;s/^/>/'
>1
a
>2
b
>3
c

# 把多条序列转成 FATSA 格式
# sed = 同时输出行号
# N: 表示读入下一行；sed 命令每次只读一行，加上 N 之后就是缓存了第 2 行，所有的操作都针对第一行；
# s: 替换；把读取的奇数行行首加一个'>'(偶数行相当于被隐藏了)
# 于是 FASTA 格式序列就出来了
ct@ehbio:~$ echo  "actg aaaaa cccccg" | tr ' ' '\n' | sed = | sed 'N;s/^/>/'
>1
actg
>2
aaaaa
>3
cccccg
```

1.5 Linux 文件内容操作

1.5.1 文件生成

seq: 产生一系列的数字; man seq 查看其具体使用。我们这使用 seq 产生下游分析所用到的输入文件。

```
# 产生从 1 到 10 的数，步长为 1
ct@ehbio:~$ seq 1 10
1
```

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

# 产生从 1 到 10 的数，步长为 1，用空格分割
ct@ehbio:~$ seq -s ' ' 1 10
1 2 3 4 5 6 7 8 9 10

# 产生从 1 到 10 的数，步长为 2
# 如果有 3 个数，中间的数为步长，最后一个始终为最大值
ct@ehbio:~$ seq -s ' ' 1 2 10
1 3 5 7 9

# 还记得前面提到的标准输入和标准输出吧
ct@ehbio:~$ cat <(seq 0 3 17) <(seq 3 6 18) >test
ct@ehbio:~$ cat test
0
3
6
9
12
15
3
9
15
```

1.5.2 文件排序

`sort`: 排序，默认按字符编码排序。如果想按数字大小排序，需添加`-n`参数。

`sort` 常用参数

-n: 数值排序

-h: 人类刻度的数值排序 (2K 1G 等)

-r: reverse, 逆序

-c: check, 不排序，查看文件是否已排序好

-k: 指定使用哪列或哪几列排序

-m: 合并已经排序好的文件

-S: 缓冲区大小，用于排序大文件时的分割排序中每次分割的文件大小

-u: 重复行只保留一次

```
# 系统默认按 ASCII 码排序，首先排 0，然后排 1, 3, 6, 9
ct@ehbio:~$ sort test
0
12
15
15
3
3
6
9
9

# 按数值大小排序
ct@ehbio:~$ sort -n test
0
3
3
6
9
9
12
15
15
```

`sort -u`: 去除重复的行，等同于 `sort | uniq`。

```
ct@ehbio:~$ sort -nu test
0
3
```

```
6
9
12
15
```

`sort file | uniq -d`: 获得重复的行。(d=duplication)

```
ct@ehbio:~$ sort -n test | uniq -d
3
9
15
```

`sort file | uniq -c`: 获得每行重复的次数。

```
# 第一列为每行出现的次数，第二列为原始的行
ct@ehbio:~$ sort -n test | uniq -c
  1 0
  2 3
  1 6
  2 9
  1 12
  2 15

# 换一个文件看的更清楚
ct@ehbio:~$ cat <<END >test2
a
b
c
b
a
e
d
a
END

# 第一列为每行出现的次数，第二列为原始的行
ct@ehbio:~$ sort test2 | uniq -c
3 a
2 b
1 c
1 d
1 e

# 在执行 uniq 操作前，文件要先排序，不然结果很诡异
ct@ehbio:~$ cat test2 | uniq -c
```

```
1 a
1 b
1 c
1 b
1 a
1 e
1 d
1 a
```

整理下 `uniq -c` 的结果，使得原始行在前，每行的计数在后。

`awk` 是一个强大的文本处理工具，其处理数据模式为按行处理。每次读入一行，进行操作。

- `OFS`: 输出文件的列分隔符 (output file column separtor)；
- `FS` 为输入文件的列分隔符 (默认为空白字符)；
- `awk` 中的列从第 1 到 n 列，分别记录为 `$1`, `$2` ... `$n`；
- `BEGIN` 表示在文件读取前先设置基本参数；与之相对应的是 `END`，只文件读取完成之后进行操作；
- 不以 `BEGIN`, `END` 开头的 `{}` 就是文件读取、处理的部分。每次对一行进行处理。后面会详细讲解。

```
# 管道符还记得吧
# awk 的操作就是读入上一步的结果，去除多余的空白，然后调换 2 列
#
ct@ehbio:~$ sort test2 | uniq -c | awk 'BEGIN{OFS="\t";}{print $2, $1}'
a    3
b    2
c    1
d    1
e    1
```

对两列文件，按照第二列进行排序, `sort -k2,2n`。

```
# 第二列按数值大小排序
ct@ehbio:~$ sort test2 | uniq -c | awk 'BEGIN{OFS="\t";}{print $2, $1}' | sort -k2, 2n
c    1
d    1
e    1
b    2
a    3

# 第二列按数值大小排序
# 第二列相同的再按第一列的字母顺序的逆序排序 (-r)
# 注意看前 3 行的顺序与上一步结果的差异
ct@ehbio:~$ sort test2 | uniq -c | awk 'BEGIN{OFS="\t";}{print $2,$1}' | sort -k2,2n -k1,1r
```

```
e    1
d    1
c    1
b    2
a    3
```

1.5.3 FASTA 序列提取

生成单行序列 FASTA 文件，提取特定基因的序列，最简单的是使用 `grep` 命令。

`grep` 在前面也提到过，以后还会经常提到，主要用途是匹配文件中的字符串，以此为基础，进行一系列的操作。如果会使用正则表达式，将会非常强大。正则表达式版本很多，几乎每种语言都有自己的规则，后面会详细展开。

```
# 生成单行序列 FASTA 文件
# 开头的大于号是为了还原命令的输入过程，可以复制下面 cat 出来的结果，方便输入。
ct@ehbio:~$ cat <<END >test.fasta
> >SOX2
> ACGAGGGACGCATCGGACGACTGCAGGACTGTC
> >POU5F1
> ACGAGGGACGCATCGGACGACTGCAGGACTGTC
> >NANOG
> CGGAAGGTAGTCGTCAGTGCAGCGAGTCCGT
> END
ct@ehbio:~$ cat test.fasta
>SOX2
ACGAGGGACGCATCGGACGACTGCAGGACTGTC
>POU5F1
ACGAGGGACGCATCGGACGACTGCAGGACTGTC
>NANOG
CGGAAGGTAGTCGTCAGTGCAGCGAGTCCGT

# grep 匹配含有 SOX2 的行
# -A 1 表示输出的行中，包含匹配行的下一行 (A: after)
ct@ehbio:~$ grep -A 1 'SOX2' test.fasta
>SOX2
ACGAGGGACGCATCGGACGACTGCAGGACTGTC

# 也可以使用 AWK
# 先判断当前行是不是 > 开头，如果是，表示是序列名字行，替换掉大于号，取出名字。
# sub 替换，sub(被替换的部分，要替换成的，待替换字符串)
# 如果不以大于号开头，则为序列行，存储起来。
# seq[name]: 相当于建一个字典，name 为 key，序列为值。然后就可以使用 name 调取序列。
# 若命令太长，可在末尾加一个 \, 换行书写
#
```

```
# awk 中$0 ~ />/ 里面的 ~ 不表示家目录，而是一个运算符，用来做模式匹配的
# /pattern/ 则表示与什么模式进行匹配，pattern 代表的是匹配模式
#
# awk 对文件是按行操作的，{} 里面的语句会对文件的每一行都进行判断或操作，循环执行
# $0: 表示当前行所有内容；$1, $2, $3 表示当前行第 1,2,3 列
#
# 关于引号，如果最外层用的是单引号，那么里面最好不要再出现单引号
# 如果最外面用的是双引号，则里面最好不要再出现单引号
# 命令会寻找最近的同样引号进行匹配。
ct@ehbio:~$ awk 'BEGIN{OFS=FS="\t"}{if($0~/>/) {name=$0; sub(">", "", name);} \
    else seq[name]=$0;}END{print ">SOX2"; print seq["SOX2"]}' test.fasta
>SOX2
ACGAGGGACGCATCGGACGACTGCAGGACTGTC
```

多行 FASTA 序列提取要麻烦些，一个办法就是转成单行序列，用上面的方式处理。

sed 和 tr 都为最常用的字符替换工具。

```
ct@ehbio:~$ cat <<END >test.fasta
> >SOX2
> ACGAGGGACGCATCGGACGACTGCAGGACTGTC
> ACGAGGGACGCATCGGACGACTGCAGGACTGTC
> ACGAGGGACGCATCGGACGACTGCAGGAC
> >POU5F1
> CGGAAGGTAGTCGTCAGTGCAGCGAGTCCGT
> CGGAAGGTAGTCGTCAGTGCAGCGAGTCC
> >NANOG
> ACGAGGGACGCATCGGACGACTGCAGGACTGTC
> ACGAGGGACGCATCGGACGACTGCAGG
> ACGAGGGACGCATCGGACGACTGCAGGACTGTC
> ACGAGGGACGCATCGGACGACTGCAGGACTGT
> END

# 新解法，更简单
ct@ehbio:~$ cat test.fasta | tr '\n' '\t' | sed 's/\t>/\n>/g' \
        | sed 's/\t/\n/' | sed 's/\t//g' >test.oneline.fa
# 分解来看
# 第一步所有行变为一行
#
# 这一步使用 tr 是因为 tr 里面可以直接识别换行符，而 sed 不可以
# 其它的替换都使用 sed
ct@ehbio:~$ cat test.fasta | tr '\n' '\t'
```

>SOX2 ACGAGGGACGCATCGGACGACTGCAGGACTGTC ACGAGGGACGCATCGGACGACTGCAGGACT-GTC ACGAGGGACGCATCGGACGACTGCAGGAC >POU5F1 CGGAAGGTAGTCGTCAGTGCAGC-

GAGTCCGT CGGAAGGTAGTCGTCAGTGCAGCGAGTCC >NANOG ACGAGGGACGCATCGGAC-GACTGCAGGACTGTC ACGAGGGACGCATCGGACGACTGCAGG ACGAGGGACGCATCGGACGACT-GCAGGACTGTC ACGAGGGACGCATCGGACGACTGCAGGACTGT

```
# >号前面加换行符
ct@ehbio:~$ cat test.fasta | tr '\n' '\t' | sed 's/\t>/\n>/g'
```

>SOX2 ACGAGGGACGCATCGGACGACTGCAGGACTGTC ACGAGGGACGCATCGGACGACTGCAGGACT-GTC ACGAGGGACGCATCGGACGACTGCAGGAC

>POU5F1 CGGAAGGTAGTCGTCAGTGCAGCGAGTCCGT CGGAAGGTAGTCGTCAGTGCAGCGAGTCC

>NANOG ACGAGGGACGCATCGGACGACTGCAGGACTGTC ACGAGGGACGCATCGGACGACTGCAGG ACGAGGGACGCATCGGACGACTGCAGGACTGTC ACGAGGGACGCATCGGACGACTGCAGGACTGT

```
# 先把第一个TAB键变为换行符，实现序列名字和序列的分离
# 再去掉序列中所有的TAB键
ct@ehbio:~$ cat test.fasta | tr '\n' '\t' | sed 's/\t>/\n>/g' \
        | sed 's/\t/\n/' | sed 's/\t//g' >test.oneline.fa
```

>SOX2

ACGAGGGACGCATCGGACGACTGCAGGACTGTCACGAGGGACGCATCGGACGACTGCAGGACTGTCACGAGGGACG

>POU5F1

CGGAAGGTAGTCGTCAGTGCAGCGAGTCCGTCGGAAGGTAGTCGTCAGTGCAGCGAGTCC

>NANOG

ACGAGGGACGCATCGGACGACTGCAGGACTGTCACGAGGGACGCATCGGACGACTGCAGGACGAGGGACGCATCGG

或者简单点，直接用前面的 `awk` 略微做下修改。

```
# 差别只在一点
# 对于单行 fasta 文件，只需要记录一行，seq[name]=$0
# 对于多好 fasta 文件，需要把每一行序列都加到前面的序列上，seq[name]=seq[name]$0
ct@ehbio:~$ awk 'BEGIN{OFS=FS="\t"}{if($0~/>/) {name=$0; sub(">", "", name);} \
    else seq[name]=seq[name]$0;}END{print ">SOX2"; print seq["SOX2"]}' test.fasta
>SOX2
ACGAGGGACGCATCGGACGACTGCAGGACTGTCACGAGGGACGCATCGGACGACTGCAGGACTGTCACGAGGGACGCATCGGACGACTGCAGGAC
```

1.6 命令运行监测

1. 监测命令的运行时间 time command

```
ct@ehbio:~$ time sleep 5

real    0m5.003s # 程序开始至结束的时间，包括其它进程占用的时间片和IO时间
user    0m0.001s # 进程真正执行占用CPU的时间，
sys 0m0.002s     # 进程在内核中调用所消耗的CPU时间
user+sys是进程实际的CPU时间。如果多线程执行，这个时间可能大于Real。如果IO是瓶颈，则real会大于us
```

2. 查看正在运行的命令和其资源使用 top

- top 输出界面第一行主要信息是负载显示，分别是 1 分钟、5 分钟、15 分钟前到现在的任务队列的平均长度。
- 一般与 CPU 数目相当为好，过大系统负载超额，反应慢。
- 在 top 输出界面输入 u, 会提示输入用户名，以查看某个用户的进程。
- 重点关注的是%MEM 列，查看系统占用的内存是否超出。

```
ct@ehbio:~$ top -a #按内存排序显示

top - 09:02:11 up 224 days,  8:34,  30 users,  load average: 40, 33, 28
Tasks: 1561 total,   1 running, 1550 sleeping,   0 stopped,  10 zombie
Cpu(s):  0.6%us,  0.2%sy,  0.0%ni, 99.2%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  2642768880k total, 2094619800k used, 548149080k free,   4310240k buffers
Swap: 86472700k total, 73226016k used, 13246684k free, 193383748k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
32527 ct        20   0 2631m 1.7g 1332 S  0.0  0.7 100:34.87 rsem-run-em
29273 ct        20   0 4094m 692m 3396 S  0.0  0.3  45:18.83 java -Xmx1000m
40148 mysql     20   0 21.9g 606m 6116 S  1.3  0.2   2536:06 /usr/sbin/mysqld
31040 ct        20   0 1887m  77m 2604 S  0.3  0.0 180:43.16 [celeryd:
```

3. 查看系统进程 `ps auwx | grep 'process_name'`

1.7 References

- 原文链接 http://blog.genesino.com//2017/06/bash1/
- 微信公众号 http://mp.weixin.qq.com/s/yKP1Kboji9N4p2SI1Ovj0Q
- Linux-总目录
- Linux-文件和目录
- Linux-文件操作
- Linux 文件内容操作
- Linux-环境变量和可执行属性
- Linux - 管道、标准输入输出
- Linux - 命令运行监测和软件安装
- Linux-常见错误和快捷操作
- Linux-文件列太多，很难识别想要的信息在哪列；别焦急，看这里。
- Linux-文件排序和 FASTA 文件操作
- Linux-应用 Docker 安装软件
- Linux 服务器数据定期同步和备份方式
- VIM 的强大文本处理方法
- Linux - Conda 软件安装方法
- 查看服务器配置信息
- Linux - SED 操作，awk 的姊妹篇
- Linux - 常用和不太常用的实用 awk 命令
- Bash 概论 - Linux 系列教程补充篇

2 Linux 下软件安装相关

视频课见 http://bioinfo.ke.qq.com。

2.1 文件属性和可执行属性

2.1.1 文件属性

文件属性 `rwx` 中 `r` 表示 `read` (数字表示为 4)、`w` 表示 `write` (数字表示为 2)、`x` 表示执行 (数字表示为 1)。三个为一组，连续出现三次 (如下面命令行中所示), 第一组表示文件的所有者拥有的权限，第二组为文件所有者所在的用户组所拥有的权限，组内所有成员都具有的权限，第三组为其它用户的权限。

`chmod` 可以修改文件或文件夹属性。

```
ct@ehbio:~$ ls -l /home
-rwxr-xr-x 1 ct ct 26 12 月  7 2016  ct

# 让自己的家目录只自己可见
ct@ehbio:~$ chmod go-rwx /home/ct
ct@ehbio:~$ ls -l /home
-rwx------ 1 ct ct 26 12 月  7 2016  ct

# 同组人增加可读和可执行属性
ct@ehbio:~$ chmod g+rx /home/ct
ct@ehbio:~$ ls -l /home
-rwxr-x--- 1 ct ct 26 12 月  7 2016  ct

# 所有人增加可读和可执行属性
ct@ehbio:~$ chmod a+rx /home/ct
ct@ehbio:~$ ls -l /home
-rwxr-xr-x 1 ct ct 26 12 月  7 2016  ct
```

2.1.2 可执行属性

Linux 下文件有一个特殊的属性即可执行属性，用来指示这个文件是一个可执行的脚本或可以运行的二进制文件。前面所提到的这些命令，都具有可执行属性。

`which`: 表示查看命令的路径。一般用于当我们想知道使用的命令来源于什么地方时，比如安装了多个 R 或多个 python，但又分不清用的是哪个时，`which` 一下，立即明了。在这儿我们用 which 获取的是可执行的命令所在的路径，进而查看

其属性。

```
ct@ehbio:~$ ls -l "`which cd`"
#rwx: 文件所有者可读、可写、可执行
#r-x: 文件所有者所在组其它成员可读、可执行，不可修改
#r-x: 其它人可读、可执行，不可修改
-rwxr-xr-x 1 root root 26 12 月  7 2016 /usr/bin/cd

ct@ehbio:~$ ls -l "`which mkdir`"
-rwxr-xr-x. 1 root root 79768 11 月  6 2016 /usr/bin/mkdir

ct@ehbio:~$ ls -l "`which python`"
#l: 代表软连接
# 软连接自身是所有人可读可写，但具体的权限依赖于其链接的文件
lrwxrwxrwx. 1 root root 7 3 月  22 15:04 /usr/bin/python -> python2

ct@ehbio:~$ ls -l "`which python2`"
# 第二层链接
lrwxrwxrwx. 1 root root 9 3 月  22 15:04 /usr/bin/python2 -> python2.7

# 链接的原始文件

ct@ehbio:~$ ls -l "`which python2.7`"
-rwxr-xr-x. 1 root root 7136 11 月  6 2016 /usr/bin/python2.7
```

`chmod a+x file`: 表示给文件增加所有人 (a) 可执行权限 (+x)

`chmod u+x file`: 表示给文件增加所有者 (u，user，) 可执行权限 (+x)

`chmod g+x, chmod o+x`: 表示给文件增加组内人或其它人可执行权限

`chmod 755 file`: 表示拥有者有可读写执行权限，其它人有可读执行权限。(`7=4+2+1; 5=4+1`)

具体使用 `man chmod` 查看其它参数使用。

```
# 新建个文件
ct@ehbio:~$ cat <<END >run.sh
> echo " I am a script created by ehbio."
> END

# 查看其权限值
ct@ehbio:~$ ls -l run.sh
-rw-rw-r-- 1 ct ct 39 6 月  14 23:12 run.sh

# 更改权限值
```

```
ct@ehbio:~$ chmod 755 run.sh

# 查看其权限值
# 注意多了 3 个 x
ct@ehbio:~$ ls -l run.sh
-rwxr-xr-x 1 ct ct 39 6 月  14 23:12 run.sh

# 去除其它用户的可执行权限
ct@ehbio:~$ chmod o-x run.sh

# 注意看少了个 x
ct@ehbio:~$ ls -l run.sh
-rwxr-xr-- 1 ct ct 39 6 月  14 23:12 run.sh

# 去除同组的可执行权限
ct@ehbio:~$ chmod g-x run.sh

# 注意看又少了个 x
ct@ehbio:~$ ls -l run.sh
-rwxr--r-- 1 ct ct 39 6 月  14 23:12 run.sh

# 去除所有人的可执行权限
ct@ehbio:~$ chmod a-x run.sh
ct@ehbio:~$ ls -l run.sh
-rw-r--r-- 1 ct ct 39 6 月  14 23:12 run.sh

# 给所有人增加可执行权限
ct@ehbio:~$ chmod a+x run.sh
ct@ehbio:~$ ls -l run.sh
-rwxr-xr-x 1 ct ct 39 6 月  14 23:12 run.sh
```

2.2 环境变量

如果一个文件有了可执行权限，是不是就可以执行了，我们来检测下。

```
ct@ehbio:~$ run.sh
-bash: run.sh: 未找到命令
```

事实上并非如此，输入命令，回车后，提示命令未找打，这是为什么呢？

这就涉及到环境变量的概念，通俗的讲，环境变量就是告诉电脑 (实际是操作系统) 几个目录。这几个目录下存储又可执行文件，如前面显示的`/usr/bin`目录，大部分的系统命令都在这个目录下。

当我们输入命令 `mkdir` 时，系统就会在环境变量所代表的几个目录从前都厚去查找，哪个里面有 `mkdir` 文件，然后去执行 `mkdir` 命令。

系统中环境变量的名字是 `PATH`，其内容可通过下面的命令显示 (根据操作系统不同和配置不同，略有差别，但格式是统一的，：分割的一堆路径)：

```
ct@ehbio:~$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
```

所以如果我们想让自己的命令能被系统找到，就需要把命令所在的目录加到环境变量里面，怎么操作呢？

加到环境变量的路径必须是全路径，全路径指以/开头或已 ~ 开头的路径 (~ 开头的路径只能个人用户有效)。

```
# 加到环境变量的路径必须是全路径，全路径指以/开头或已 ~ 开头的路径
# 注意第一个 PATH 不含 $，第二个 PATH 有 $ 符号
# 我们后面会讲什么时候用 $，什么时候不用 $
ct@ehbio:~$ export PATH=$PATH:/home/ct
ct@ehbio:~$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/ct
```

这时，我们在执行那个命令试试，成功运行了。

```
ct@ehbio:~$ run.sh
I am a script created by ehbio.
```

所以在以后安装了新的软件，或者写了新的脚本后，都把软件的可执行程序和可执行的脚本所在的目录，加到环境变量里面就可以了。

但是，在命令行中执行 `export`，对环境变量所做的修改，只对当前终端有效，退出后就无效了。为了使得这一操作，长期有效，我们需要把这句话写入一个文件中，一个登陆服务器就会被自动读取的文件中。

对于普通用户，在远程登录终端时，家目录下的 `~/.bash_profile` (不是 `~/.bashrc`, 在本地登录时才会被读取) 会自动被读取，所以我们需要把 `export` 语句加入到这个文件中。

```
# 这是我的 ~/.bash_profile 中的内容，主要是最好一行。可以连续的加入多个路径。
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi
```

```
export PATH=$PATH:/home/ct:/home/bin:/home/soft/bowtie2/bin
```

前面提到，系统查找命令的顺序是从环境变量的第一个目录到最后一个目录，在第一次碰到查询的命令后，就调用执行。假如系统存在一个 python 命令，我们自己又安装了一个 python (假如在/home/ct/anaconda/bin 目录下)，如果我们想执行自己的 python 程序，就需要把/home/ct/anaconda/bin 写在 $PATH 前面，如下

```
# 注意 $PATH 的顺序

ct@ehbio:~$ export PATH=/home/ct/anaconda/bin:$PATH
```

环境变量设置的两种方式：1. 不同给 $PATH 后面新增目录；2. 建议一个目录，放在 $PATH 中，之后新安装的软件采用软链方式。

至此，我们可以熟练使用环境变量来简化命令的输入过程了，因为如果没有环境变量，我们就得需要运行/home/ct/anaconda/bin/python 来运行 python 命令了。

环境变量这块，自己多操作下，就会慢慢理解熟练了。

2.2.1 环境变量的补充

PATH 只是众多环境变量中的一个变量，用于存储可执行文件所在的目录，以便在用户输入命令时可以查询的到。尤其是自己写的脚本或安装的程序，系统不会知道它们在哪个路径下，需要我们去提供给系统这些新的路径，学名叫设置环境变量。

此外常用到的环境变量还有 LD_LIBARY_PATH: 指定动态链接库 (so 文件) 的位置，一般在安装软件出错时会用到；PYTHONPATH: 指定 Python 的安装包的路径；PERL5LIB: 指定 perl 的安装包的路径。

设置环境变量要注意 2 点：1. 设置新的环境变量时一般要包含原始的环境变量，不能覆盖；2. 注意自己的目录和系统环境变量的目录的顺序，想让哪个先被找到，就先放哪个。

2.3 软件安装的几种方式

不同于 windows，Linux 下软件安装的方式比较多样，有些也比较复杂。每种安装方式都有自己的优点和局限，也都有可能遇到问题。在我们理解了原理之后，借助谷歌，可以更好地帮助解决问题。

2.3.1 系统包管理器安装

软件安装最方便的、一般也不容易出问题的是利用系统自带的包管理工具，可以解决大部分的依赖问题。

```
# centos
# 如果长时间没更新，先运行下update
yum update
# 如果不知道软件具体名字，可以先用一个关键字search一下，选择正式的名字
# 需要注意的是一般的服务器都是64 bit，需要选x86_64版本
yum search soft_name or soft_description
yum search soft_official_name
```

但也有一些不足，主要 3 点：

1. 需要根用户的权限。
2. 如果系统版本老，安装的软件版本也会比较老。使用新版本有时又会发生冲突。
3. 生物信息学中不少软件不在系统的安装源里面。

2.3.2 下载二进制文件

解决这些问题，就需要自己去软件官网查找最新的分法包，又有两种可能，一种是分法包直接就是编译好的软件，下载下来设置下可执行属性并放入环境变量就可以运行了，如于 `blast` 或 `bowtie` 这样的工具。

blast 的链接为ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ncbi-blast-2.7.1+-x64-linux.tar.gz。

```
wget ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ncbi-blast-2.7.1+-x64-linux.tar
tar xvzf ncbi-blast-2.7.1+-x64-linux.tar.gz
cd ncbi*
cd bin
# 直接进入 bin 目录，找到对应可执行文件，链接到在环境变量的目录中去。
# 具体可看视频的操作  http://bioinfo.ke.qq.com
ln -s `pwd`/* ~/bin
```

另一种则是需要从源码编译安装，下面主要讲解下这个。

2.3.3 源码编译安装

源码编译经典的三部曲 `configure`, `make`, `make install`。如果不出问题，一步步执行下来就安装好了，也不一定要知其所以然。但出了问题，就不是比较容易解决的。如果知道这背后的机制，还是会有帮助的。

- configure 是检查系统的库文件、类文件、依赖软件是否存在以及它们的版本是否满足需求，并根据实际检测结果生成 Makefile 的工具。一般是一堆 bash 命令的组合。通常也需要在这一步配置一些参数。最常用的就是指定软件的安装目录--prefix=/home/ct/soft/specific_name。
- make 则是具体的编译过程。编译的语句都写在了 Makefile 中。make 默认编译 Makefile 中出现的第一个 target，也可以指定 target 编译，并根据 Makefile 的设置方式依次编译所有依赖的东西。

有些软件的安装，在执行完 make 后就获得了可执行程序，可以跳过 make install 的过程，只需要把可执行程序放入环境变量就可以运行了。但部分软件还需要一些依赖关系，所以需要执行 make install 才算完成了完整的安装。

- make install 通常是拷贝 make 编译出来的可执行文件或者依赖的库文件 (如果有的话) 到 configure 时的--prefix 指定的目录下。
- 安装好的软件放入环境变量, 就可以快乐的运行了。

两条注意:

- 从源码编译最难解决的问题就是依赖的库文件、头文件、其它软件的缺失或版本不匹配，没有统一的解决办法，原则就是缺啥补啥。
- 三部曲每一步的执行，屏幕上都会输出比较多的信息，一定仔细看最后有没有 ERROR 类的字样，对判断软件有无安装成功和下一步要怎么做会很有帮助。

举一个例子，编译安装 samtools。具体看视频解释http://bioinfo.ke.qq.com。

```
wget https://jaist.dl.sourceforge.net/project/samtools/samtools/1.7/samtools-1.7.tar.bz2
tar xvzf samtools-1.7.tar.bz2
cd samtoo*
./configure --prefix=/home/ct/soft/samtools
make
make install
cd /home/ct/soft/samtools/bin
ln -s `pwd`/* ~/bin
```

小练习：尝试源码安装 EMBOSS, 下载地址 ftp://emboss.open-bio.org/pub/EMBOSS/emboss-latest.tar.gz.

EMBOSS 是欧洲分子生物学开放软件包，主要做序列比对，数据库搜搜，蛋白 motif 分析和功能域分析，序列模式搜索，引物设计等。

Table 2.1: Popular applications of EMBOSS.

Popular.applications	Functions
prophet	Gapped alignment for profiles.
infoseq	Displays some simple information about sequences.
water	Smith-Waterman local alignment.
pepstats	Protein statistics.
showfeat	Show features of a sequence.
palindrome	Looks for inverted repeats in a nucleotide sequence.
eprimer3	Picks PCR primers and hybridization oligos.
profit	Scan a sequence or database with a matrix or profile.
extractseq	Extract regions from a sequence.
marscan	Finds MAR/SAR sites in nucleic sequences.
tfscan	Scans DNA sequences for transcription factors.
patmatmotifs	Compares a protein sequence to the PROSITE motif database.
showdb	Displays information on the currently available databases.
wossname	Finds programs by keywords in their one-line documentation.
abiview	Reads ABI file and display the trace.
tranalign	Align nucleic coding regions given the aligned proteins.

2.3.4 Python 包的安装

在没有 `Anaconda`(或其前身 canopy) 出现之前，Python 包以其管理混乱、安装困难著称。有了 `Anaconda` 后，不只 python 包的安装简单了，其它软件的安装也都方便了 (详见后面 Anaconda 的两个福利)。

- 首先下载 Anaconda 的安装包 https://www.continuum.io/downloads。
- Anaconda 的安装包做的很人性化，一个 bash 脚本，只要运行 `bash Anacond*x86_64.sh`，然后按照提示操作就可以了。
- 安装好后，设置或刷新下环境变量就可以使用了。
- 此后再安装 python 的包只需要执行 `pip install pakcage_name` 或 `conda install pakckage_name` 就可以了。
- 这里唯一需要注意的就是确认使用的 `python` 或 `pip` 确实是 Anaconda 安装的 `python` 或 `pip`。
  - `which python` 查看使用的 python 命令。
  - 如果使用的还是系统默认的 python，则需要检查下环境变量的设置，尤其前面提到的环境变量里面不同目录放置的顺序。

2.3.5 Anaconda 的两个福利

1. 头文件和库文件库

这是 Anaconda 安装后的目录结构

```
bin   envs  Examples  imports  lib    LICENSE.txt  pkgs     share  var
conda-meta  etc   gcc include  lib64  mkspecsplugins  ssl
```

其中 lib 目录下，一部分是依赖的动态链接库，.so 文件；这也是在源码编译时最常见的拦路虎。通常，只需要把这个目录放入环境变量 LD_LIBRARY_PATH 里面比如 export LD_LIBARY_PATH=${LD_LIBARY_PATH}:anaconda_path/lib 就可以解决问题。

```
cairo                   libitm.a             libQtScript.so.4
cmake                   libitm.la            libQtScript.so.4.8
engines                 libitm.so            libQtScript.so.4.8.7
gcc                     libitm.so.1          libQtScriptTools.la
gcj-4.8.5-14            libitm.so.1.0.0      libQtScriptTools.prl
glib-2.0                libitm.spec          libQtScriptTools.so
libargtable2.a          libjpeg.a            libQtScriptTools.so.4
libargtable2.la         libjpeg.la           libQtScriptTools.so.4.8
libargtable2.so         libjpeg.so           libQtScriptTools.so.4.8.7
libargtable2.so.0       libjpeg.so.8         libQtSql.la
libargtable2.so.0.1.8   libjpeg.so.8.4.0     libQtSql.prl
libasan.a               libmkl_avx2.so       libQtSql.so
libasan.la              libmkl_avx512_mic.so libQtSql.so.4
libasan_preinit.o       libmkl_avx512.so     libQtSql.so.4.8
libasan.so              libmkl_avx.so        libQtSql.so.4.8.7
```

2. bioconda

bioconda 提供了一个虚拟环境，方便软件的编译安装，随后会具体介绍。

2.3.6 R 和 R 包的安装

如果使用的是新版的操作系统。直接可以用 sudo apt-get install r-base 或者 yum install r-base 来安装。

若系统版本老，或没有根用户权限，则需要下载编译源码安装，最新地址为https://cran.r-project.org/src/base/R-latest.tar.gz。

具体编译方式为 (Linux 下软件安装见 http://blog.genesino.com/2016/06/bash1):

```
# configure是收集系统信息，生成Makefile的过程
# --enable-R-shlib 需要设置，使得其他程序包括Rstudio可以使用R的动态库
# --prefix指定软件安装目录，需使用绝对路径
```

```
./configure --prefix=/home/ehbio/R/3.4.0 --enable-R-shlib

# 也可以使用这个命令，共享系统的blas库，提高运输速度
#./configure --prefix=/home/ehbio/R/3.4.0 --enable-R-shlib --with-blas --with-lapack

# make是编译的过程
make

# 安装到指定目录的过程
make install
```

安装完成之后，在 Linux 终端输入 R 即可启动交互式运行界面，ctrl+d 退出 R 运行界面。若提示找不到命令，需要判断有没有加入进环境变量。

如何安装 R 包

```
install.packages("package_name")
# 指定安装来源
install.packages("package_name", repo="http://cran.us.r-project.org")

# 安装Bioconductor的包
source('https://bioconductor.org/biocLite.R')
biocLite('BiocInstaller')
biocLite(c("RUVSeq","pcaMethods"))

# 安装Github的R包
install.packages("devtools")
devtools::install_github("JustinaZ/pcaReduce")

# 手动安装，首先下载包的源文件（压缩版就可），然后在终端运行下面的命令。
ct@ehbio:~$ R CMD INSTALL package.tar.gz

# 移除包
remove.packages("package_name")

# 查看所有安装的包
library()

# 查看特定安装包的版本
installed.packages()[c("ggplot2"), c("Package", "Version")]

#  Package  Version
# "DESeq2" "1.14.1"

# 查看默认安装包的位置
```

```
.libPaths()

# 查看已加载的包
.packages()

# 调用安装的包
library(package_name)
```

自动安装包

```
usePackage <- function(p) {
    if (!is.element(p, installed.packages()[,1])) {
        install.packages(p, dep = TRUE)
    }
    require(p, character.only = TRUE)
}
```

需要注意的也是依赖的软件或库文件的版本，同样的 Anaconda 提供的 lib 库也可以直接拿来用。

2.4 Conda 安装

Conda 是一种通用包管理系统，旨在构建和管理任何语言的任何类型的软件。通常与 Anaconda (集成了更多软件包，https://www.anaconda.com/download/#download) 和 Miniconda(只包含基本功能软件包, https://conda.io/miniconda.html) 一起分发。

最初接触到 Anaconda 是用于 Python 包的安装。Anaconda 囊括了 100 多个常用的 Python 包，一键式安装，解决 Python 包安装的痛苦。但后来发现，其还有更多的功能，尤其是其增加了 bionconda (https://bioconda.github.io/index.html) 频道后，生物信息分析的 1500 多个软件都可以一键安装了，免去了编译时间浪费和解决库文件安装的问题。对于经常编译软件的人，这一点还不够有吸引力。最有吸引力的是它的工作环境概念，可以简单的配置不同 Python 版本的环境、不同 Python 包的环境、不同 R 环境和 R 包的环境，对于生物信息软件繁杂的应用和频繁的更新提供了很大的便利。

2.4.1 Conda 安装和配置

在上面给出的链接下载 Anaconda 或 Conda 对应版本的分发包之后，安装就是运行下面的命令，根据提示一步步操作，主要是修改安装路径 (如果是根用户，可以安装到/anaconda 下，其它任意目录都可以，但路径短还是有好处的；普通用户安装到自己有权限的目录下)

```
bash Miniconda2-latest-Linux-x86_64.sh
```

安装完成之后，记得把安装路径下的 bin 文件夹加入到环境变量中。

2.4.2 Conda 基本使用

在 Conda 安装配置好之后，就可以使用了。

```
conda list # 列出安装的软件包
conda search <package ambigious name> # 搜索需要安装的软件包，获取其完成名字
```

以搜索 numpy 为例：

```
conda search numpy  # * 表示对于版本的包已安装
```

```
Fetching package metadata .............
numpy                     1.7.2        py27_blas_openblas_201  conda-forge     [blas_openl
                          1.7.2        py27_blas_openblas_202  conda-forge     [blas_openl
                          1.12.0             py36_0  defaults
                          1.12.0       py36_nomkl_0  defaults        [nomkl]
                       *  1.12.1             py27_0  defaults
                          1.12.1       py27_nomkl_0  defaults        [nomkl]
                          1.13.1             py36_0  defaults
                          1.13.1       py36_nomkl_0  defaults        [nomkl]
numpy-indexed             0.3.2              py27_0  conda-forge
                          1.0.47             py35_0  conda-forge
                          1.0.47             py36_0  conda-forge
numpy_groupies            0.9.6              py27_0  conda-forge
                          0.9.6              py35_0  conda-forge
                          0.9.6              py36_0  conda-forge
numpy_sugar               1.0.6              py27_0  conda-forge
                          1.0.6              py34_0  conda-forge
numpydoc                  0.6.0              py27_0  conda-forge
                          0.6.0              py34_0  conda-forge
xnumpy                    0.0.1              py27_0  conda-forge
```

安装包

```
conda install <package name> # 安装软件包
conda install numpy=1.7.2 # 安装特定版本的软件包
conda remove <package name> # 移除软件包
```

安装 R

```
# 具体见下面
# 安装R,及80多个常用的数据分析包，包括idplyr, shiny, ggplot2, tidyr, caret 和 nnet
conda install -c r r-essentials
# 安装单个包
# conda install -c https://conda.binstar.org/bokeh ggplot
```

获取帮助信息

```
conda -h # 查看conda可用的命令
conda install -h #查看install子命令的帮助
```

只是这些命令就可以省去不少安装的麻烦了，但是如果软件没搜索到呢？

2.4.3 Conda 的 channel

Conda 默认的源访问速度有些慢，可以增加国内的源；另外还可以增加几个源，以便于安装更多的软件，尤其是 bioconda 安装生信类工具。conda-forge 通道是 Conda 社区维护的包含很多不在默认通道里面的通用型软件。r 通道是向后兼容性通道，尤其是使用 R3.3.1 版本时会用到。后加的通道优先级更高，因此一般用下面列出的顺序添加。清华镜像具体见https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/。

```
conda config --add channels conda-forge # Lowest priority
conda config --add channels \
         https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
conda config --add channels r # Optional
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/msys2/
# Anocanda清华镜像
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
# 清华通道，最高优先级
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/bioconda/
conda config --set show_channel_urls yes
```

```
# 显示已有的通道
conda config --get channels
```

conda 通道的配置文件一般在 `~/.condarc` 里面，内容如下。全局控制 conda 的安装在 `conda_path/.condarc`，具体操作见https://conda.io/docs/user-guide/configuration/admin-multi-user-install.html。

```
channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/bioconda/
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/ # Anocanda清华镜像
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/msys2/
  - bioconda
  - defaults
  - r
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
  - conda-forge
```

2.4.4 创建不同的软件运行环境

这是 Conda 最有特色的地方，可以通过创建不同的环境，同时运行不同软件的多个版本。

新创建的软件环境的目录为 `anaconda_path/envs/enrironment_name`，具体见下面的 3 个例子。

- 安装 ete3

使用官方的推荐命令安装时出了问题，py3.5 的包装到了 py2.7 环境下。解决办法，新建一个 `py2.7` 的环境，然后安装。

```
# 新建一个环境，命名为phylo，指定其内安装的python版本为2.7
conda create -n phylo python=2.7

# 在phylo环境中安装 ete3
# ete3存在于2个通道中，官方推荐使用自己的通道，但没有成功
# -n 指定安装环境  -c 指定下载通道
# conda install -n phylo -c etetoolkit ete3 ete3_external_apps

# bioconda通道里面也有ete3，下面的安装未指定具体通道，
# 将在前面设定的几个通道里面按先后顺序查找安装
conda install -n phylo ete3 ete3_external_apps

# 默认安装到了anaconda_path下面的envs/phylo目录下（在屏幕输出也会有显示）
# 这个目录下存在bin文件夹，一般使用全路径就可以调用，如下
# anaconda_path/envs/phylo/bin/ete3 -h # 但有时会因为依赖关系而失败

# 所以激活本次安装环境是比较不容易出问题的使用方式
source activate phylo
```

```
# 在新环境里面执行命令操作
ete3 -h
# 其它操作

# 退出新环境
source deactivate phylo
```

- 创建 R 环境 Reference1

```
# Create a new conda environment called r,并且在里面安装anaconda
conda create -n r anaconda

# Switch to r environment
source activate r

# 在新环境里面安装R Installs R
conda install -c r r

# Install R kernel for IPython notebook
conda install -c r r-irkernel

# Install ggplot
conda install -c https://conda.binstar.org/bokeh ggplot

# 最后退出新环境
source deactivate r
```

- 创建比对工具环境 (bioconda 中的例子，https://bioconda.github.io/index.html#set-up-channels)

```
# 环境名字为 aligners
# 环境中安装 bwa bowtie hisat star
conda create -n aligners bwa bowtie hisat star

# 如果还想继续安装
conda install -n aligners hisat2

# 启动新环境
source activate aligners
star -h

source deactivate aligners
```

- 移除环境

如果环境不需要了，或出了错，则可以移除。比如需要移除 `phylo` 环境，执行 `conda remove -n phylo --all`。

2.4.5 Conda 配置 R

在添加了不同的源之后，有些源更新快，有些更新慢，经常会碰到版本不一的问题。而且软件版本的优先级，低于源的优先级。保险期间，先做下搜索，获得合适的版本号，然后再选择安装。

```
conda search r-essentials

r-essentials                1.0                  r3.2.1_0  r
                            1.0                 r3.2.1_0a  r
                            1.1                  r3.2.1_0  r
                            1.1                  r3.2.2_0  r
                            1.1                 r3.2.1_0a  r
                            1.1                 r3.2.2_0a  r
                            1.1                  r3.2.2_1  r
                            1.1                 r3.2.2_1a  r
                            1.4                          0  r
                            1.4.1                r3.3.1_0  r
                            1.4.2                        0  r
                            1.4.2                r3.3.1_0  r
                            1.4.3                r3.3.1_0  r
                            1.5.0                        0  r
                            1.5.1                        0  r
                            1.5.2                r3.3.2_0  r
                            1.5.2                r3.4.1_0  r
                            1.6.0                r3.4.1_0  r
                            1.0                  r3.2.1_0  defaults
                            1.0                 r3.2.1_0a  defaults
                            1.1                  r3.2.1_0  defaults
                            1.1                  r3.2.2_0  defaults
                            1.1                 r3.2.1_0a  defaults
                            1.1                 r3.2.2_0a  defaults
                            1.1                  r3.2.2_1  defaults
                            1.1                 r3.2.2_1a  defaults
                            1.4                          0  defaults
                            1.4.1                r3.3.1_0  defaults
                            1.4.2                        0  defaults
                            1.4.2                r3.3.1_0  defaults
                            1.4.3                r3.3.1_0  defaults
                            1.5.0                        0  defaults
```

```
                              1.5.1                      0  defaults
                              1.5.2                r3.3.2_0  defaults
                              1.5.2                r3.4.1_0  defaults
                              1.6.0                r3.4.1_0  defaults
                              1.5.2                r3.3.2_0  conda-forge
                              1.5.2                r3.3.2_0  https://mirrors.tuna.tsinghua.edu
anaconda/cloud/conda-forge
```

从上面可以看到清华的源版本同步于 conda-forge, 都比较老，还是指定 `r` 通道安装。

```
conda install -c r -n r r-essentials=1.6.0
```

`R` 会安装于 `conda_path/envs/r/bin` 中，软链到位于环境变量的目录中即可正常使用。

2.4.6 Conda 环境简化运行

为了方便不同环境里面程序的运行，我写了一个 shell 脚本 (`conda_env_run.sh`)，具体运行如下：

```
# -c: 表示实际需要运行的命令
# -e: 表示需要启动的软件环境，也就是上面conda create建立的环境
# -b：一般不需要指定，如果conda没在环境变量中需要给出conda的安装路径
conda_env_run.sh -c  'ete3 -h mod' -e phylo

conda_env_run.sh -c  'bwa mem -h' -e aligner -b "/usr/local/anaconda2/bin"
```

`conda_env_run.sh` 内容如下

```
#!/bin/bash

#set -x

usage()
{
cat <<EOF
${txtcyn}

***CREATED BY Chen Tong (chentong_biology@163.com)***

Usage:
```

```
$0 options${txtrst}

${bldblu}Function${txtrst}:

This is designed to run conda program in given environment.
It will automatically activate the environment, run the program and
deactivate the environment.

Thress commands from conda, 'activate', 'conda', 'deactivate' must
be in PATH or you should spcify <-b> parameter.

${txtbld}OPTIONS${txtrst}:
    -c  Full command to be run ${bldred}[NECESSARY]${txtrst}
    -e  Environment name${bldred}[NECESSARY]${txtrst}
    -b  Conda path${bldred}[NECESSARY]${txtrst}
EOF
}

command_cmd=''
environment=''
conda_path=''

while getopts "hc:e:b:" OPTION
do
    case $OPTION in
        h)
            echo "Help mesage"
            usage
            exit 1
            ;;
        c)
            command_cmd=$OPTARG
            ;;
        e)
            environment=$OPTARG
            ;;
        b)
            conda_path=$OPTARG
            ;;
        ?)
            usage
            echo "Unknown parameters"
            exit 1
            ;;
    esac
done
```

```
if [ -z ${environment} ]; then
    echo 1>&2 "Please give command and environment."
    usage
    exit 1
fi

if ! [ -z ${conda_path} ]; then
    export PATH=${conda_path}:${PATH}
fi

source activate ${environment}
${command_cmd}
source deactivate ${environment}
```

2.5 Makefile 知识

Makefile 通常的格式和布局如下，有兴趣的可以自己去学，或者我们再出一个教程。

```
# 假设当前文件夹下Makefile文件中内容如下
ct@ehbio:~$ cat Makefile
# first: target名字
# echo "compile first": target对应的命令，任何Linux命令都可以
first:
echo "compile first"
all: first second
echo "compile all"
second:
echo "compile second"

# 直接运行make，会make第一个出现的target
ct@ehbio:~$ make
echo "compile first"
compile first
# make first与直接make相同，因为它出现在第一个
ct@ehbio:~$ make first
echo "compile first"
compile first
# all依赖于first, second，因此make all会先执行make first, make second
# 然后才是自己所代表的命令
ct@ehbio:~$ make all
echo "compile first"
compile first
echo "compile second"
```

```
compile second
echo "compile all"
compile all
```

2.6 Docker 安装

2.6.1 Docker 能做什么

The key benefit of Docker is that it allows users to package an application with all of its dependencies into a standardized unit for software development.

- 提供一个虚拟的操作平台，供我们安装依赖不同版本系统的工具软件。
- 提供一个即时可用的应用软件或者流程的镜像，开发者把软件部署到系统镜像中，使用者可以直接下载下来使用，省去了个人安装软件的烦恼。
- 提供一个系统资源分配手段，给不同用户的程序分配独立的计算资源。

2.6.2 Docker 的几个基本概念

- 镜像 (Images): 可以认为是超级轻量级的虚拟机的快照。镜像会有自己的唯一 ID，名字和标签，比如 `ubuntu:latest`, `django:1.6` 等。通常都是在已有的镜像（多数是 Linux 操作系统的镜像）的基础上构建自己的具有新功能的镜像。
- 容器 (Containers): 可以认为是超级轻量级的虚拟机，是镜像运行起来所处的可读写的状态。容器里面可以安装、运行程序，还可以把安装好的程序存储起来获得新的镜像。

与虚拟机很大的不同在于，一个容器通常只运行一个程序。在 Docker 中，应用程序和数据文件是分开的，因此可以在不影响数据的情况下快速升级代码或系统。

- 数据卷 (Volumes): 永久保存数据的磁盘空间。Docker 允许用户定义哪一部分是应用程序，哪一步分是数据，并且把他们分隔开。这就保证了在 Docker 中容器的生命周期是短暂的，而数据的存储是永恒的。

数据卷存储在运行 Docker 的宿主机上，对每个容器来说是特有的。我们可以启动同一个镜像来产生多个容器，并且分别给他们分配一个数据卷。

数据卷也可用于在不同的容器间共享数据。具体参见http://blog.genesino.com//2016/09/docker-lamp/

• 联通 (Links): 容器启动后会分配有一个私有 IP，其它容器可以通过这个 IP 地址与这个容器通讯。

假如有个正在运行的数据库容器 (dbapp)，那么我们可以在网络服务器容器 (webserver) 中通过指定端口连接 dbapp 与数据库容器通讯。

2.6.3 安装和配置

• Centos 6.5 安装 Docker

“‘bash

添加 epel 的源 su -c ‘rpm -Uvh http://download.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm’ yum update

安装 Docker yum install docker-io

启动 Docker 服务 service docker start # 关闭 Docker 服务 service docker stop

设置 Docker 开机启动 /sbin/chkconfig –add docker /sbin/chkconfig docker on “‘

• 其他新版操作系统的安装可以直接使用系统自带的 `yum` 或 `apt` 工具，启动和配置如上。

```
bash   apt-get install docker.io
```

2.6.4 Docker 用户权限

默认情况下，Docker 命令的运行需要根用户权限。一个解决办法是把用户加入 `docker` 用户组，原因是 Docker 能够将 `/run/docker.socket` 的文件权限设为 `660`、用户组设为 `docker`。当把用户加入到 `docker` 用户组后，就无需使用 `sudo` 或 `su` 命令切换获取根用户权限。check here

```
### 以下操作都是在根用户下进行的

### 增加一个用户组 docker
# groupadd docker
### 把用户 ${USER} 加入 docker 用户组
# usermod -aG docker ${USER}
### 重启 docker 服务 (可不执行)
```

```
# service docker restart
### 新窗口登录 ${USER}
```

但通常只应把信任的用户加入 docker 用户组因为 docker 用户组的权限相当于 root。

如果打算只允许用户访问一个特定的容器，可以写一个简单脚本

```
# cat /bin/docker_container1
#!/bin/sh
docker run -ti --rm container_name /bin/sh
```

脚本完成后，配置 sudoers

```
# grep username /etc/sudoers
username    ALL=(ALL)  NOPASSWD: /bin/docker_container1
```

更多权限设置见http://dockone.io/article/589

2.6.5 Docker 试用

- 查看本地 Docker 的信息 `docker info`
- 运行 Docker 需要有一个镜像和容器。镜像是容器的只读版本，最基础的镜像是一个操作系统，是运行其他命令的基础。因此我们需要先获取一个操作系统镜像，通常使用 Ubuntu 系统, CentOS 系统和 Alpine (只有 5M)。我们也可以根据所要运行软件的需要，来获取不同的操作系统，方便软件的安装。
- 搜索镜像 `docker search ubuntu`; 镜像的名字通常由用户名/镜像名构成, 无用户名的为官方认证镜像。

```
root@server:~# docker search ubuntu
NAME                DESCRIPTION       STARS     OFFICIAL   AUTOMATED
ubuntu              Ubuntu is ...     4595      [OK]
ubuntu-upstart      Upstart is...      66       [OK]
jordi/ubuntu        Ubuntu bas...      1                    [OK]
```

- 获取镜像
- `docker pull ubuntu` 获取镜像的最新版本 (不指定版本号即为 latest)
- `docker pull ubuntu:14.04` 获取指定版本的镜像；14.04 为镜像的版本号 (又称 `TAG`)。
- 查看本机 Docker 中存在的镜像 `docker images`

```
REPOSITORY    TAG       IMAGE ID        CREATED       VIRTUAL SIZE
ubuntu        latest    37b164bb431e    4 days ago    126.6 MB
```

- 获得了镜像之后，我们需要运行镜像；运行起来的镜像就是容器，是可读写的。我们可以在容器中安装软件、运行命令，就如在正常的操作系统中一样。

在容器中运行单个命令或程序, 通常加`--rm`参数，容器运行结束之后就自动删除。如果我们想保留容器的修改，则不能加`--rm`参数。

```
docker run --rm ubuntu echo "Hello from docker"
Hello from docker
```

交互式运行容器`docker run -it ubuntu`, 这时可以发现终端的用户名变了

```
root@server:~# docker run -it ubuntu
root@57cb695e904f:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@57cb695e904f:/#
```

`docker run --help`可以查看这个命令的参数。

- 在容器中部署软件，安装 `build-essential`和`r-base`; *build-essential* 是编译软件包的基础，提供需要的编译器、头文件和库文件。*r-base* 是编译 R 语言程序包的基础。

```
apt-get update
#apt-get install -y build-essential r-base
apt-get install toliet
```

这一步我们可以安装任意的软件，测试时可以选择小一点的软件包。最开始时选择了安装 `build-essential`, 只是为了学习，到后来发现安装这个并没有什么用，也不方便测试。为了简单起见，可以尝试安装 `Apache`。在本文后面有个简单的测试 `Apache` 安装的例子。

- 运 行 `docker commit - m 'Add build- essential r- base' - a ct5869 8aca49b869be ct5869/ubuntu-dev:v1''`。
- 测试运行新的镜像`docker run --rm -it username/ubuntu-dev:v1`。
- 挂载宿主机硬盘在容器内部操作, 通过`-v`参数，路径都为绝对路径，`docker run --rm -v /host_absolute_dir:/container_absolute_dir username/ ubuntu- dev:v1 echo 'test' >/ container_absolute_dir/test_file`这样，就相当于把 host 机目录`/host_absolute_dir`链接为 docker 容器路径`/container_absolute_dir`。
- 如果只是自己用，到现在就可以结束了，我们可以在镜像里面继续更多的操作了。
- 另外我们还可以运用导出和导入来迁移镜像
  - 导出镜像：`docker export image_id >ubuntu-dev.v1.tar`

  - 导入镜像：`cat ubuntu-dev.v1.tar | docker import - username/ubuntu-dev:v1`

- 如果我们想把镜像分发给别人使用，就需要把镜像传到镜像仓库比如 Docker Hub。我们需要现在Docker hub注册，用注册的用户名替换掉前文提到的 `username`。
- 注册成功之后，在本地服务器尝试登录，用以把登录信息存储在本地，方便后续使用。运行 `docker login`，按提示输入用户名、密码和邮件。登录成功会返回 `Login Succeeded`.
- 运行 `docker push username/ubuntu-dev:v1` 把准备好的镜像上传；等待片刻，完成上传。这时就可以再 Docker hub 上看到上传的镜像了。
- 其它用户可以使用 `docker pull username/ubuntu-dev:v1` 来获取安装好编译环境的 ubuntu 系统了。

2.6.6 Docker 系统基本操作

- 当一个容器不再使用时，运行 `docker rm container_id` 移除容器，以节省空间。这不会对镜像造成影响。
- 当一个容器不再使用时，运行 `docker rm -v container_id` 移除容器及其挂载卷，以节省空间。这不会对镜像造成影响。
- 批量删除退出的容器 `docker rm -v $(docker ps -a -q -f status=exited)`。
- 对于只需要单次运行的容器，比如执行一个命令等，则只需要在 `docker run` 时添加`--rm` 参数就好。这样容器运行结束后会自动删除。
- 运行 `docker rmi username/ubuntu-dev:v1` 移除镜像。
- 运行 `docker tag 26d99f722dca username/ubuntu-dev:v0` 修改镜像的名字。
- 运行 `docker run -d --name=container_name username/ubuntu-dev:v1` 指定运行的 container 的名字。
- 运 行 `docker run -- rm - ti - v / host_abs_dir:/ container_abs_dir:ro username/ ubuntu-dev:v1` 挂载只读目录。
- 运行 `docker stop containde_id/container_name` 停止镜像。
- 运行 `docker rm $(docker ps -a -q)` 和 `docker rmi $(docker images -q)` 移除全部镜像。**BE CARE-FULL**
- 查看 Docker 镜像的创建历史 `docker history image_name`

```
IMAGE          CREATED       CREATED BY SIZE                 COMMENT
3d4f934accdb   7 months ago        /bin/sh -c #(nop) CMD ["/run.sh" ]              0 B
aa321fa8d23f   7 months ago        /bin/sh -c #(nop) EXPOSE  3306/tcp 80/tcp       0 B
6446fbfc507d   7 months ago        /bin/sh -c #(nop) VOLUME  [/etc/mysql /var/lib   0 B
44e98bdf2bbf   7 months ago        /bin/sh -c #(nop) ENV PHP_POST_MAX_SIZE=10M     0 B
bedff16caee9   7 months ago        /bin/sh -c #(nop) ENV  PHP_UPLOAD_MAX_FILESIZE   0 B
72b723ccc97f   7 months ago        /bin/sh -c mkdir -p /app && rm -fr /var/www/h   0 B
```

- 查看镜像的 JSON 文件 `docker inspect image_name`

- Docker images 的安装路径为 `/var/lib/docker`。
- `/var/lib/docker/{driver-name}` will contain the driver specific storage for contents of the images.
- `/var/lib/docker/graph/<id>` now only contains metadata about the image, in the json and layersize files.
- 查看 Docker 容器启动和运行日志

```
docker logs --tail=all container_id
```

2.6.7 使用 **Dockerfile** 自动构建镜像

除了可以像上面那样一步步地获取镜像、修改容器、存储镜像、上传镜像等操作外，我们还可以使用 Dockerfile 自动实现上述操作。

典型的 Dockerfile 如下所示，

```
FROM alpine
MAINTAINER username username@internet.com
RUN apk add --no-cache apache2 apache2-utils
COPY public_html /var/www/html
EXPOSE 80 443
CMD ["rc-service apache2 start"]
```

- `FROM` 为除注释之外的第一条命令，用来声明镜像的基础系统。
- `MAINTAINER` 设置镜像维护人的信息。
- `RUN` 在容器内部运行 shell 命令。
- `COPY` 是把本地的 bash 配置文件拷贝到新维护的镜像中；COPY 的文件的路径是相对于 docker build 的 PATH，一般是当前路径；
- `CMD` 指定容易运行时默认执行的命令，如出现多个，只有最后一个会被运行。

运行命令 `docker build -t="username/httpd-alpine:v1" .` 就可以构建镜像了。最后的. 表示 Dockerfile 在当前目录，也可指定其他目录。`public_html` 必须与 Dockerfile 在同一目录。

2.6.8 **Docker** 的特征

- Docker will watch only one single process. If you need multiple processes, you need to add a monitor like Monit or Supervisor at the top-level to take care of the others. But this is not recommended.

2.6.9 Docker 使用注意

- 避免安装不必要的软件包。
- 每个容器都只运行一个进程。
- 最小化层：每执行一个命令，都会产生一个层。

2.6.10 参考

- 入门级 http://blog.saymagic.cn/2015/06/01/learning-docker.html
- 入门级 https://www.dwhd.org/20151115_140935.html
- 入门级 http://www.cnblogs.com/kevinX/p/5458244.html
- Start (english version) https://scotch.io/tutorials/getting-started-with-docker
- Start (english version) https://prakhar.me/docker-curriculum/
- Greate english version https://blog.talpor.com/2015/01/docker-beginners-tutorial/
- Docker trick https://blog.docker.com/2014/06/why-you-dont-need-to-run-sshd-in-docker/
- Docker root and non-root http://www.2cto.com/os/201508/432930.html

2.7 References

- https://samrelton.wordpress.com/2015/07/02/rconda/
- https://www.anaconda.com/blog/developer-blog/anaconda-r-users-sparkr-and-rbokeh/
- http://www.bioinfo-scrounger.com/archives/209
- 清华大学开源镜像站
- Linux 学习 - 又双叒叕一个软件安装方法

3 Linux 神器

视频课见 http://bioinfo.ke.qq.com。

3.1 正则表达式

正则表达式 (regular expression) 是用来做模糊匹配的，匹配符合特定模式的文本。最早来源于 Unix 系统中的 `sed` 和 `grep` 命令，在各个程序语言，如 `perl`, `python` 中也都有实现。不同程序语言中正则表达式语法大体通用，细节上又各自有自己的特色。

语法	说明	表达式实例	完整匹配的字符串
字符			
一般字符	匹配自身	abc	abc
.	匹配任意除换行符"\n"外的字符。 在DOTALL模式中也能匹配换行符。	a.c	abc
\	转义字符，使后一个字符改变原来的意思。 如果字符串中有字符*需要匹配，可以使用*或者字符集[*]。	a\.c a\\c	a.c a\c
[...]	字符集（字符类）。对应的位置可以是字符集中任意字符。 字符集中的字符可以逐个列出，也可以给出范围，如[abc]或[a-c]。第一个字符如果是^则表示取反，如[^abc]表示不是abc的其他字符。 所有的特殊字符在字符集中都失去其原有的特殊含义。在字符集中如果要使用]、-或^，可以在前面加上反斜杠，或把]、-放在第一个字符，把^放在非第一个字符。	a[bcd]e	abe ace ade
预定义字符集（可以写在字符集[...]中）			
\d	数字：[0-9]	a\dc	a1c
\D	非数字：[^\d]	a\Dc	abc
\s	空白字符：[<空格>\t\r\n\f\v]	a\sc	a c
\S	非空白字符：[^\s]	a\Sc	abc
\w	单词字符：[A-Za-z0-9_]	a\wc	abc
\W	非单词字符：[^\w]	a\Wc	a c
数量词（用在字符或(...)之后）			
*	匹配前一个字符0或无限次。	abc*	ab abccc
+	匹配前一个字符1次或无限次。	abc+	abc abccc
?	匹配前一个字符0次或1次。	abc?	ab abc
{m}	匹配前一个字符m次。	ab{2}c	abbc
{m,n}	匹配前一个字符m至n次。 m和n可以省略：若省略m，则匹配0至n次；若省略n，则匹配m至无限次。	ab{1,2}c	abc abbc
*? +? ?? {m,n}?	使 * + ? {m,n}变成非贪婪模式。	示例将在下文中介绍。	
边界匹配（不消耗待匹配字符串中的字符）			
^	匹配字符串开头。 在多行模式中匹配每一行的开头。	^abc	abc
$	匹配字符串末尾。 在多行模式中匹配每一行的末尾。	abc$	abc
\A	仅匹配字符串开头。	\Aabc	abc
\Z	仅匹配字符串末尾。	abc\Z	abc
\b	匹配\w和\W之间。	a\b!bc	a!bc
\B	[^\b]	a\Bbc	abc
逻辑、分组			
|	|代表左右表达式任意匹配一个。 它总是先尝试匹配左边的表达式，一旦成功匹配则跳过匹配右边的表达式。 如果|没有被包括在()中，则它的范围是整个正则表达式。	abc|def	abc def
(...)	被括起来的表达式将作为分组，从表达式左边开始每遇到一个分组的左括号'('，编号+1。 另外，分组表达式作为一个整体，可以后接数量词。表达式中的|仅在该组中有效。	(abc){2} a(123|456)c	abcabc a456c
(?P<name>...)	分组，除了原有的编号外再指定一个额外的别名。	(?P<id>abc){2}	abcabc
\<number>	引用编号为<number>的分组匹配到的字符串。	(\d)abc\1	1abc1 5abc5
(?P=name)	引用别名为<name>的分组匹配到的字符串。	(?P<id>\d)abc(?P=id)	1abc1 5abc5
特殊构造（不作为分组）			
(?:...)	(...)的不分组版本，用于使用'|'或后接数量词。	(?:abc){2}	abcabc
(?iLmsux)	iLmsux的每个字符代表一个匹配模式，只能用在正则表达式的开头，可选多个。匹配模式将在下文中介绍。	(?i)abc	AbC
(?#...)	#后的内容将作为注释被忽略。	abc(?#comment)123	abc123
(?=...)	之后的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	a(?=\d)	后面是数字的a
(?!...)	之后的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	a(?!\d)	后面不是数字的a
(?<=...)	之前的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	(?<=\d)a	前面是数字的a
(?<!...)	之前的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	(?<!\d)a	前面不是数字的a
(?(id/name) yes-pattern |no-pattern)	如果编号为id/别名为name的组匹配到字符，则需要匹配yes-pattern，否则需要匹配no-pattern。 |no-patern可以省略。	(\d)abc(?(1)\d|abc)	1abc2 abcabc

Figure 3.1: 正则表达式基本语法

```
# 假如有这么一个测试文件
ct@ehbio: ~/$ cat <<END >url.list
http://www.ehbio.com/ImageGP
http://www.ehbio.com/Training
http://www.ehbio.com/Esx
www.ehbio.com
http://www.ehbio.com/ImageGP/index.php/Home/Index/Lineplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/GOenrichmentplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/PHeatmap.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Boxplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Barplot.html
```

```
http://www.ehbio.com/ImageGP/index.php/Home/Index/Volcanoplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Manhattanplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Histogram.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/VennDiagram.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/UpsetView.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Densityplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/PCAplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/PCoAplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/CPCoAplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Sankey.html
http://blog.genesino.com
https://blog.csdn.net/qazplm12_3/
https://blog.csdn.net/woodcorpse/
blog.csdn.net/woodcorpse/article/details/79313846

ImageGP is one of the Best online plot.
123456789
END
```

获取以 `https` 开头的行

```
ct@ehbio: ~/$ grep '^https' url.list
https://blog.csdn.net/qazplm12_3/
https://blog.csdn.net/woodcorpse/
```

获取包含数字的行

```
ct@ehbio:~/$ grep '[0-9]' url.list
https://blog.csdn.net/qazplm12_3/
blog.csdn.net/woodcorpse/article/details/79313846
123456789
```

获取空行

```
ct@ehbio:~/$ grep '^$' url.list
```

获取 `html` 结尾的行

```
ct@ehbio:~/$ grep 'html$' url.list
```

```
http://www.ehbio.com/ImageGP/index.php/Home/Index/Lineplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/GOenrichmentplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/PHeatmap.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Boxplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Barplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Volcanoplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Manhattanplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Histogram.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/VennDiagram.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/UpsetView.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Densityplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/PCAplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/PCoAplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/CPCoAplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Sankey.html
```

获取 Boxplot 和 Barplot 的地址

```
# 未能满足要求
ct@ehbio:~/$ grep 'B.*plot' url.list
http://www.ehbio.com/ImageGP/index.php/Home/Index/Boxplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Barplot.html
ImageGP is one of the Best online plot.

# 一个办法：更长的匹配
ct@ehbio:~/$ grep 'B.*plot.html' url.list
http://www.ehbio.com/ImageGP/index.php/Home/Index/Boxplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Barplot.html

# 限定中间不能有空格
ct@ehbio:~/$ grep 'B[^ ]*plot' url.list
http://www.ehbio.com/ImageGP/index.php/Home/Index/Boxplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Barplot.html

# 限定中间只能有2个字符
ct@ehbio:~/$ grep 'B..plot' url.list
http://www.ehbio.com/ImageGP/index.php/Home/Index/Boxplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Barplot.html

# 2个字符的另外一种写法
ct@ehbio:~/$ grep -P 'B.{2}plot' url.list
http://www.ehbio.com/ImageGP/index.php/Home/Index/Boxplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Barplot.html

# 2个字符的再一种写法，只允许出现特定字符
```

```
ct@ehbio:~/$ grep -P 'B[arox]*plot' url.list
http://www.ehbio.com/ImageGP/index.php/Home/Index/Boxplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/Barplot.html
```

获取 PCA 或 PCoA 相关的行

```
ct@ehbio:~/$ grep 'PCo*A' url.list
http://www.ehbio.com/ImageGP/index.php/Home/Index/PCAplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/PCoAplot.html
http://www.ehbio.com/ImageGP/index.php/Home/Index/CPCoAplot.html
```

3.2 awk 命令

前面的学习过程中已经提到了 `awk` 和 `sed` 的使用，作为一个引子。现在则详细列举关于 `awk` 常用的操作和一些偏门的操作。

3.2.1 awk 基本参数解释

awk 擅长于对文件按行操作，每次读取一行，然后进行相应的操作。

awk 读取单个文件时的基本语法格式是 `awk 'BEGIN{OFS=FS="\t"}{print $0, $1;}' filename`。

读取多个文件时的语法是 `awk 'BEGIN{OFS=FS="\t"}ARGIND==1{print $0, $1;}ARGIND==2{print $0;}' file1 file2`。

awk 后面的命令部分是用引号括起来的，可以单引号，可以双引号，但注意不能与内部命令中用到的引号相同，否则会导致最相邻的引号视为一组，引发解释错误。引号不可以嵌套

- `OFS`: 文件输出时的列分隔符 (output field separtor)
- `FS`: 文件输入时的列分隔符 (field separtor)
- `BEGIN`: 设置初始参数，初始化变量
- `END`: 读完文件后做最终的处理
- 其它 `{}`：循环读取文件的每一行
- `$0` 表示一行内容；`$1`, `$2`, ... `$NF` 表示第一列，第二列到最后一列。
- `NF (number of fields)` 文件多少列；`NR (number of rows)` 文件读了多少行: `FNR` 当前文件读了多少行，常用于多文件操作时。

- `a[$1]=1`: 索引操作，类似于 python 中的字典，在 `ID map`，统计中有很多应用。

3.2.2 常见操作

- 针对特定列的计算，比如 wig 文件的标准化

```
# 注意除了第一行是空格，其它行都是 tab 键分割
ct@ehbio:~/sxbd$ cat <<END >ehbio.wig
variableStep chrom=chr2
300701  12
300702  10
300703  11
300704  13
300705  12.5
END

ct@localhost:~/sxbd$ awk 'BEGIN{OFS=FS="\t"}\
    {if(FNR>1) $2=$2*10^6/(2.5*10^6); print $0}' ehbio.wig
variableStep chrom=chr2
300701  4.8
300702  4
300703  4.4
300704  5.2
300705  5
```

- 计算某列内容出现的次数。

```
# 怎么获得 count 文件，应该不难吧
ct@ehbio:~/sxbd$ cat count
ID  Type
Pou5f1  Pluripotency
Nanog   Pluripotency
Sox2    Neuron
Tet1    Epigenetic
Tet3    Epigenetic
Myc Oncogene

ct@ehbio:~/sxbd$ awk 'BEGIN{OFS=FS="\t"}{if(FNR>1) a[$2]+=1;}END\
    {print "Type\tCount"; for(i in a) print i,a[i];}' count
Type    Count
Neuron  1
Epigenetic  2
Oncogene    1
Pluripotency    2
```

```
# 这个也可以用下面方式代替，但不直接
ct@ehbio:~/sxbd$ tail -n +2 count | cut -f 2 | sort | uniq -c | \
        sed -e 's/^  *//' -e 's/  */\t/'
2   Epigenetic
1   Neuron
1   Oncogene
2   Pluripotency
```

- 之前也提到过的列操作，从 GTF 文件中提取启动子区域

GRCh38.gtf 可以从ftp://ftp.ensembl.org/pub/release-91/gtf/homo_sapiens/Homo_sapiens.GRCh38.91.gtf.gz下载，或使用提供的测试文件。

```
ct@ehbio:~/sxbd$ sed 's/"/\t/g' GRCh38.gtf | \
    awk 'BEGIN{OFS=FS="\t"}{if($3=="gene") {ensn=$10; symbol=$16; \
        if($7=="+") {start=$4-1; up=start-1000; if(up<0) up=0; dw=start+500; \
        print $1,up, dw, ensn, symbol, $7;} else \
        if($7=="-") {start=$5-1; up=start+1000; dw=start-500; \
        if(dw<0) dw=0; print $1,dw,up,ensn,symbol,$7}}}' | sort -k1,1 -k2,2n \
        >GRCh38.promoter.bed
```

- 数据矩阵的格式化输出

```
ct@ehbio:~/sxbd$ cat numeric.matrix
ID  A   B   C
a   1.002   1.234   1.999
b   2.333   4.232   0.889

ct@ehbio:~/sxbd$ awk '{if(FNR==1) print $0; \
    else {printf "%s%s",$1,FS; for (i=2; i<=NF; i++) \
        printf "%.1f %s", $i, (i==NF?RS:FS)}}' numeric.matrix
ID  A   B   C
a 1.0  1.2  2.0
b 2.3  4.2  0.9
```

- 判断 FASTQ 文件中，输出质量值的长度是与序列长度不一致的序列 ID

```
ct@ehbio:~/sxbd$ cat <<END | gzip -c >Test_2.fq.gz
>ehbio1
ACGTCGACGACGAGAGGAGAGGAGCCCTCTCGCCCGCCCTACTACCACCCACACACAACACAAGTGT
+
FFFFFFA$A#$$AFEEEEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

```
>ehbio2
ACGTCGACGACGAGAGGAGAGGAGCCCTCTCGCCCGCCCTACTACCACCCACACACAACACAAGTGT
+
FFFFFF$A#$$AFEEEEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
>ehbio3
ACGTCGACGACGAGAGGAGAGGAGCCTCTCGCCCGCCCTACTACCACCCACACACAACACAAGTGT
+
FFFFFFA$A#$$AFEEEEFFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
END
```

```
ct@ehbio:~/sxbd$ zcat Test_2.fq.gz | \
    awk '{if(FNR%4==1) ID=$0; else if(FNR%4==2) seq_len=length($0); \
        else if(FNR%4==0) {quality_len=length($0); if(seq_len!=quality_len) print ID; }}'
```

- 筛选差异基因

```
# TAB 键分割的文件
ct@ehbio:~/sxbd$ cat de_gene
ID  log2fc  padj
A   1   0.001
B   -1  0.001
C   1   0.001
D   2   0.0001
E   -0.51   0.051
F   0.1 0.1
G   1   0.1

ct@ehbio:~/sxbd$ awk '$3<0.05 || NR==1' de_gene
ID  log2fc  padj
A   1   0.001
B   -1  0.001
C   1   0.001
D   2   0.0001

ct@ehbio:~/sxbd$ awk 'BEGIN{OFS=FS="\t"}{if(FNR==1) print $0; \
    else {abs_log2fc=($2<0?$2*(-1):$2);if(abs_log2fc>=1 && $3<0.05) print $0;}}' de_gene
ID  log2fc  padj
A   1   0.001
B   -1  0.001
C   1   0.001
D   2   0.0001
```

- 筛选差异基因存储到不同的文件

```
ct@ehbio:~/sxbd$ awk 'BEGIN{OFS=FS="\t"; up="up"; dw="dw";}\
    {if(FNR==1) {print $0 >up; print $0 >dw;} else \
    if ($3<0.05) {if ($2>=1) print $0 >up; else if($2<=-1) print $0 >dw;}}' de_gene

ct@ehbio:~/sxbd$ head up dw
==> up <==
ID  log2fc  padj
A   1   0.001
C   1   0.001
D   2   0.0001

==> dw <==
ID  log2fc  padj
B   -1  0.001
```

- 筛选差异基因存储到不同的文件 (自动版)

```
awk 'BEGIN{OFS=FS="\t"; up=ARGV[1]".up"; dw=ARGV[1]".dw";}\
    {if(FNR==1) {print $0 >up; print $0 >dw;} \
    else if ($3<=0.05) {if($2<=-1) print $0 >up; else if ($2>=1) print $0 >dw;}}' de_gene
```

- ID map，常用于转换序列的 ID、提取信息、合并信息等

```
# TAB 键分割的文件
ct@ehbio:~/sxbd$ cat id_map
ENSM    Symbol  Entrez
ENSG00000280516 TMEM42  693149
ENSG00000281886 TGM4    7047
ENSG00000280873 DGKD    8527
ENSG00000281244 ADAMTS13    11093
ENSG00000280701 RP11-272D20.2
ENSG00000280674 ZDHHC3  51304
ENSG00000281623 Y_RNA
ENSG00000280479 CACFD1  11094
ENSG00000281165 SLC2A6  11182
ENSG00000281879 ABO 28
ENSG00000282873 BCL7A   605
ENSG00000280651 AC156455.1  100506691
ct@ehbio:~/sxbd$ vim ensm
ct@ehbio:~/sxbd$ cat ensm
ENSG00000281244
ENSG00000281165
ENSG00000282873
```

```
ct@ehbio:~/sxbd$ awk 'BEGIN{OFS=FS="\t"}ARGIND==1{if(FNR>1) ensm2entrez[$1]=$3;}\
        ARGIND==2{print ensm2entrez[$1];}' id_map ensm
11093
11182
605

# 替代解决方案，注意 -w 的使用，避免部分匹配。最稳妥的方式还是使用 awk。

ct@ehbio:~/sxbd$ grep -w -f ensm id_map | cut -f 3
11093
11182
605
```

- 转换大小写, toupper, tolower

```
ct@ehbio:~/sxbd$ cat symbol
Tgm4
Dgkd
Abo

ct@ehbio:~/sxbd$ awk 'BEGIN{OFS=FS="\t"}ARGIND==1{if(FNR>1) ensm2entrez[$2]=$3;}\
        ARGIND==2{print ensm2entrez[toupper($1)];}' id_map symbol
7047
8527
28
```

- awk 数值操作

```
ct@ehbio:~/sxbd$ cat <<END >file
2
4
3.1
4.5
5.4
7.6
8
END

# log2 对数
ct@ehbio:~/sxbd$ awk 'BEGIN{OFS="\t";FS="\t"}{print log($0)/log(2)}' file

# 取整，四舍五入
ct@ehbio:~/sxbd$ awk 'BEGIN{OFS="\t";FS="\t"}{print int($1+0.5);}' file
```

- awk 定义函数

```
ct@ehbio:~/sxbd$ cat <<END | sed 's/  */\t/g'>file
1   2   3   4
5   6   7   8
9   10  11  12
END
```

```
ct@ehbio:~/sxbd$ awk 'function abs(x){return ((x < 0.0) ? -x : x)}BEGIN{OFS="\t";FS="\t"}\
    {pos[1]=$1;pos[2]=$2;pos[3]=$3;pos[4]=$4; len=asort(pos); \
    for(i=len;i>1;i--) print abs(pos[i]-pos[i-1]);}' file
```

- 字符串匹配

```
# TAB 键分割的文件
ct@ehbio:~/sxbd$ cat ens.bed
1   100 105
2   100 105
3   100 105
Mt  100 105
X   100 105

ct@ehbio:~/sxbd$ awk 'BEGIN{OFS=FS="\t"}{if($1~/^[0-9XY]/) $1="chr"$1; else \
    if($1~/M.*/) gsub(/M.*/, "chrM", $1); print $0}' ens.bed
chr1    100 105
chr2    100 105
chr3    100 105
chrM    100 105
chrX    100 105
```

- 字符串分割

```
ct@ehbio:~/sxbd$ cat trinity_id
Trinity_C1_g1_i1
Trinity_C1_g1_i2
Trinity_C1_g1_i3
Trinity_C2_g1_i1
Trinity_C3_g1_i1
Trinity_C3_g3_i2
ct@ehbio:~/sxbd$ awk 'BEGIN{OFS=FS="\t"}{count=split($1, geneL, "_"); gene=geneL[1]; \
    for(i=2;i<count;i++) gene=gene"_"geneL[i]; print gene,$1;}' trinity_id
Trinity_C1_g1   Trinity_C1_g1_i1
Trinity_C1_g1   Trinity_C1_g1_i2
Trinity_C1_g1   Trinity_C1_g1_i3
Trinity_C2_g1   Trinity_C2_g1_i1
Trinity_C3_g1   Trinity_C3_g1_i1
Trinity_C3_g3   Trinity_C3_g3_i2
```

- awk 脚本

```
ct@ehbio:~/sxbd$ cat <<END >grade.awk
BEGIN{OFS=FS="\t"; up=ARGV[1]".up"; dw=ARGV[1]".dw";}
    {if(FNR==1) {print $0 >up; print $0 >dw;}
                 else if ($3<=0.05) {
                         if($2<=-1) print $0 >up;
                         else if ($2>=1) print $0 >dw;}
    }
END

ct@ehbio:~/sxbd$ awk -f grade.awk de_gene
```

- awk 给每行增加行号，使其变为唯一

```
ct@ehbio:~/sxbd$ awk 'BEGIN{OFS="\t";FS="\t"}NR!=1{$4=$4"_"NR;print $0}' file
```

3.2.3 糅合操作

- awk 中执行系统命令 (注意引号的使用)

```
# input_mat
ct@ehbio:~/sxbd$ cat <<END | sed 's/  */\t/g' >input_mat
SRR1    root
SRR2    leaf
SRR3    stem
END

ct@ehbio:~/sxbd$ touch SRR1.fq SRR2.fq SRR3.fq

ct@ehbio:~/sxbd$ ls
SRR1.fq SRR2.fq SRR3.fq

# 系统命令组成字符串，交给 system函数运行
ct@ehbio:~/sxbd$ awk 'BEGIN{OFS=FS="\t"}{system("mv "$1".fq "$2".fq");}' input_mat

#
ct@ehbio:~/sxbd$ ls
leaf.fq root.fq stem.fq
```

- awk 引用系统变量

```
ct@ehbio:~/sxbd$ echo 1 | awk -v ehbio="shengxinbaodian" \
                        -v ehbio2="sxbd" '{print ehbio, ehbio2;}'
shengxinbaodian sxbd
```

3.3 SED 命令

3.3.1 sed 基本参数解释

sed 是 stream editor 的简称，擅长对文件进行各种正则操作、插入操作、替换操作和删除操作，可以全局，可以指定特定范围的行或者特定特征的行。

s/pat/replace/: 正则替换

前插行 i, 后插行 a, 替换行 c, 删除行 d, 输出行 p

N: 读入下一行，同时存储；n: 读入下一行，抛弃当前行

3.3.2 常见操作

- 替换特定的文本

```
# 空格是我们不太喜欢出现在文件中的一个符号，尤其是作为列名字时
# 列使用 TAB 键分割
ct@ehbio:~/sxbd$ cat <<END | sed 's/;/\t/g' mat
ID;2 cell;4 cell;8 cell;embryo
Pou5f1_1;2;3;4;5
Nanog_1;2;3.2;4.3;5
c-Myc;2;3;4;5
Tet1_3;2;3;4;5
END

# 一次替换
ct@ehbio:~/sxbd$ sed 's/ /_/' mat
ID  2_cell  4 cell  8 cell  embryo
Pou5f1_1    2   3   4   5
Nanog_1 2   3.2 4.3 5
c-Myc   2   3   4   5
Tet1_3  2   3   4   5

# 全部替换
```

```
ct@ehbio:~/sxbd$ sed 's/ /_/g' mat
ID  2_cell  4_cell  8_cell  embryo
Pou5f1_1    2   3   4   5
Nanog_1 2   3.2 4.3 5
c-Myc   2   3   4   5
Tet1_3  2   3   4   5
```

- 获得逗号分隔的一组数

```
ct@ehbio:~/sxbd$ echo `seq 1 10` | sed 's/ /,/g'
1,2,3,4,5,6,7,8,9,10
```

- 针对指定行替换

```
ct@ehbio:~/sxbd$ sed '2,$ s/_[0-9]//g' mat
ID  2 cell  4 cell  8 cell  embryo
Pou5f1  2   3   4   5
Nanog   2   3.2 4.3 5
c-Myc   2   3   4   5
Tet1    2   3   4   5
```

- 替换特定出现位置

```
# 替换第一个空格
ct@ehbio:~/sxbd$ sed 's/ /_/1' mat
ID  2_cell  4 cell  8 cell  embryo
Pou5f1_1    2   3   4   5
Nanog_1 2   3.2 4.3 5
c-Myc   2   3   4   5
Tet1_3  2   3   4   5
# 替换第二个空格
ct@ehbio:~/sxbd$ sed 's/ /_/2' mat
ID  2 cell  4_cell  8 cell  embryo
Pou5f1_1    2   3   4   5
Nanog_1 2   3.2 4.3 5
c-Myc   2   3   4   5
Tet1_3  2   3   4   5
# 替换第二个及以后的空格
ct@ehbio:~/sxbd$ sed 's/ /_/2g' mat
ID  2 cell  4_cell  8_cell  embryo
Pou5f1_1    2   3   4   5
Nanog_1 2   3.2 4.3 5
c-Myc   2   3   4   5
Tet1_3  2   3   4   5
```

- 给序列起名字

```
ct@ehbio:~/sxbd$ cat seq
ACDGTFGGCATGCDTGD
ACDGAGCDTAGCDGTA
CAGDTAGDCTADTG
ct@ehbio:~/sxbd$ sed = seq
1
ACDGTFGGCATGCDTGD
2
ACDGAGCDTAGCDGTA
3
CAGDTAGDCTADTG
# 同时缓冲两行，但只对第一行行首操作
ct@ehbio:~/sxbd$ sed = seq | sed 'N;s/^/>/;'
>1
ACDGTFGGCATGCDTGD
>2
ACDGAGCDTAGCDGTA
>3
CAGDTAGDCTADTG
```

- 给文件增加标题行

```
ct@ehbio:~/sxbd$ tail -n +2 mat | sort -k2,2n
c-Myc    2    3    4    5
Nanog_1  2    3.2  4.3  5
Pou5f1_1     2    3    4    5
Tet1_3   2    3    4    5

# 1 表示第一行
# i 表示插入，在指定行前面插入新行
ct@ehbio:~/sxbd$ tail -n +2 mat | sort -k2,2n | sed '1 i ID\t2_cell\t4_cell\t8_cell\tembryo'
ID   2_cell   4_cell   8_cell   embryo
c-Myc    2    3    4    5
Nanog_1  2    3.2  4.3  5
Pou5f1_1     2    3    4    5
Tet1_3   2    3    4    5
```

- 提取特定或指定范围的行

```
# -n 是必须的，阻止程序自动输出匹配行，不然会导致重复输出
ct@ehbio:~/sxbd$ sed -n '2,4p' mat
Pou5f1_1     2    3    4    5
Nanog_1  2    3.2  4.3  5
```

```
c-Myc   2   3   4   5

ct@ehbio:~/sxbd$ sed -n '4p' mat
c-Myc   2   3   4   5
```

- 提取符合特定模式的行

/pattern/支持普通字符串和正则表达式匹配

```
ct@ehbio:~/sxbd$ sed -n '/_/ p' mat
Pou5f1_1    2   3   4   5
Nanog_1 2   3.2 4.3 5
Tet1_3  2   3   4   5

ct@ehbio:~/sxbd$ sed -n '/-/ p' mat
c-Myc   2   3   4   5
```


- 去除文件中的空行

```
ct@ehbio:~/sxbd$ cat mat
ID  2 cell  4 cell  8 cell  embryo
Pou5f1_1    2   3   4   5
Nanog_1 2   3.2 4.3 5

c-Myc   2   3   4   5
Tet1_3  2   3   4   5


# 空行就是只有行首和行尾的行
ct@ehbio:~/sxbd$ sed '/^$/d' mat
ID  2 cell  4 cell  8 cell  embryo
Pou5f1_1    2   3   4   5
Nanog_1 2   3.2 4.3 5
c-Myc   2   3   4   5
Tet1_3  2   3   4   5
```

- 原位删除

```
ct@ehbio:~/sxbd$ cat mat
ID  2 cell  4 cell  8 cell  embryo
Pou5f1_1    2   3   4   5
Nanog_1 2   3.2 4.3 5
```

```
c-Myc    2    3    4    5
Tet1_3   2    3    4    5

# -i 参数的使用
ct@ehbio:~/sxbd$ sed -i '/^$/d' mat
ct@ehbio:~/sxbd$ cat mat
ID   2 cell   4 cell   8 cell   embryo
Pou5f1_1     2    3    4    5
Nanog_1 2    3.2 4.3 5
c-Myc    2    3    4    5
Tet1_3   2    3    4    5
```

- 删除指定范围的行

```
ct@ehbio:~/sxbd$ cat mat
ID   2 cell   4 cell   8 cell   embryo
Pou5f1_1     2    3    4    5
Nanog_1 2    3.2 4.3 5
c-Myc_2 2    3    4    5
Tet1_3   2    3    4    5

ct@ehbio:~/sxbd$ sed '2,3d' mat
ID   2 cell   4 cell   8 cell   embryo
c-Myc_2 2    3    4    5
Tet1_3   2    3    4    5
```

- 记忆匹配

\(\) 启动记忆匹配；\1 为第一个匹配项，\2 为第二个匹配项；匹配项的计数根据左括号出现的位置来定，第一个 （包括起来的为\1。

```
ct@ehbio:~/sxbd$ echo "hah ehbio hah" | sed 's/ \(.*\) /\t\1\t\1\t/'
hah ehbio    ehbio    hah
```

- 奇偶数行处理

```
ct@ehbio:~/sxbd$ echo -e "odd\neven\nodd\neven"
odd
even
odd
even
```

```
# 奇偶数行合并
ct@ehbio:~/sxbd$ echo -e "odd\neven\nodd\neven" | sed 'N;s/\n/\t/'
odd even
odd even

# 取出偶数行，比较简单
# 注意 n （小写） 撇掉了奇数行
ct@ehbio:~/sxbd$ echo -e "odd\neven\nodd\neven" | sed -n 'n;p'
even
even

# 取出奇数行
# 先都读进去，然后替换偶数行为空值，再输出
ct@ehbio:~/sxbd$ echo -e "odd\neven\nodd\neven" | sed -n 'N;s/\n.*//;p'
odd
odd
```

- Windows/Linux 换行符困境

Windows 下的换行符是`\r\n`, Linux 下换行符是`\n`, MAC 下换行符是`\r`。所以 Windows 下的文件拷贝到 Linux 后，常会出现行尾多一个 `^M` 符号的情况，从而引起匹配或其它解析问题。

`^M` 的输是 `ctrl+v+M` `ctrl+v;ctrl+m`，不是简单的输入 `^`，再输入 `M`。

```
ct@ehbio:~/sxbd$ cat -A windows.txt
ID^M$
A^M$
B^M$
C^M$
ct@ehbio:~/sxbd$ sed 's/^M//' windows.txt | cat -A
ID$
A$
B$
C$
```

- sed 中使用 bash 变量

```
# 注意双引号的使用
ct@ehbio:~/sxbd$ bash_variable='ehbio'
ct@ehbio:~/sxbd$ echo "sheng xin bao dan " | sed "s/$/$bash_variable/"
sheng xin bao dan ehbio
```

3.4 VIM 的使用

VIM 是一款功能强大的文本编辑工具，也是我在 Linux，Windows 下编辑程序和文本最常用的工具。

3.4.1 初识 VIM

VIM 分多种状态模式，写入模式，正常模式，可视化模式。

- 正常模式：打开或新建文件默认在正常模式，可以浏览，但不可以写入内容。这个模式也可以称作命令行模式，这个模式下可以使用 VIM 强大的命令行和快捷键功能。其它模式下按 ESC 就可以到正常模式。
- 写入模式：在正常模式下按字母 i (光标前插入), o (当前光标的下一行操作), O (当前光标的上一行操作)，a (光标后插入) 都可以进入写入模式，就可以输入内容了。
- 可视化模式：通常用于选择特定的内容。

进入写入模式后，VIM 使用起来可以跟记事本一样了。在写入文字时，可以利用组合键 CTRL+n 和 CTRL+p 完成写作单词的自动匹配补全，从而加快输入速度，保证输入的前后一致。

正常模式有更强大的快捷键编辑功能，把手从鼠标上解放出来。

- dd: 删除一行
- 3dd: 删除一行
- dw: 删除一个单词
- d3w: 删除 3 个单词
- yy: 复制一行
- 3yy: 复制三行
- yw: 复制一个单词
- p: (小写 p) 粘贴到下一行
- P: (大写 P) 粘贴到上一行
- >>: 当前行右缩进一个 TAB
- 3>>: 当前行及后 2 行都向右缩进一个 TAB
- <<: 当前行左缩进一个 TAB
- 3<<: 当前行及后 2 行都向左缩进一个 TAB
- /word: 查找特定单词

- `u`: 撤销上一次操作
- `.`: 重复上一次操作
- `CTRL+r`: 重做撤销的操作
- `y$`: 从当前复制到行尾
- `d$`: 从当前删除到行尾

跳转操作

- `gg`: 跳到文件开头
- `G`: 跳到文件结尾
- `zt`: 当前行作为可视屏幕的第一行
- `5G`: 跳到第 5 行

正常模式下输入冒号进入更强大的命令行定制功能。

- `:5d`: 删除第 5 行
- `:20,24y`：复制 20 到 24 行
- `:.,+3y`：复制当前行和下面 3 行
- `:2,11>`: 右缩进
- `:w`: 保存文件
- `:q`: 退出编辑器
- `:vsplit`: 分屏

键盘操作不容易被捕获，看右下角可以得到一点信息。动图请点击查看。

VIM 还有不少魔性操作，具体可以看这两个帖子：

- http://coolshell.cn/articles/5426.html
- http://coolshell.cn/articles/11312.html

3.4.2 VIM 中使用正则表达式

这儿以提取生信宝典公众号中发过的原创文章的 HTML 代码为例子，获得原创文章的名字和链接，用以制作文章列表。

部分数据如下所示，利用正则表达式的第一步就是找规律。

- 这段文字是 JSON 格式，列表和字典的组合，使用 `json` 函数可以很容易解析。但我们这通过正则表达式解析。
- `title` 后面跟随的文章的题目; `url` 后面跟随的是文章的链接。
- `{"` 和 `"}` 标记每篇文章的信息的开始和结束。
- `auth_apply_num` 是目前不关注的信息。


```
1 ":[{"auth_apply_num":0,"author_type":1,"bizuin":3291706085,"create_time":"2017-08-24","delete_flag"
  :0,"index":1,"msgid":2247484235,"reprint_num":0,"title":"生信软件系列 - NCBI使用","update_time":0,"
  url":"http://mp.weixin.qq.com/s?__biz=MzI5MTcwNjA4NQ==&amp;mid=2247484235&amp;idx=1&amp;sn=1afa3b9b
  954e9f8bb99d954a5305681c&amp;chksm=ec0dc6c1db7a4fd7098a872b887f8e077aa2d1fcf101614df050524d945eceda
  54150ecebc72#rd"},{"auth_apply_num":0,"author_type":1,"bizuin":3291706085,"create_time":"2017-08-23
  ","delete_flag":0,"index":1,"msgid":2247484194,"reprint_num":0,"title":"新出炉的Cytoscape视频教程",
  "update_time":0,"url":"http://mp.weixin.qq.com/s?__biz=MzI5MTcwNjA4NQ==&amp;mid=2247484194&amp;idx=
  1&amp;sn=61bcbe1c48e195c5c830396865789723&amp;chksm=ec0dc6a8db7a4fbeaa9cdd7245127edd382f3e4d13a6163
  6c2cbc52062b32d7565bf282fca5e#rd"},{"auth_apply_num":0,"author_type":1,"bizuin":3291706085,"create_
  time":"2017-08-20","delete_flag":0,"index":3,"msgid":2247484184,"reprint_num":0,"title":"扩增子图标
  解读8网络图：节点OTU或类Venn比较","update_time":0,"url":"http://mp.weixin.qq.com/s?__biz=MzI5MTcwNj
  A4NQ==&amp;mid=2247484184&amp;idx=3&amp;sn=4dd39fa3188b902c827af7c12f601457&amp;chksm=ec0dc692db7a4
  f8486e04471b349844a52852c449bebd7e8503a2fca5689147949f74333b802#rd"},{"auth_apply_num":0,"author_ty
  pe":1,"bizuin":3291706085,"create_time":"2017-08-20","delete_flag":0,"index":1,"msgid":2247484184,"
  reprint_num":0,"title":"Cytoscape之操作界面介绍","update_time":0,"url":"http://mp.weixin.qq.com/s?_
  _biz=MzI5MTcwNjA4NQ==&amp;mid=2247484184&amp;idx=1&amp;sn=2384ba8a6a6ea3ed7dac1b42217c710a&amp;chks
  m=ec0dc692db7a4f84a03feb4de092aa0712b1bca2975f1cee8fac8342d0a65f836f5f94e9d877#rd"},{"auth_apply_nu
  m":0,"author_type":1,"bizuin":3291706085,"create_time":"2017-08-16","delete_flag":0,"index":1,"msgi
  d":2247484167,"reprint_num":0,"title":"本地安装UCSC基因组浏览器","update_time":0,"url":"http://mp.w
  eixin.qq.com/s?__biz=MzI5MTcwNjA4NQ==&amp;mid=2247484167&amp;idx=1&amp;sn=ef9699899164d23013d92b613
  31c6562&amp;chksm=ec0dc68ddb7a4f9bfb83b9942196622f8b8f52b041d3f4a12786e6464752fbeaf64c9a640e39#rd"}
  ,{"auth_apply_num":0,"author_type":1,"bizuin":3291706085,"create_time":"2017-08-16","delete_flag":0
  ,"index":1,"msgid":2247484166,"reprint_num":0,"title":"NGS基础 - GTF/GFF文件格式解读和转换","update
```


下面的动画展示了如何通过正则表达式，把这段文字只保留题目和链接，并转成 `Markdown` 的格式。

- `:set wrap`: 折行显示
- `:s/"}, {"/\r/g`: : 开启命令行模式；s: 是替换，之前讲 Linux 命令时也多次提及；/作为分割符，三个一起出现，前两个/中的内容为被替换内容，后两个/中的内容为替换成的内容；这里没有使用正则表达式，直接是原字符的替换，`\r` 表示换行符。这样把每篇文章的信息单行显示，方便后续处理。
- `:%s/auth_apply.*"title":"/[/` : % 表示对所有行进行操作；被替换的内容是 `auth_apply` 和 `title":"` 及其之间的内容 (`.*` 表示，. 表示任意字符，* 表示其前面的字符出现任意次)
- `:%s/".*"url":"/](/` : 从题目到 url 之间的内容替换掉；第一次替换时忘记了第一行中开头还有引号，结果出现了误操作，后面又退回去，手动删除特殊部分，其它部分继续匹配。
- `:%s/$/)/` : 表示在行尾 ($) 加上)，就组成了 Markdown 中完整的链接形式 `[context](link)`。
- `:%s/^/* /` : 表示在行首 (^) 加上 * 变成 Markdown 格式的列表

至此就完成了生信宝典公众号文章到 Markdown 链接的转换，可以放到菜单栏文章集锦里面方便快速查询了。

一步步的处理也有些麻烦，有没有办法更简单些呢？动画可查看链接。

- 首先也是把每篇文章的信息处理为单行显示，一样的模式更容易操作，去掉第一行行首不一致的部分
- 使用上下箭头可以回溯之前的命令，类似于 Linux 终端下的操作
- `%s/.*title":"\([^"]*\).*url":"\(.*\)/* [\1](\2)/c`: 这个是记忆匹配，记录下匹配的内容用于替换，`\(`和`\)` 表示记忆匹配的开始和结束，自身不匹配任何字符，只做标记使用；从左只右，第一个`\(`中的内容记录为`\1`,

第二个\(中的内容记录为\2, 以此类推。尤其在存在括号嵌套的情况下，注意匹配位置，左括号出现的顺序为准。在匹配文章题目时使用了 `[^"]*` 而不是 `.*`，是考虑到正则表达式的匹配是贪婪的，会囊括更多的内容进来，就有可能出现非预期情况，所以做这么个限定，匹配所有非" 内容。

正则表达式在数据分析中有很多灵活的应用，可以解决复杂的字符串抽提工作。常用的程序语言或命令如 `pytho`, `R`, `grep`, `awk`, `sed` 都支持正则表达式操作，语法也大体相似。进一步学习可参考一下链接：

- VIM 正则表达式 http://blog.csdn.net/u014015972/article/details/50688837
- Pyton 正则表达式 https://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html

3.5 References

- www.ehbio.com/Training
- Linux 学习 - 常用和不太常用的实用 awk 命令
- Linux-总目录
- Linux-文件和目录
- Linux-文件操作
- Linux 文件内容操作
- Linux-环境变量和可执行属性
- Linux - 管道、标准输入输出
- Linux - 命令运行监测和软件安装
- Linux-常见错误和快捷操作
- Linux-文件列太多，很难识别想要的信息在哪列；别焦急，看这里。
- Linux-文件排序和 FASTA 文件操作
- Linux-应用 Docker 安装软件
- Linux 服务器数据定期同步和备份方式
- VIM 的强大文本处理方法
- Linux - Conda 软件安装方法
- 查看服务器配置信息
- Linux - SED 操作，awk 的姊妹篇
- Linux - 常用和不太常用的实用 awk 命令
- Bash 概论 - Linux 系列教程补充篇
- CIRCOS 圈图绘制 - circos 安装
- CIRCOS 圈图绘制 - 最简单绘图和解释
- CIRCOS 圈图绘制 - 染色体信息展示和调整
- CIRCOS 增加热图、点图、线图和区块属性

4 Bash 字符串处理

视频课见 http://bioinfo.ke.qq.com。

4.1 Bash 特殊字符

1. 通配符:

*: 匹配任何字符

**: 匹配任何字符串

*?: 匹配任何单个字符

2. 集合运算符

用一些单个字、一个连续范围或断续的字符集合作为通配符

[a-z]: 用字符集合作通配符匹配单个字符, 如: [aeiou], [a-o], [A-Z], [0-9]

[!A-Za-z0-9]: 除了集合外的所有字符组成的集合作通配符

3. 花括号展开式（可以嵌套）:

c{a{r,t,n}, b{r,t,n}}s 可以匹配 cars cats cans cbrs cbts cbns

4. 其它特殊字符:

(): 子 shell 运行；比如 (cd ehbio; mdkir ysx) 进入 ehbio 目录，新建 ysx 文件夹，运行完之后还在当前目录。

': 强引用字符串, 不解释特殊字符

": 弱引用字符串, 解释所有特殊字符

;: 命令分隔符（命令终止符），运行在一行里执行多条命令; 一般在终端直接写判断语句或执行 `for` 循环时用。

#: 行注释
$: 变量表达式，变量解析 &: 在后台执行命令，在 `for` 循环中也可用作命令分割符，取代 `done` 前面的;

4.2 Bash 变量

1. 自定义变量

用户自定义的变量由字母、数字和下划线组成, 并且变量名的第一个字符不能为数字, 且变量名大小写敏感。

`varname=value` 注意 **bash** 不能在等号两侧留空格

shell 语言是非类型的解释型语言, 给一个变量赋值实际上就是定义了变量, 而且可以赋不同类型的值。引用变量有两种方式, `$varname` 和 `${varname}`, 为防止变量在字符串中产生歧义建议使用第二种方式, 引用未定义的变量其值为空。

```
ct@ehbio:~$ a="EHBIO"
ct@ehbio:~$ echo ${a}
EHBIO
ct@ehbio:~$ echo $a
EHBIO

#出错了
ct@ehbio:~$ echo $agood

#引用变量时大括号的作用
ct@ehbio:~$ echo ${a}good
EHBIOgood
ct@ehbio:~$ echo $a.good
EHBIO.good

#出错了
ct@ehbio:~$ echo $a_good

#引用变量时大括号的作用
ct@ehbio:~$ echo ${a}_good
EHBIO_good
```

为了使变量可以在其它进程中使用, 需要将变量导出: `export varname`

2. 环境变量

可以用 `set` 命令给变量赋值或查看环境变量值, 使用 `unset` 命令清除变量值, 使用 `export` 导出变量将可以使其它进程访问到该环境变量。可以把设置保存到`.bashrc` 或`.bash_profile` 中, 成为永久的环境变量。

环境变量不限于我们之前讲过的可执行程序的环境变量、动态库、Python 模块的环境变量，任何变量都可以的。

3. 位置变量

位置变量对应于命令行参数, 其中 `$0` 为脚本名称, `$1` 为第一个参数, 依次类推, 参数超过 9 个必须使用 `${}` 引用变量。shell 保留这些变量, 不允许用户以另外的方式定义它们, 传给脚本或函数的位置变量是局部和只读的, 而其余变量为全局的（可以用 local 关键字声明为局部）。

4. 其它变量

`$?`: 保存前一个命令的返回码; `0` 为运行成功，常用来判断上一个程序的退出状态。

`$$`: 当前 shell 的进程号

`$!`: 上一个子进程的进程号

`$#`: 传给脚本或函数的参数个数, 即位置变量数减 `1`(1 代表脚本自身)
`$*` 和 `$@`: 传给脚本的所有参数 (不包含脚本本身), 每个参数以 `$IFS` 分隔（一般内为空格, TAB, 换行）; 两者的不同点是引号括起来时，`$*` 会被作为一个整体，`$@` 还是单个的参数。


```
ct@ehbio:~$ cat ehbio_testParam.sh
#!/bin/bash

echo "EHBIO${IFS}great"

echo '$*'
echo -ne "\t";
echo $*

echo '$@'
echo -ne "\t";
echo $@

echo 'Each element in $*:'

for i in "$*"; do
    echo -ne "\t";
    echo $i;
```

```
done


echo 'Each element in $@:'
for i in "$@"; do
    echo -ne "\t";
    echo $i;
done

ct@ehbio:~$ bash ehbio_testParam.sh sheng xin bao dian
EHBIO
great
$*
    sheng xin bao dian
$@
    sheng xin bao dian
Each element in $*:
    sheng xin bao dian
Each element in $@:
    sheng
    xin
    bao
    dian
```

4.3 Bash 操作符

1. 字符串操作符（替换操作符）

`${var:-word}`: 如果 var 存在且不为空, 返回它的值, 否则返回 word
`${var:=word}`: 如果 var 存在且不为空, 返回它的值, 否则将 word 赋给 var, 返回它的值
`${var:+word}`: 如果 var 存在且不为空, 返回 word, 否则返回空
`${var:?message}` 如果 var 存在且不为空, 返回它的值,
否则显示 “-bash: var: message”, 然后退出当前命令或脚本
`${var:offset[:length]}` 从 offset 位置开始返回 var 的一个长为 length 的子串,
若没有 length, 则默认到 var 串末尾

```
ct@ehbio:~$ echo ${var:?message}
-bash: var: message
ct@ehbio:~$ var='sheng xin bao dian'
ct@ehbio:~$ echo ${var:6:3}
xin
ct@ehbio:~$ echo ${var:?message}
sheng xin bao dian
ct@ehbio:~$ echo $?
```

```
0
ct@ehbio:~$ unset var
ct@ehbio:~$ echo ${var:?message}
-bash: var: message
ct@ehbio:~$ echo $?
1
ct@ehbio:~$ echo ${var:=ehbio}
ehbio
ct@ehbio:~$ echo ${var}
ehbio
```

2. 模式匹配操作符

`${var#pattern}` 从 var 头部开始, 删除和 pattern 匹配的最短模式串, 然后返回剩余串

`${var##pattern}` 从 var 头部开始, 删除和 pattern 匹配的最长模式串, 然后返回剩余串, `basename path = ${path##*/}`

`${var%pattern}` 从 var 尾部开始, 删除和 pattern 匹配的最短模式串, 然后返回剩余串, `dirname path = ${path%/*}`

`${var%%pattern}` 从 var 尾部开始, 删除和 pattern 匹配的最长模式串, 然后返回剩余串

`${var/pattern/string}` 用 string 替换 var 中和 pattern 匹配的最长模式串

个人最常用的是最后一个，常用于 `for` 循环中。

```
ct@ehbio:~$ var='sheng xin bao dian good'
ct@ehbio:~$ ${var/good/great}
-bash: sheng: command not found
ct@ehbio:~$ echo ${var/good/great}
sheng xin bao dian great
```

比如获取 fastq 文件的名字部分

```
for i in `ls *_1.fq.gz`; do j=${i/_1.fq.gz/}; echo "$j"; done
```

4.4 Shell 中条件和 test 命令

Bash 可以使用 `[ … ]` 结构或 `test` 命令测试复杂条件
格式: `[ expression ]` 或 `test expression`
返回一个代码, 表明条件为真还是为假, 返回 `0` 为真, 否则为假。
注: 左括号后和右括号前空格是必须的语法要求

1. 文件测试操作符

`-d file`: file 存在并且是一个目录
`-e file`: file 存在
`-f file`: file 存在并且是一个普通文件
`-g file`: file 存在并且是 SGID(设置组 ID) 文件
`-r file`: 对 file 有读权限
`-s file`: file 存在并且不为空
`-u file`: file 存在并且是 SUID(设置用户 ID) 文件
`-w file`: 对 file 有写权限
`-x file`: 对 file 有执行权限, 如果是目录则有查找权限
`-O file`: 拥有 file
`-G file`: 测试是否是 file 所属组的一个成员
`-L file`: file 为符号链接
`file1 –nt file2`: file1 比 file2 新

`file1 –ot file2`: file1 比 file2 旧

举两个例子

```
ct@ehbio:~$ touch older
ct@ehbio:~$ touch newer

ct@ehbio:~$ if test -e older; then echo "older esists"; fi
older esists
ct@ehbio:~$ if test -s older; then echo "older is unempty"; fi
ct@ehbio:~$ if [ -s older ]; then echo "older is unempty"; fi
ct@ehbio:~$ if [ ! -s older ]; then echo "older is empty"; fi
older is empty
ct@ehbio:~$ if [ newer -nt older ]; then echo "newer"; fi
newer
```

2. 字符串操作符

`str1=str2` str1 和 str2 匹配

`str1!=str2` str1 和 str2 不匹配
`str1>str2` str1 大于 str2
`-n str` str 的长度大于 0（不为空）`-z str` str 的长度为 0（空串），常用于判断必须的命令行参数是否传入

```
# 字符串的大小比较的是最先遇到的不同的ASCII码的大小
ct@ehbio:~$ if test "10">"20"; then echo "10>20"; fi
10>20
ct@ehbio:~$ if test 10>20; then echo "10 < 20"; fi
```

3. 整数操作符

var1 –eq var2 var1 等于 var2
var1 –ne var2 var1 不等于 var2
var1 –ge var2 var1 大于等于 var2
var1 –gt var2 var1 大于 var2
var1 –le var2 var1 小于等于 var2
var1 –lt var2 var1 小于 var2

`ge`: great equal; `gt`: great than

需要注意的是常用的数学运算符给了字符串比较，数字比较使用的却是英文缩写

数学表达式也可以

```
if (( 3>2 )); then echo 'TRUE'; fi
TRUE
```

4. 逻辑操作符

`!expr` 对 expr 求反 `expr1 && expr2` 对 expr1 与 expr2 求逻辑与, 当 expr1 为假时不再执行 expr2
`expr1 || expr2` 对 expr1 与 expr2 求逻辑或, 当 expr1 为真时不再执行 expr2

4.5 Shell 流控制

1. 条件语句: if

`if`, `then`, `elif`, `else`, `fi` 是关键词，其它的是需要替换掉的。

```
if conditions; then
    do sth when conditions are true
elif another_conditions; then
    do sth when another_conditions are true
else:
    do sth when above condiitons are all false
fi


if test $guanzhu_sxbd == "already"; then
    echo "Enjoy it"
elif test $guanzhu_hjyz == "already"; then
    echo "Enjoy it"
else
    echo "Guan zhu them"
fi

Enjoy it
```

2. 确定性循环: `for do done` 常用的批量操作方式

遍历一个列表，取出每个元素，针对性操作。

```
for i in `ls *_1.fq.gz`; do
    echo "$i";
done
```

3. 不确定性循环: `while` 和 `until`

```
declare -i a #定义整数变量
a=1   # 初始化变量
while test $a -lt 3; do
    echo $a
    a=$a+1
done

echo $a
```

4. 选择结构: `case` 和 `select` (类似 getopts)

```
ct@ehbio:~$ cat select_case.sh
```

```
PS3="Input the position of selected parameter (1 for exit):"

select opts in a b c d
do
    case $opts in
        a)
            exit 0;
            ;;
        b)
            echo " Parameters $opts"
            ;;
        c)
            echo " Parameters $opts"
            ;;
        d)
            echo " Parameters $opts"
            ;;
        ?)
            echo "Unknown"
            ;;
    esac
done

ct@ehbio:~$ bash select_case.sh
1) a
2) b
3) c
4) d
Input the position of selected parameter (1 for exit):2
 Parameters b
Input the position of selected parameter (1 for exit):3
 Parameters c
Input the position of selected parameter (1 for exit):4
 Parameters d
Input the position of selected parameter (1 for exit):1
```

5. 命令 shift

将存放在位置变量中的命令行参数依次向左传递 shift n 命令行参数向左传递 n 个参数串

```
ct@ehbio:~$ cat ehbio_testParam.sh
#!/bin/bash
```

```
echo 'Each element in $*:'

for i in "$*"; do
    echo -ne "\t";
    echo $i;
done

echo $1
shift

for i in "$*"; do
    echo -ne "\t";
    echo $i;
done
ct@ehbio:~$ bash ehbio_testParam.sh sheng xin bao dian
Each element in $*:
    sheng xin bao dian
sheng
    xin bao dian
```

4.6 Shell 函数

`function function_name () { function body}` 定义函数，函数参数的获取同命令行参数获取。

```
ct@ehbio:~$ cat test_func.sh
function show_ehbio () {
    echo $@
    echo $1
}

show_ehbio "EHBIO great" "SXBD great"
ct@ehbio:~$ bash test_func.sh
EHBIO great SXBD great
EHBIO great
```

4.7 输入输出

1. I/O 重定向

管道、标准输入输出之前有过详细介绍。

<: 输入重定向

>: 输出重定向 (没有文件则创建, 有则覆盖)

>>: 输出重定向 (没有则创建, 有则追加到文件尾部)

<<: 输入重定向 (here 文档)

```
command << label
input…
label
```
说明： 使一个命令的输入为一段shell脚本(input…)，直到标号(label)结束

```
ftp: USER=anonymous
PASS=YC@163.com
#-i: 非交互模式 -n: 关闭自动登录
ftp –i –n << END
open ftp.163.com
user $USER $PASS
cd /pub
close
END
#END标记输入结束
```

2. 字符串 I/O 操作

字符串输出: echo

命令选项: -e: 启动转义序列 -n: 取消输出后换行 (前面已经用到过)

3. 字符串输入: read 可以用于用户交互输入, 也可以用来一次处理文本文件中的一行

命令选项:

```
ct@ehbio:~$ read -p "Enter the best tutorial: " tutorial
Enter the best tutorial: Sheng Xin Bao Dian
ct@ehbio:~$ echo $tutorial
Sheng Xin Bao Dian
```

```
# 隐藏输入内容
ct@ehbio:~$ read -s -p "Enter your password: " password
Enter your password:
ct@ehbio:~$ echo $password
haha
```

4.8 命令行处理命令行处理命令:

`getopts` 有两个参数, 第一个为字母和冒号组成的选项列表字符串, 第二个为一个变量名

选项列表字符串以冒号开头的选项字母排列组成, 如果一选项需要一个参数则该选项字母后跟一个冒号

`getopts` 分解第一参数, 依次将选项摘取出来赋给第二个参数变量

如果某选项有参数, 则读取参数到内置变量 `OPTARG` 中内置变量 `OPTIND` 保存着将被处理的命令行参数（位置参数）的数值选项列表处理完毕 `getopts` 返回 `1`, 否则返回 `0` 如:

在我们推出的一步绘图脚本里面，就是使用 `Bash` 封装的 R 脚本，通过修改命令行参数，完成热图、柱状图、线图、Venn 图、火山图、泡泡图等图形的绘制和定制。

```
while getopts "hf:m:a:A:b:I:t:x:l:j:J:d:F:G:H:P:L:y:V:D:c:C:B:X:Y:R:w:u:r:o:O:s:S:p:z:Z:v:e:E:i
do
        case $OPTION in
                h)
                        usage
                        exit 1
                        ;;
                f)
                        file=$OPTARG
                        ;;
                m)
                        melted=$OPTARG
                        ;;
                .
                .
                .
                ?)
                        usage
                        exit 1
                        ;;
        esac
```

```
done
```

4.9 进程和作业控制

命令行运行监测和软件安装文中讲述了部分监测命令。

如果一个命令需要运行比较久，一般使用 `nohup cmmand &` 来放入后台不中断运行，这样推出终端也不影响程序。

`command &` 是把程序放入后台。

`jobs`: 查看后台进程

`bg`: 显示后台进程, 即用 Ctrl+z 挂起或 '命令 &' 执行的进程

`fg job_id`: 将后台进程转到前台执行

`kill -9 process_id`: 强制杀掉某个进程

5 Bioinfo tools

5.1 寻找 Cas9 的同源基因并进行进化分析

见 PPT

5.2 emboss 的使用

EMBOSS 是欧洲分子生物学开放软件包，主要做序列比对，数据库搜搜，蛋白 motif 分析和功能域分析，序列模式搜索，引物设计等。

emboss 可以使用源码编译安装或用 Conda 安装，在前面的基础课中已有过讲述。

下载地址 ftp://emboss.open-bio.org/pub/EMBOSS/emboss-latest.tar.gz。

下载地址 http://primer3.sourceforge.net/。

```
# Make sure bioconda channel has added
# http://blog.genesino.com/2017/09/bioconda/
ct@ehbio:~$ conda install emboss
ct@ehbio:~$ url=https://sourceforge.net/projects/primer3/files/primer3/2.3.7/
ct@ehbio:~$ wget ${url}primer3-2.3.7.tar.gz -O primer3-2.3.7.tar.gz
ct@ehbio:~$ tar xvzf primer3-2.3.7.tar.gz
ct@ehbio:~$ cd primer3-2.3.7/src
ct@ehbio:~$ make all
# 确保~/bin在环境变量中
ct@ehbio:~$ ln -s `pwd`/primer3_core ~/bin/primer32_core

# Error: thermodynamic approach chosen,  but path to thermodynamic parameters not specified
ct@ehbio:~$ mkdir /opt/primer3_config
ct@ehbio:~$ cp -R primer3-2.3.7/src/primer3_config/* /opt/primer3_config
```

测试数据

```
ct@ehbio:~$ cat <<END >test.fa
>comp24_c0_seq1
TTACTCTCATCCTCCCCTTGTTGAAAGATTGGCTGCAATTGATGAACCCGATAAGAAGGTCAACTAAGAGAAGTGTAC
TTTTACGCATGGCATGGCATGGCGAGATATGGCTGTAATATGAGTATTATTTTCCTATGTTGCTACCGATATTTTCTA
```

Table 5.1: Popular applications of EMBOSS.

Popular.applications	Functions
prophet	Gapped alignment for profiles.
infoseq	Displays some simple information about sequences.
water	Smith-Waterman local alignment.
pepstats	Protein statistics.
showfeat	Show features of a sequence.
palindrome	Looks for inverted repeats in a nucleotide sequence.
eprimer3	Picks PCR primers and hybridization oligos.
profit	Scan a sequence or database with a matrix or profile.
extractseq	Extract regions from a sequence.
marscan	Finds MAR/SAR sites in nucleic sequences.
tfscan	Scans DNA sequences for transcription factors.
patmatmotifs	Compares a protein sequence to the PROSITE motif database.
showdb	Displays information on the currently available databases.
wossname	Finds programs by keywords in their one-line documentation.
abiview	Reads ABI file and display the trace.
tranalign	Align nucleic coding regions given the aligned proteins.

```
TTTGCATATGAAAATTCCAAACCCAGAGTTAGGGGCCATATCTAAAGGGAATTTGCTAACGAGTAAATGGGAAAATAG
GAAATGTCAGAGGAGAtagcctagcctagcctagcctagccTCGCCTCATGTAACGAAATACAATTTAAATTTTGCTT
TACAGCTAATAGTCAGACTTTACATTTTGCTAAAA
END
```

设计引物

```
ct@ehbio:~$ eprimer32 -sequence test.fa -outfile test.fa.primer \
    -targetregion 0,371 -optsize 20 -numreturn 3 \
    -minsize 15 -maxsize 25 \
    -opttm 50 -mintm 45 -maxtm 55 \
    -psizeopt 200 -prange 100-280
```

引物结果

```
# EPRIMER32 RESULTS FOR comp24_c0_seq1

#                     Start  Len    Tm     GC%   Sequence

   1 PRODUCT SIZE: 200
     FORWARD PRIMER     126   20  50.17  35.00  TATTTTCCTATGTTGCTACC
```

```
     REVERSE PRIMER     306    20   50.01   30.00   ACTATTAGCTGTAAAGCAAA


   2 PRODUCT SIZE: 199
     FORWARD PRIMER     134    20   49.88   30.00   TATGTTGCTACCGATATTTT

     REVERSE PRIMER     313    20   50.30   35.00   AAGTCTGACTATTAGCTGTA


   3 PRODUCT SIZE: 198
     FORWARD PRIMER     134    20   49.88   30.00   TATGTTGCTACCGATATTTT

     REVERSE PRIMER     312    20   50.30   35.00   AGTCTGACTATTAGCTGTAA
```

整理引物格式位 PrimerSearch 需要的格式

```
ct@ehbio:~$ awk '{if($0~/EPRIMER32/) {seq_name=$5;count=1;} else \
    if($0~/FORWARD PRIMER/) forward=$7; else if ($0~/REVERSE PRIMER/) \
    {reverse=$7; printf("%s@%d\t%s\t%s\n", seq_name,count,forward, reverse); \
    count+=1;} }' test.fa.primer >all_primer_file


ct@ehbio:~$ cat all_primer_file
comp24_c0_seq1@1        TATTTTCCTATGTTGCTACC    ACTATTAGCTGTAAAGCAAA
comp24_c0_seq1@2        TATGTTGCTACCGATATTTT    AAGTCTGACTATTAGCTGTA
comp24_c0_seq1@3        TATGTTGCTACCGATATTTT    AGTCTGACTATTAGCTGTAA
```

模拟 PCR

```
ct@ehbio:~$ primersearch -seqall test.fa -infile all_primer_file \
    -mismatchpercent 5 -outfile test.database.primerSearch


Primer name comp24_c0_seq1@1
Amplimer 1
    Sequence: comp24_c0_seq1

    TATTTTCCTATGTTGCTACC hits forward strand at 126 with 0 mismatches
    ACTATTAGCTGTAAAGCAAA hits reverse strand at [23] with 0 mismatches
    Amplimer length: 200 bp

Primer name comp24_c0_seq1@2
Amplimer 1
```

```
    Sequence: comp24_c0_seq1

    TATGTTGCTACCGATATTTT hits forward strand at 134 with 0 mismatches
    AAGTCTGACTATTAGCTGTA hits reverse strand at [16] with 0 mismatches
    Amplimer length: 199 bp

Primer name comp24_c0_seq1@3
Amplimer 1
    Sequence: comp24_c0_seq1

    TATGTTGCTACCGATATTTT hits forward strand at 134 with 0 mismatches
    AGTCTGACTATTAGCTGTAA hits reverse strand at [17] with 0 mismatches
    Amplimer length: 198 bp
```

needleall 读入两个文件，第一个文件的每个序列都与第二个文件的每个序列进行全局比对，采用 Needleman-Wunsch 算法。

```
# 生成测试数据
ct@ehbio:~$ cat <<END >generateRandom.awk
BEGIN{srand(seed); seq[0]="A"; seq[1]="C"; seq[2]="G"; seq[3]="T"}
    {for(i=1;i<=chrNum;i++)
        {print ">"label""i; len=(10-int(rand()*10)%2)/10*expected_len;
        for(j=0;j<=len;j++) printf("%s", seq[int(rand()*10)%4]); print "";
        }
    }
END

ct@ehbio:~$ echo 1 | awk -v seed=$RANDOM -v label=mm -v chrNum=2 \
             -v expected_len=40 -f generateRandom.awk >test1.fa
ct@ehbio:~$ echo 1 | awk -v seed=$RANDOM -v label=hs -v chrNum=2 \
             -v expected_len=40 -f generateRandom.awk >test2.fa


ct@ehbio:~$ cat test1.fa
>mm1
GTATACATCCGTAATCGGATAAAAGCGTACTATGGCG
>mm2
TAATTTCCCATGCACTATCACAACCCCTCGGATCAGACGCC
ct@ehbio:~$ cat test2.fa
>hs1
GCAAACGATTGGCCGGACGTCATCACTCCCCTCCGCGGATG
>hs2
CACAGTCCACGCTTTAAACGTACGAACAGACTTCCTT


# 输出格式见： http://emboss.sourceforge.net/docs/themes/AlignFormats.html
```

```
# Both fa and fq are supported
# -auto: 关闭弹出选项
ct@ehbio:~$ needleall -asequence test1.fa -bsequence test2.fa -gapopen 10 -gapextend 0.5 \
    -outfile test12.needle.alignment -auto -aformat3 pair


ct@ehbio:~$ cat test12.needle.alignment
########################################
# Program: needleall
# Rundate: Fri 30 Mar 2018 13:49:30
# Commandline: needleall
#    -asequence test1.fa
#    -bsequence test2.fa
#    -auto
#    -aformat3 pair
# Align_format: pair
# Report_file: test1.needleall
########################################

#=======================================
#
# Aligned_sequences: 2
# 1: mm1
# 2: hs1
# Matrix: EDNAFULL
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 62
# Identity:      15/62 (24.2%)
# Similarity:    15/62 (24.2%)
# Gaps:          46/62 (74.2%)
# Score: 27.0
#
#
#=======================================

mm1                 1 ------------------GT-ATACA------TCCGTAATCGGATAAAAG     25
                                        || || ||      ||||    |||||.
hs1                 1 GCAAACGATTGGCCGGACGTCAT-CACTCCCCTCCG----CGGATG----     41

mm1                26 CGTACTATGGCG     37

hs1                42 ------------     41


#=======================================
```

```
#
# Aligned_sequences: 2
# 1: mm2
# 2: hs1
# Matrix: EDNAFULL
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 51
# Identity:      23/51 (45.1%)
# Similarity:    23/51 (45.1%)
# Gaps:          20/51 (39.2%)
# Score: 41.0
#
#
#=======================================

mm2                1 -----TAATTTCCCATGCAC--TATCACAACCCCT---CGGATCAGACGC      40
                         ..|||..||  |.||  .||||| .|||||  |||||.
hs1                1 GCAAACGATTGGCC--GGACGTCATCAC-TCCCCTCCGCGGATG------      41

mm2               41 C      41

hs1               42 -      41


#=======================================
#
# Aligned_sequences: 2
# 1: mm1
# 2: hs2
# Matrix: EDNAFULL
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 51
# Identity:      18/51 (35.3%)
# Similarity:    18/51 (35.3%)
# Gaps:          28/51 (54.9%)
# Score: 26.0
#
#
#=======================================

mm1                1 GTATACA-TCCGTAATCGGATAAAAGCGTACTATGGCG------------      37
                       .||| |||    .||..|.||| ||||      ||
hs2                1 ---CACAGTCC----ACGCTTTAAA-CGTA------CGAACAGACTTCCT      36
```

```
mm1                38 -      37

hs2                37 T      37


#=======================================
#
# Aligned_sequences: 2
# 1: mm2
# 2: hs2
# Matrix: EDNAFULL
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 55
# Identity:      18/55 (32.7%)
# Similarity:    18/55 (32.7%)
# Gaps:          32/55 (58.2%)
# Score: 36.0
#
#=======================================

mm2                 1 TAATTTCCCATGCACTATCACAACCC---CT-----CG---GATCAGACG     39
                                  ||||..||   ||     ||   ||.||||||.
hs2                 1 ----------------CACAGTCCACGCTTTAAACGTACGAACAGACT     32

mm2                40 CC---     41
                      .|
hs2                33 TCCTT    37


#---------------------------------------
#---------------------------------------


ct@ehbio:~$ needleall -asequence test1.fa -bsequence test2.fa -gapopen 10 -gapextend 0.5 \
    -outfile test12.needle.score -auto


# 序列1 序列2 比对长度 比对得分
ct@ehbio:~$ cat test12.needle.score
mm1 hs1 62 (27.0)
mm2 hs1 51 (41.0)
mm1 hs2 51 (26.0)
mm2 hs2 55 (36.0)
```

5.3 使用 samtools 计算 SNP

1. 安装 samtools 和 bedtools

```
ct@ehbio:~$ conda install samtools
ct@ehbio:~$ conda install bedtools
```

2. 产生随机的基因组文件。

```
# srand: 随机数发生器。设置固定的种子，保证每次出来的结果一致
# rand: 返回[0,1)之间的随机数，包含0不包含1
ct@ehbio:~$ echo 1 | awk -v seed=1 -v label=chr -v chrNum=4 \
              -v expected_len=60000 -f generateRandom.awk >genome.fa
# 显示前60个碱基
ct@ehbio:~$ ct@ehbio:~/bio$ head genome.fa | cut -c 1-60
>chr1
GACCCACACTACGAGGCTCCCAACGATCAGGATTCCTATTCCCTCCTCGCTACCGGAAAA
>chr2
AGCCCTTACACCATCTGAGTCTGGCACACTTTTAGAACATCTACCCGTCACGAACAAGAA
>chr3
GTACAAGGCCCGGGGCTCGGACATTAAGCTCCTCCACTCAGCAGTCAAGTCAAACGAACA
>chr4
ACGCCCGTCAATTAGAGGCATTCAAAGACACCCGCCCGTGCTACAATAGGTACTACAACC
```

3. 产生随机的测序文件

```
# -N: 获得40K read pairs
# mut.txt: 突变位点或区域
ct@ehbio:~$ wgsim -N 40000 genome.fa ehbio_1.fq ehbio_2.fq > mut.txt
```

```
# FASTQ格式序列，4行一组
# 第一行以@开头，后面为序列名字，
# 第二行为序列
# 第三行+开头，后面一般无内容；若有，也是序列名字
# 第四行，质量值，对应序列中每个碱基的测序准确度
ct@ehbio:~$ head ehbio_1.fq
@chr1_17674_18124_2:0:0_2:0:0_0/1
TCGTTCAGTGGTGGTTACTCGTAGGGTCTTCCATCTGAGGCGGGCGAGCGGACGCCTTTTCTGCCTCCAG
+
2222222222222222222222222222222222222222222222222222222222222222222222
```

```
@chr1_29806_30221_2:0:0_0:0:0_1/1
TTAAGTGTGCTTGGACAACGGATATGCAAGTGTCTTTGATATATCGTTAGGGATAGGTTAATTAAGGGTC
+
2222222222222222222222222222222222222222222222222222222222222222222222
```

```
# 获得的突变文件如下
# Check IUPAC here: http://www.bioinformatics.org/sms/iupac.html
Col1: chromosome
Col2: position
Col3: original base
Col4: new base (IUPAC codes indicate heterozygous)
Col5: which genomic copy/haplotype
ct@ehbio:~$ head mut.txt
chr1    6274    T   C   -
chr1    6923    C   Y   +
chr1    7022    C   Y   +
chr1    10426   A   W   +
chr1    11130   C   S   +
chr1    12135   G   R   +
```

4. 创建基因组索引

```
ct@ehbio:~$ bwa index genome.fa
# samtools fadix快速获取某区域序列
ct@ehbio:~$ samtools faidx genome.fa
```

5. 序列比对回基因组

```
ct@ehbio:~$ bwa mem -t 3 genome.fa ehbio_1.fq ehbio_2.fq | gzip >map.sam.gz
```

6. 筛选比对上的高质量 reads

```
ct@ehbio:~$ samtools view -F4 -q1 -b map.sam.gz -o map.bam
# 下面2个排序用法都可以，看使用的samtools版本
ct@ehbio:~$ #samtools sort -@ 2 map.bam map.sortP
ct@ehbio:~$ samtools sort -@ 2 -o map.sortP.bam map.bam
ct@ehbio:~$ samtools index map.sortP.bam
```

7. 统计比对 reads 数

```
ct@ehbio:~$ samtools view -c map.sortP.bam
79998
```

8. 统计未比对上的 reads 数

```
ct@ehbio:~$ samtools view -c -f 4 map.sam.gz
```

9. 统计比对到正链的 reads 数

```
ct@ehbio:~$ samtools view -c -F 16 map.sam.gz
40001
```

10. 获取 properly-paired 的 reads 数

```
ct@ehbio:~$ samtools view -f2 -F 256 -c map.sortP.bam
79996
```

11. 查看每个位置碱基比对或错配情况

```
# -Q 0: 测试数据使用，默认为-Q 13，表示过滤掉低质量测序碱基
ct@ehbio:~$ samtools mpileup -f genome.fa -Q 0 map.sortP.bam | less
# chrName   coordinate  ref_base    coverage    reads_base  reads_quality
chr1    5       C       1           ^].     2
chr1    6       A       2           .^].    22
chr1    7       C       2           ..      22
chr1    8       A       2           ..      22
chr1    9       C       2           G.      22
chr1    10      T       3           ..^].   222
chr1    11      A       3           ...     222
chr1    12      C       4           ...^].  2222
chr1    13      G       4           ....    2222
chr1    14      A       4           ....    2222
chr1    15      G       4           ....    2222
chr1    16      G       4           ....    2222
```

mpileup format (http://samtools.sourceforge.net/pileup.shtml)

测序碱基列解释：

1. 点 (.) 代表匹配正链碱基
2. 逗号 (,) 代表匹配负链碱基
3. 大写字母 (ACGTN) 表示正链错配
4. 小写字母 (acgtn) 表示负链错配
5. 模式\+[0-9]+[ACGTNacgtn]+ 表示在当前参考位置和下一个参考位置之间有插入，插入碱基数是 + 后面的证书，插入碱基是数字后面的字母串。下面展示的是 2 bp 的插入

   seq2 156 A 11 .$……+2AG.+2AG.+2AGGG <975;:<<<<<

6. 模式-[0-9]+[ACGTNacgtn]+' 参考基因组存在碱基缺失。下面展示的是 4 bp'缺失：

   seq3 200 A 20 „„,..,.-4CACC.-4CACC….,.„.^~. ==<<<<<<<<<<<::<;2<<

7. 符号 ^ 表示测序序列起始位置落于此 (A symbol ^' marks the start of a read segment which is a contiguous subsequence on the read separated byN/S/H' CIGAR operations). 后面跟随的符号的 ASCII 值减去 33 表示该位置碱基的质量。符号 '$' 表示测序序列片段的终止。主要用于从 pileup 文件中获得原始测序 reads。

12. 输出 vcf 格式

```
# 获得未压缩的vcf格式，方便查看
ct@ehbio:~$ samtools mpileup -f genome.fa -Q 0 -vu map.sortP.bam >snp.vcf
```

5.4 Bedtools 使用

Bedtools是处理基因组信息分析的强大工具集合。

```
bedtools: flexible tools for genome arithmetic and DNA sequence analysis.
usage:    bedtools <subcommand> [options]

The bedtools sub-commands include:

[ Genome arithmetic ]
    intersect     Find overlapping intervals in various ways.
                  求区域之间的交集，可以用来注释peak，计算reads比对到的基因组区域
                  不同样品的peak之间的peak重叠情况。
    window        Find overlapping intervals within a window around an interval.
    closest       Find the closest, potentially non-overlapping interval.
                  寻找最近但可能不重叠的区域
    coverage      Compute the coverage over defined intervals.
                  计算区域覆盖度
    map           Apply a function to a column for each overlapping interval.
    genomecov     Compute the coverage over an entire genome.
```

```
    merge          Combine overlapping/nearby intervals into a single interval.
                   合并重叠或相接的区域
    cluster        Cluster (but don't merge) overlapping/nearby intervals.
    complement     Extract intervals _not_ represented by an interval file.
                   获得互补区域
    subtract       Remove intervals based on overlaps b/w two files.
                   计算区域差集
    slop           Adjust the size of intervals.
                   调整区域大小，如获得转录起始位点上下游3 K的区域
    flank          Create new intervals from the flanks of existing intervals.
    sort           Order the intervals in a file.
                   排序，部分命令需要排序过的bed文件
    random         Generate random intervals in a genome.
                   获得随机区域，作为背景集
    shuffle        Randomly redistrubute intervals in a genome.
                   根据给定的bed文件获得随机区域，作为背景集
    sample         Sample random records from file using reservoir sampling.
    spacing        Report the gap lengths between intervals in a file.
    annotate       Annotate coverage of features from multiple files.

[ Multi-way file comparisons ]
    multiinter     Identifies common intervals among multiple interval files.
    unionbedg      Combines coverage intervals from multiple BEDGRAPH files.

[ Paired-end manipulation ]
    pairtobed      Find pairs that overlap intervals in various ways.
    pairtopair     Find pairs that overlap other pairs in various ways.

[ Format conversion ]
    bamtobed       Convert BAM alignments to BED (& other) formats.
    bedtobam       Convert intervals to BAM records.
    bamtofastq     Convert BAM records to FASTQ records.
    bedpetobam     Convert BEDPE intervals to BAM records.
    bed12tobed6    Breaks BED12 intervals into discrete BED6 intervals.

[ Fasta manipulation ]
    getfasta       Use intervals to extract sequences from a FASTA file.
                   提取给定位置的FASTA序列
    maskfasta      Use intervals to mask sequences from a FASTA file.
    nuc            Profile the nucleotide content of intervals in a FASTA file.

[ BAM focused tools ]
    multicov       Counts coverage from multiple BAMs at specific intervals.
    tag            Tag BAM alignments based on overlaps with interval files.

[ Statistical relationships ]
    jaccard        Calculate the Jaccard statistic b/w two sets of intervals.
```

```
                计算数据集相似性
    reldist     Calculate the distribution of relative distances b/w two files.
    fisher      Calculate Fisher statistic b/w two feature files.

[ Miscellaneous tools ]
    overlap     Computes the amount of overlap from two intervals.
    igv         Create an IGV snapshot batch script.
                用于生成一个脚本，批量捕获IGV截图
    links       Create a HTML page of links to UCSC locations.
    makewindows Make interval "windows" across a genome.
                把给定区域划分成指定大小和间隔的小区间 (bin)
    groupby     Group by common cols. & summarize oth. cols. (~ SQL "groupBy")
                分组结算，不只可以用于bed文件。
    expand      Replicate lines based on lists of values in columns.
    split       Split a file into multiple files with equal records or base pairs.

[ General help ]
    --help      Print this help menu.
    --version   What version of bedtools are you using?.
    --contact   Feature requests, bugs, mailing lists, etc.
```

1. 安装 bedtools

```
ct@ehbio:~$ conda install bedtools
```

2. 获得测试数据集 (http://quinlanlab.org/tutorials/bedtools/bedtools.html)

```
ct@ehbio:~$ mkdir bedtools
ct@ehbio:~$ cd bedtools
ct@ehbio:~$ url=https://s3.amazonaws.com/bedtools-tutorials/web
ct@ehbio:~/bedtools$ curl -O ${url}/maurano.dnaseI.tgz
ct@ehbio:~/bedtools$ curl -O ${url}/cpg.bed
ct@ehbio:~/bedtools$ curl -O ${url}/exons.bed
ct@ehbio:~/bedtools$ curl -O ${url}/gwas.bed
ct@ehbio:~/bedtools$ curl -O ${url}/genome.txt
ct@ehbio:~/bedtools$ curl -O ${url}/hesc.chromHmm.bed
```

3. 交集 (intersect)

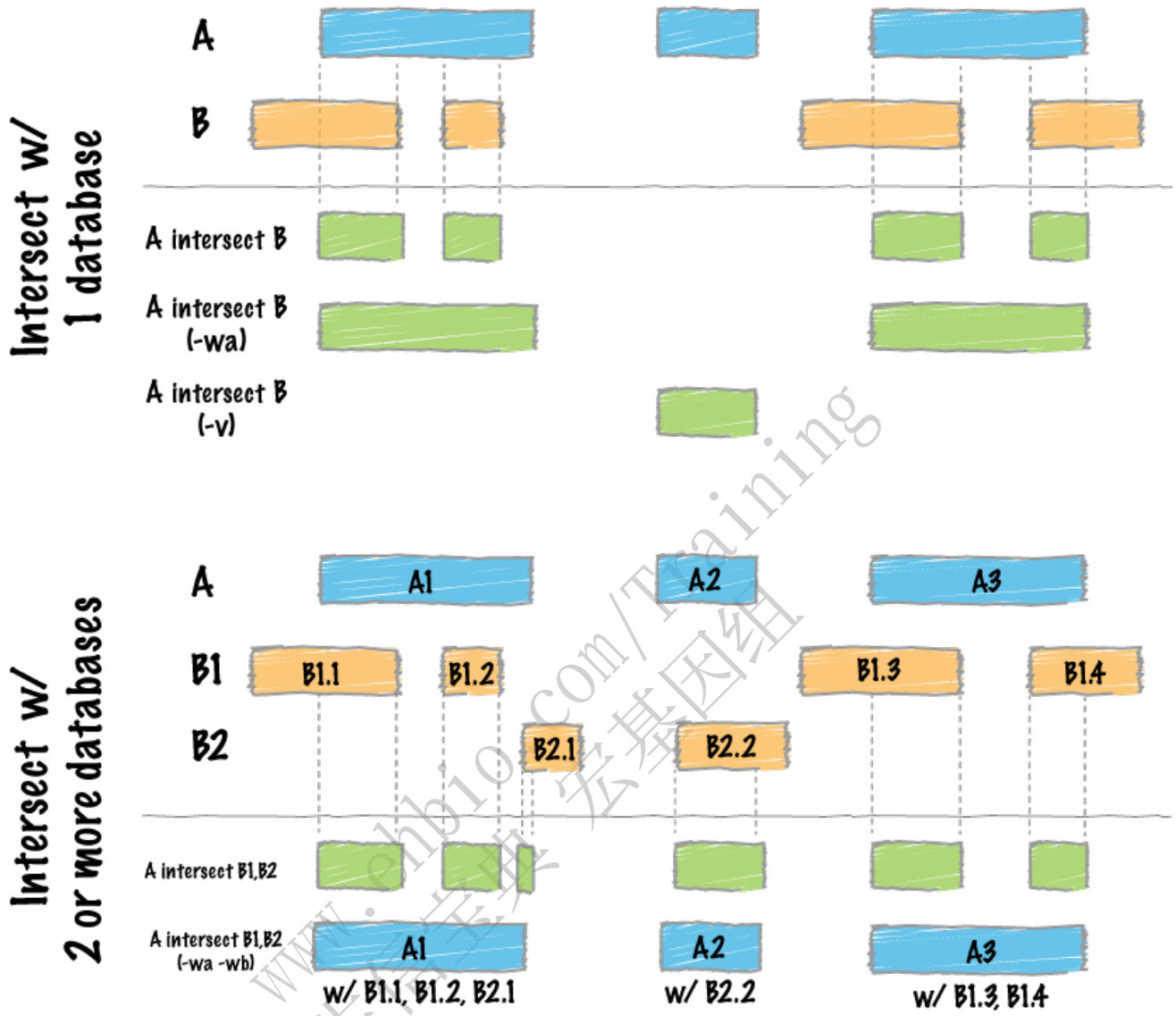

Figure 5.1: bedtools intersect

查看输入文件，bed 格式，至少三列，分别是染色体，起始位置 (0-based, 包括)，终止位置 (1-based，不包括)。第四列一般为区域名字，第五列一般为空，第六列为链的信息。更详细解释见http://www.genome.ucsc.edu/FAQ/FAQformat.html#format1。

自己做研究 CpG 岛信息可以从 UCSC 的 Table Browser 获得，具体操作见http://blog.genesino.com/2013/05/ucsc-usages/。

```
ct@ehbio:~/bedtools$ head -n 3 cpg.bed exons.bed
==> cpg.bed <==
chr1    28735   29810   CpG:_116
```

```
chr1    135124  135563  CpG:_30
chr1    327790  328229  CpG:_29

==> exons.bed <==
chr1    11873   12227   NR_046018_exon_0_0_chr1_11874_f 0   +
chr1    12612   12721   NR_046018_exon_1_0_chr1_12613_f 0   +
chr1    13220   14409   NR_046018_exon_2_0_chr1_13221_f 0   +
```

获得重叠区域 (既是外显子，又是 CpG 岛的区域)

```
ct@ehbio:~/bedtools$ bedtools intersect -a cpg.bed -b exons.bed | head -5
chr1    29320   29370   CpG:_116
chr1    135124  135563  CpG:_30
chr1    327790  328229  CpG:_29
chr1    327790  328229  CpG:_29
chr1    327790  328229  CpG:_29
```

输出重叠区域对应的原始区域 (与外显子存在交集的 CpG 岛)

```
ct@ehbio:~/bedtools$ bedtools intersect -a cpg.bed -b exons.bed -wa -wb > | head -5
```

- chr1 28735 29810 CpG:_116 chr1 29320 29370 NR_024540_exon_10_0_chr1_29321_r 0 -
- chr1 135124 135563 CpG:_30 chr1 134772 139696 NR_039983_exon_0_0_chr1_134773_r 0 -
- chr1 327790 328229 CpG:_29 chr1 324438 328581 NR_028322_exon_2_0_chr1_324439_f 0 +
- chr1 327790 328229 CpG:_29 chr1 324438 328581 NR_028325_exon_2_0_chr1_324439_f 0 +
- chr1 327790 328229 CpG:_29 chr1 327035 328581 NR_028327_exon_3_0_chr1_327036_f 0 +

计算重叠碱基数

```
ct@ehbio:~/bedtools$ bedtools intersect -a cpg.bed -b exons.bed -wo | head -10
```

- chr1 28735 29810 CpG:_116 chr1 29320 29370 NR_024540_exon_10_0_chr1_29321_r 0 - 50
- chr1 135124 135563 CpG:_30 chr1 134772 139696 NR_039983_exon_0_0_chr1_134773_r 0 - 439
- chr1 327790 328229 CpG:_29 chr1 324438 328581 NR_028322_exon_2_0_chr1_324439_f 0 + 439
- chr1 327790 328229 CpG:_29 chr1 324438 328581 NR_028325_exon_2_0_chr1_324439_f 0 + 439
- chr1 327790 328229 CpG:_29 chr1 327035 328581 NR_028327_exon_3_0_chr1_327036_f 0 + 439
- chr1 713984 714547 CpG:_60 chr1 713663 714068 NR_033908_exon_6_0_chr1_713664_r 0 - 84
- chr1 762416 763445 CpG:_115 chr1 761585 762902 NR_024321_exon_0_0_chr1_761586_r 0 - 486
- chr1 762416 763445 CpG:_115 chr1 762970 763155 NR_015368_exon_0_0_chr1_762971_f 0 + 185

- chr1 762416 763445 CpG:_115 chr1 762970 763155 NR_047519_exon_0_0_chr1_762971_f 0 + 185
- chr1 762416 763445 CpG:_115 chr1 762970 763155 NR_047520_exon_0_0_chr1_762971_f 0 + 185

计算第一个 (-a)bed 区域有多少个重叠的第二个 (-b)bed 文件中有多少个区域

```
ct@ehbio:~/bedtools$ bedtools intersect -a cpg.bed -b exons.bed -c | head
chr1    28735   29810   CpG:_116    1
chr1    135124  135563  CpG:_30 1
chr1    327790  328229  CpG:_29 3
chr1    437151  438164  CpG:_84 0
chr1    449273  450544  CpG:_99 0
chr1    533219  534114  CpG:_94 0
chr1    544738  546649  CpG:_171    0
chr1    713984  714547  CpG:_60 1
chr1    762416  763445  CpG:_115    10
chr1    788863  789211  CpG:_28 9
```

另外还有`-v` 取出不重叠的区域, `-f` 限定重叠最小比例，`-sorted` 可以对按 `sort -k1,1 -k2,2n` 排序好的文件加速操作。

同时对多个区域求交集 (可以用于 peak 的多维注释)

```
# -names标注注释来源
# -sorted: 如果使用了这个参数，提供的一定是排序好的bed文件
ct@ehbio:~/bedtools$ bedtools intersect -a exons.bed \
    -b cpg.bed gwas.bed hesc.chromHmm.bed -sorted -wa -wb -names cpg gwas chromhmm \
    | head -10000  | tail -10
```

- chr1 27632676 27635124 NM_001276252_exon_15_0_chr1_27632677_chromhmm chr1 27633213 27635013 5_Strong_Enhancer
- chr1 27632676 27635124 NM_001276252_exon_15_0_chr1_27632677_chromhmm chr1 27635013 27635413 7_Weak_Enhancer
- chr1 27632676 27635124 NM_015023_exon_15_0_chr1_27632677_f chromhmm chr1 27632613 27632813 6_Weak_Enhancer
- chr1 27632676 27635124 NM_015023_exon_15_0_chr1_27632677_f chromhmm chr1 27632813 27633213 7_Weak_Enhancer
- chr1 27632676 27635124 NM_015023_exon_15_0_chr1_27632677_f chromhmm chr1 27633213 27635013 5_Strong_Enhancer
- chr1 27632676 27635124 NM_015023_exon_15_0_chr1_27632677_f chromhmm chr1 27635013 27635413 7_Weak_Enhancer
- chr1 27648635 27648882 NM_032125_exon_0_0_chr1_27648636_f cpg chr1 27648453 27649006 CpG:_63

- chr1 27648635 27648882 NM_032125_exon_0_0_chr1_27648636_f chromhmm chr1 27648613 27649413 1_Active_Promoter
- chr1 27648635 27648882 NR_037576_exon_0_0_chr1_27648636_f cpg chr1 27648453 27649006 CpG:_63
- chr1 27648635 27648882 NR_037576_exon_0_0_chr1_27648636_f chromhmm chr1 27648613 27649413 1_Active_Promoter

4. 合并区域

Figure 5.2: bedtools merge

`bedtools merge` 输入的是按 `sort -k1,1 -k2,2n` 排序好的 bed 文件。

只需要输入一个排序好的 bed 文件，默认合并重叠或邻接区域。

```
ct@ehbio:~/bedtools$ bedtools merge -i exons.bed | head -n 5
chr1    11873   12227
chr1    12612   12721
chr1    13220   14829
chr1    14969   15038
chr1    15795   15947
```

合并区域并输出此合并后区域是由几个区域合并来的

```
ct@ehbio:~/bedtools$ bedtools merge -i exons.bed -c 1 -o count | head -n 5
```

```
chr1    11873   12227   1
chr1    12612   12721   1
chr1    13220   14829   2
chr1    14969   15038   1
chr1    15795   15947   1
```

合并相距 90 nt 内的区域，并输出是由哪些区域合并来的

```
# -c: 指定对哪些列进行操作
# -o: 与-c对应，表示对指定列进行哪些操作
# 这里的用法是对第一列做计数操作，输出这个区域是由几个区域合并来的
# 对第4列做收集操作，记录合并的区域的名字，并逗号分隔显示出来
ct@ehbio:~/bedtools$ bedtools merge -i exons.bed -d 340 -c 1,4 -o count,collapse | head -4
chr1    11873   12227   1   NR_046018_exon_0_0_chr1_11874_f
chr1    12612   12721   1   NR_046018_exon_1_0_chr1_12613_f
chr1    13220   15038   3   NR_046018_exon_2_0_chr1_13221_f,NR_024540_exon_0_0_chr1_14362_r,NR_
chr1    15795   15947   1   NR_024540_exon_2_0_chr1_15796_r
```

5. 计算互补区域

给定一个全集，再给定一个子集，求另一个子集。比如给定每条染色体长度和外显子区域，求非外显子区域。给定基因区，求非基因区。给定重复序列，求非重复序列等。

重复序列区域的获取也可以用上面提供的链接 http://blog.genesino.com/2013/05/ucsc-usages/。

```
ct@ehbio:~/bedtools$ head genome.txt
chr1    249250621
chr10   135534747
chr11   135006516
chr11_gl000202_random   40103
chr12   133851895
chr13   115169878
chr14   107349540
chr15   102531392

ct@ehbio:~/bedtools$ bedtools complement -i exons.bed -g genome.txt | head -n 5
chr1    0   11873
chr1    12227   12612
chr1    12721   13220
chr1    14829   14969
chr1    15038   15795
```

6. 基因组覆盖广度和深度

计算基因组某个区域是否被覆盖，覆盖深度多少。有下图多种输出格式，也支持 RNA-seq 数据，计算 junction-reads 覆盖。

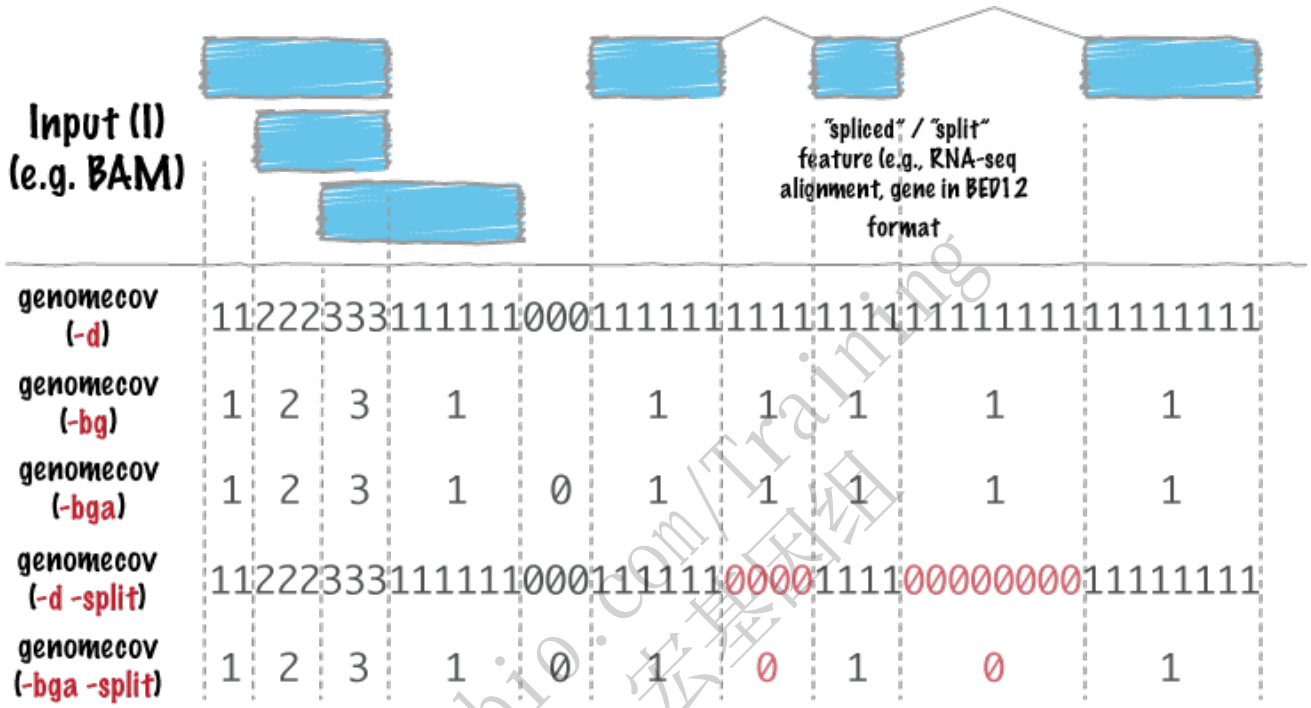

Figure 5.3: bedtools genomecov

genome.txt 里面的内容就是染色体及对应的长度。

```
# 对单行FASTA，可如此计算
# 如果是多行FASTA，则需要累加
ct@ehbio:~/bedtools$ awk 'BEGIN{OFS=FS="\t"}{\
    if($0~/>/) {seq_name=$0;sub(">","",seq_name);} \
    else {print seq_name,length;} }' ../bio/genome.fa | tee ../bio/genome.txt
chr1    60001
chr2    54001
chr3    54001
chr4    60001
ct@ehbio:~/bedtools$ bedtools genomecov -ibam ../bio/map.sortP.bam -bga \
    -g ../bio/genome.txt | head

# 这个warning很有意思，因为BAM中已经有这个信息了，就不需要提供了
*****
*****WARNING: Genome (-g) files are ignored when BAM input is provided.
```

```
*****
# bedgraph文件，前3列与bed相同，最后一列表示前3列指定的区域的覆盖度。
chr1    0   11  0
chr1    11  17  1
chr1    17  20  2
chr1    20  31  3
chr1    31  36  4
chr1    36  43  6
chr1    43  44  7
chr1    44  46  8
chr1    46  48  9
chr1    48  54  10
```

两个问题：

1. 怎么计算有多少基因组区域被测到了？

2. 怎么计算平均测序深度是多少？

7. 数据集相似性

`bedtools jaccard` 计算的是给定的两个 `bed` 文件之间交集区域 (intersection) 占总区域 (union-intersection) 的比例 (jaccard) 和交集的数目 (n_intersections)。

```
ct@ehbio:~/bedtools$ bedtools jaccard \
    -a fHeart-DS16621.hotspot.twopass.fdr0.05.merge.bed \
    -b fHeart-DS15839.hotspot.twopass.fdr0.05.merge.bed
intersection    union-intersection  jaccard n_intersections
81269248    160493950   0.50637 130852
```

小问题：1. 如何用 bedtools 其它工具算出这个结果？2. 如果需要比较的文件很多，怎么充分利用计算资源？

一个办法是使用 `for` 循环, 双层嵌套。这种用法也很常见，不管是单层还是双层 for 循环，都有利于简化重复运算。

```
ct@ehbio:~/bedtools$ for i in *.merge.bed; do \
    for j in *.merge.bed; do \
    bedtools jaccard -a $i -b $j | cut -f3 | tail -n +2 | sed "s/^/$i\t$j\t/"; \
    done; done >total.similarity
```

另一个办法是用 parallel，不只可以批量，更可以并行。

```
root@ehbio:~# yum install parallel.noarch
```

```
# parallel 后面双引号("")内的内容为希望用parallel执行的命令，
# 整体写法与Linux下命令写法一致。
# 双引号后面的 三个相邻冒号 (:::)默认用来传递参数的，可多个连写。
# 每个三冒号后面的参数会被循环调用，而在命令中的引用则是根据其出现的位置，分别用{1}, {2}
# 表示第一个三冒号后的参数，第二个三冒号后的参数。
#
# 这个命令可以替换原文档里面的整合和替换，相比于原文命令生成多个文件，这里对每个输出结果
# 先进行了比对信息的增加，最后结果可以输入一个文件中。
#
ct@ehbio:~/bedtools$ parallel "bedtools jaccard -a {1} -b {2} | awk 'NR> | cut -f 3 \
    | sed 's/^/{1}\t{2}\t/'" ::: `ls *.merge.bed` ::: `ls *.merge.bed`  >totalSimilarity.
2

# 替换掉无关信息
ct@ehbio:~/bedtools$ sed -i -e 's/.hotspot.twopass.fdr0.05.merge.bed//' \
    -e 's/.hg19//' totalSimilarity.2
```

原文档的命令，稍微有些复杂，利于学习不同命令的组合。使用时推荐使用上面的命令。

```
ct@ehbio:~/bedtools$ parallel "bedtools jaccard -a {1} -b {2} \
      | awk 'NR>1' | cut -f 3 \
      > {1}.{2}.jaccard" \
     ::: `ls *.merge.bed` ::: `ls *.merge.bed`
```

This command will create a single file containing the pairwise Jaccard measurements from all 400 tests.

```
find . \
    | grep jaccard \
    | xargs grep "" \
    | sed -e s"/\.\///" \
    | perl -pi -e "s/.bed./.bed\t/" \
    | perl -pi -e "s/.jaccard:/\t/" \
    > pairwise.dnase.txt
```

A bit of cleanup to use more intelligible names for each of the samples.

```
cat pairwise.dnase.txt \
| sed -e 's/.hotspot.twopass.fdr0.05.merge.bed//g' \
| sed -e 's/.hg19//g' \
> pairwise.dnase.shortnames.txt
```

Now let's make a 20x20 matrix of the Jaccard statistic. This will allow the data to play nicely with R.

```
awk 'NF==3' pairwise.dnase.shortnames.txt \
| awk '$1 ~ /^f/ && $2 ~ /^f/' \
| python make-matrix.py \
> dnase.shortnames.distance.matrix
```

5.5 SRA toolkit 使用

SRA toolkit https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=software根据服务器下载对应的二进制编码包。

CentOS 下地址：https://ftp-trace.ncbi.nlm.nih.gov/sra/sdk/2.9.0/sratoolkit.2.9.0-centos_linux64.tar.gz。

常用的是 `fastq-dump`，从 NCBI 的 SRA 数据库下载测序原始文件并转化为 FASTQ 格式供后续分析使用。

```
# --split-3: 若是双端测序则拆分
# --gzip: 拆分后压缩文件
# fastq-dump -v --split-3 --gzip SRR_number
fastq-dump -v --split-3 --gzip SRR502564
```

下面是我写的一个脚本，有 3 个用途。一是在 `fastq-dump` 下载过程中，如果断了可以再次启动下载；二是下载完成之后，给下载的文件进行重命名为方便识别的名字；三是清空下载缓存。

所需要的输入文件是一个 2 列文件，第一列为 SRR 号，第二列为样品名字，TAB 键分割。

```
SRR2952884      Y25_root
SRR2952883      Y18_root
SRR2952882      Y12_root
SRR2952881      Y05_root
```

```
#!/bin/bash

set -x
```

```
set -e
set -u

usage()
{
cat <<EOF >&2
${txtcyn}
Usage:

$0 options${txtrst}

${bldblu}Function${txtrst}:

This script is used to download sra files from a file containing SRA
accession numbers and transfer SRA to fastq format.

The format of input file:

SRR2952884      Y25_root
SRR2952883      Y18_root
SRR2952882      Y12_root
SRR2952881      Y05_root

The second column will be treated as the prefix of final fastq files.

${txtbld}OPTIONS${txtrst}:
    -f  Data file with format described above${bldred}[NECESSARY]${txtrst}
    -z  Is there a header[${bldred}Default TRUE${txtrst}]
EOF
}

file=

while getopts "hf:z:" OPTION
do
    case $OPTION in
        h)
            usage
            exit 1
            ;;
        f)
            file=$OPTARG
            ;;
        ?)
            usage
            exit 1
            ;;
```

```
    esac
done

if [ -z $file ]; then
    usage
    exit 1
fi


#IFS="\t"

cat $file | while read -r -a array
do
    sra="${array[0]}"
    name="${array[1]}"
    #echo $sra, $name
    #prefetch -v $sra
    #prefetch -v $sra
    #prefetch -v $sra
    #/bin/cp ~/ncbi/public/sra/${sra}.sra ${name}.sra
    while true
    do
        fastq-dump -v --split-3 --gzip ${sra}
        a=$?
        if [ "$a" == "0" ]; then break; fi
        sleep 5m
    done

    #/bin/cp ~/ncbi/public/sra/${sra}* .
    if [ "$a" == "0" ]
    then
        rename "${sra}" "${name}" ${sra}*
        /bin/rm ~/ncbi/public/sra/${sra}.sra
    fi
done
```


5.6 生信流程开发

最基本的是 Bash 脚本，把上面 call SNP 的命令放到一个 Bash 脚本文件中即可。另外可以使用 Makefile 和 Airflow 进行更高级一些的开发。

Airflow 使用见 http://blog.genesino.com/2016/05/airflow/。

一篇不错的英文 Makefile 教程 http://blog.genesino.com/2011/04/introduction-to-making-makefiles/。

5.7 数据同步和备份

5.7.1 scp

最简单的备份方式，就是使用 cp (本地硬盘) 或 scp (远程硬盘) 命令，给自己的结果文件新建一个拷贝；每有更新，再拷贝一份。具体命令如下：

```
cp -fur source_project project_bak
scp -r source_project user@remote_server_ip:project_bak
```

为了实现定期备份，我们可以把上述命令写入 crontab 程序中，设置每天的晚上 23:00 执行。对于远程服务器的备份，我们可以配置免密码登录，便于自动备份。后台输入免密码登录服务器，获取免密码登录服务器的方法。

```
# Crontab format
# Minute    Hour    Day    Month    Week    command
# * 表示每分/时/天/月/周
# 每天23:00 执行cp命令
0   23  *   *   *   cp -fur source_project project_bak
#   */2 表示每隔2分分/时/天/月/周执行命令
#   每隔24小时执行cp命令
0   */24    *   *   *   cp  -fur    source_project  project_bak
0   0   */1 *   *   scp -r  source_project  user@remote_server_ip:project_bak

# 另外crotab还有个特殊的时间
# @reboot: 开机运行指定命令
@reboot cmd
```


5.7.2 rsync

cp 或 scp 使用简单，但每次执行都会对所有文件进行拷贝，耗时耗力，尤其是需要拷贝的内容很多时，重复拷贝对时间和硬盘都是个损耗。

rsync 则是一个增量备份工具，只针对修改过的文件的修改过的部分进行同步备份，大大缩短了传输的文件的数量和传输时间。具体使用如下：

```
# 把本地project目录下的东西备份到远程服务器的/backup/project目录下
# 注意第一个project后面的反斜线，表示拷贝目录内的内容，不在目标目录新建project文件夹。注意与第二
# -a: archive mode, quals -rlptgoD
# -r: 递归同步
```

```
# -p: 同步时保留原文件的权限设置
# -u: 若文件在远端做过更新，则不同步，避免覆盖远端的修改
# -L: 同步符号链接链接的文件，防止在远程服务器出现文件路径等不匹配导致的软连接失效
# -t: 保留修改时间
# -v: 显示更新信息
# -z: 传输过程中压缩文件，对于传输速度慢时适用
rsync -aruLptvz --delete project/ user@remoteServer:/backup/project
rsync -aruLptvz --delete project user@remoteServer:/backup/
```

rsync 所做的工作为镜像，保证远端服务器与本地文件的统一。如果本地文件没问题，远端也不会有问题。但如果发生误删或因程序运行错误，导致文件出问题，而在同步之前又没有意识到的话，远端的备份也就没了备份的意义，因为它也被损坏了。误删是比较容易发现的，可以及时矫正。但程序运行出问题，则不一定了。

5.7.3 rdiff-backup

这里推荐一个工具 rdiff-backup 不只可以做增量备份，而且会保留每次备份的状态，新备份和上一次备份的差别，可以轻松回到之前的某个版本。唯一的要求就是，本地服务器和远端服务器需要安装统一版本的 rdiff-backup。另外还有 2 款工具 duplicity 和 ‘Rsnapshot 也可以做类似工作，但方法不一样，占用的磁盘空间也不一样，具体可查看原文链接中的比较。

具体的 rdiff-backup 安装和使用见http://mp.weixin.qq.com/s/c2cspK5b4sQScWYMBtG63g。

6 Bioinfo questions

- 进入 sxbd 目录，查看目录下的文件有哪些？
- 查看 GTF 文件的内容和格式 (如果没有，可在ftp://ftp.ensembl.org/pub/release-91/gtf/homo_sapiens/Homo_sapiens.GRCh38.91.gtf.gz下载。)
- 给每个区域的行首增加 `chr` 标签，并去掉 `#` 开头的行。

```
grep -v '^#' GRCh38.gtf | sed 's/^/chr/' >GRCh38.new.gtf
```

- 统计 GTF 文件中染色体数目？

```
ct@ehbio:~/sxbd$ cut -f1 GRCh38.new.gtf | uniq -c >chrCount.txt
ct@ehbio:~/sxbd$ awk '{print $2"\t"$1}' chrCount.txt
chr1    236802
chr2    194223
chr3    160954
chr4    106152
chr5    115953
chr6    116635
chr7    122750
chrX    81525
chr8    95038
chr9    91333
chr11   159595
chr10   94467
chr12   154317
chr13   38817
chr14   93293
chr15   97353
chr16   125435
chr17   166619
chr18   47336
chr20   57175
chr19   163738
chrY    7167
chr22   56380
chr21   28928
chrMT   144
chrKI270728.1   120
chrKI270727.1   88
chrKI270442.1   6
chrGL000225.1   3
chrGL000009.2   8
```

```
chrGL000194.1   26
chrGL000205.2   17
chrGL000195.1   27
chrKI270733.1   12
chrGL000219.1   12
chrGL000216.2   3
chrKI270744.1   3
chrKI270734.1   96
chrGL000213.1   52
chrGL000220.1   12
chrGL000218.1   8
chrKI270731.1   11
chrKI270750.1   3
chrKI270721.1   25
chrKI270726.1   11
chrKI270711.1   151
chrKI270713.1   20
ct@ehbio:~/sxbd$ awk '/chr[0-9XYM]/' chrCount.txt
 236802 chr1
 194223 chr2
 160954 chr3
 106152 chr4
 115953 chr5
 116635 chr6
 122750 chr7
  81525 chrX
  95038 chr8
  91333 chr9
 159595 chr11
  94467 chr10
 154317 chr12
  38817 chr13
  93293 chr14
  97353 chr15
 125435 chr16
 166619 chr17
  47336 chr18
  57175 chr20
 163738 chr19
   7167 chrY
  56380 chr22
  28928 chr21
    144 chrMT
ct@ehbio:~/ sxbd$ awk '/ chr[0-9XYM]/' chrCount.txt | sed 's/  *\ ([0-9]*\) \ (chr.*\)/
\2\t\1/'
chr1    236802
chr2    194223
```

```
chr3    160954
chr4    106152
chr5    115953
chr6    116635
chr7    122750
chrX    81525
chr8    95038
chr9    91333
chr11   159595
chr10   94467
chr12   154317
chr13   38817
chr14   93293
chr15   97353
chr16   125435
chr17   166619
chr18   47336
chr20   57175
chr19   163738
chrY    7167
chr22   56380
chr21   28928
chrMT   144
```

- 统计 GTF 文件中基因数目？

```
ct@ehbio:~/sxbd$ time cut -f 3 GRCh38.new.gtf | sort | uniq -c
 712821 CDS
1199851 exon
 144659 five_prime_utr
  58302 gene
    119 Selenocysteine
  83743 start_codon
  75493 stop_codon
 137545 three_prime_utr
 200310 transcript

real    0m8.314s
user    0m8.259s
sys 0m0.679s
# 更快
ct@ehbio:~/sxbd$ time awk '{a[$3]+=1}END{for(i in a) print i,a[i];}' GRCh38.new.gtf
five_prime_utr 144659
exon 1199851
three_prime_utr 137545
```

```
CDS 712821
gene 58302
start_codon 83743
Selenocysteine 119
stop_codon 75493
transcript 200310

real    0m1.898s
user    0m1.504s
sys 0m0.394s
```

• 计算 GTF 中外显子总长度？

```
# 这个是冗余的外显子，后面在计算非冗余外显子
ct@ehbio:~/sxbd$ awk '{if($3=="exon") sum+=$5-$4+1;}END\
    {print "Total redundant exon length", sum;}' GRCh38.new.gtf
```

• 计算 GTF 文件中每个基因的转录本数目？

```
# 第一个办法：基因和对应的转录本是排序好的，直接判断计算就可以
awk 'BEGIN{OFS=FS="\t";}{if($3=="gene" && count>0) {print count; count=0;} else \
    {if($3=="transcript") count+=1;}}END{print count}' GRCh38.new.gtf

# 第二个方法：取出所有基因和转录本名字
sed 's/"/\t/g' GRCh38.new.gtf | awk '$3=="transcript"' | cut -f 10,14 | cut -f 1 | uniq -
c

# 第三个方法：与第二个类似，但使用了groupBy

sed 's/"/\t/g' GRCh38.new.gtf | awk '$3=="transcript"' | cut -f 10,14 | \
    bedtools groupby -g 1 -c 1,2 -o count,collapse | head
ENSG00000223972 2   ENST00000456328,ENST00000450305
ENSG00000227232 1   ENST00000488147
ENSG00000278267 1   ENST00000619216
ENSG00000243485 2   ENST00000473358,ENST00000469289
ENSG00000284332 1   ENST00000607096
ENSG00000237613 2   ENST00000417324,ENST00000461467
ENSG00000268020 1   ENST00000606857
ENSG00000240361 2   ENST00000642116,ENST00000492842
ENSG00000186092 2   ENST00000641515,ENST00000335137
ENSG00000238009 5   ENST00000466430,ENST00000477740,ENST00000471248,ENST00000610542,ENST0000045

sed 's/"/\t/g' GRCh38.new.gtf | awk '$3=="transcript"' | cut -f 10,14 | \
    bedtools groupby -g 1 -c 1,2 -o count,collapse >geneTrCount.txt
```

- 计算 GTF 文件中基因所拥有的平均转录本数目

```
awk 'BEGIN{OFS=FS="\t"}{sum+=$2}END{print sum/NR;}' geneTrCount.txt
# 3.43573
```

- 生成一个多行 Fasta 测试序列供后续运算 (也可使用我们前面提供的脚本生成)

```
cat <<END >test.fa
>id1
ACGCATGGGGGGGGGGGGGGG
AGTATGGTCCAGTA
>id11
AGTGGGGGGGGGGGGGGGTTCCT
cgactaggcagtctgagttga
>id21
AGTGGGGGGGGGGGGGGGTTCCT
cgactaggcagtctgagttga
END
```

- `test.fa` 中的序列全转成大写。

```
# \U 转换为大写
# & 表示所有匹配内容
sed -i '/^[^>]/ s/.*/\U&/' test.fa
```

- 计算多行 FASTA 文件 `test.fa` 中每条序列长度，输出类似 `genome.txt` 格式的文件 (文件有两列，第一列为序列 ID，第二列为序列长度)

```
awk 'BEGIN{OFS="\t"; size=0;}{if($0~/>/) {if(size>0) print geneName,size; \
    geneName=$0; sub(">","",geneName); size=0;} else \
    {size+=length}}END{print geneName,size}' test.fa
```

- 多行 FASTA 转单行 FASTA 序列

```
# conditions?true_value:false_value 三目运算符，条件为真时，返回冒号前结果，否则冒号后结果
# 对于非第一行的>，输出前先输出一个换行
awk '/^>/&&NR>1{print "";}{printf "%s",/^>/?$0"\n":$0}' test.fa >singleLine.fa
```

• 取出单行 FASTA 文件中序列长度大于 40 的序列的名字？

```
awk 'BEGIN{OFS="\t";}{if($0~/>/) {geneName=$0; sub(">","",geneName); } else \
    {if (length($0)>40) print geneName;}}' singleLine.fa
```

• 分别用 `awk` 和 `grep` 从 `test.fa` 中提取给定 ID 对应的序列。

```
ID list:
id1
id21
```

• 利用 AWK 对基因表达数据进行标准化

```
cat <<END | sed 's/  */\t/g' >test.expr
ID  sampleA sampleB sampleC
A   1   2   3
B   4   5   6
C   6   7   8
D   10  11  12
END

# 单列
awk 'ARGIND==1{if(FNR>1) sum=sum+$2;}\
     ARGIND==2{if(FNR>1) {$3=$2/sum;} print $0;}' test.expr test.expr

# 多列
awk 'ARGIND==1{if(FNR>1) {for(i=2;i<=NF;i++) sum[i]=sum[i]+$i;}}\
     ARGIND==2{if(FNR>1) for(i=2;i<=NF;i++) {$i=$i/sum[i];} print $0;}' test.expr test.expr
```

• 写出 3 种写法，去掉上一题 `test.expr` 矩阵中的第一行？

```
awk 'FNR>1' test.expr

tail -n +2 test.expr

sed -n '2,$p' test.expr
```

• 分别用 `awk` 和 `sed` 给 `test.expr` 矩阵加上标题行？

```
sed '1 iheaderline' test.expr
awk '{if(FNR==1} print "headerline"; print $0' test.expr
```

- 给定一个 BAM 文件，怎么计算有多少基因组区域被测到了？平均测序深度是多少？

```
bedtools genomecov -ibam ../bio/map.sortP.bam -bga
```

- 如何使用 bedtools 的其它工具或其它 Linux 命令实现 bedtools jaccard 子功能？bedtools jaccard 计算的是给定的两个 bed 文件之间交集区域 (intersection) 占总区域 (union-intersection) 的比例 (jaccard) 和交集的数目 (n_intersections)。

```
ct@localhost:~/bedtools$ cat test1.bed
chr1    1   100
chr2    1   50
chr3    20  50
ct@localhost:~/bedtools$ cat test2.bed
chr1    50  150
chr3    1   50
chr4    1   50
chr5    1   50
ct@localhost:~/bedtools$ bedtools jaccard -a test1.bed -b test2.bed
intersection    union-intersection  jaccard n_intersections
80  296 0.27027 2
ct@localhost:~/bedtools$ bedtools intersect -a test1.bed -b test2.bed -wao \
    | awk '{sum+=$NF}END{print sum;}'
80
ct@localhost:~/bedtools$ cat test1.bed test2.bed | awk '{sum+=$3-$2}END{print sum;}'
376
```

7 Supplemental

serverInfo.sh

```
#!/bin/bash

echo "This lists the information of this computer."

echo

echo "Hostname is $(tput setaf 3)`hostname`$(tput sgr0),\
Ip address is $(tput setaf 3)\
`/sbin/ifconfig | sed -n '2p' | cut -d ':' -f 2 | cut -d ' ' -f 1`.
$(tput sgr0)"


nuclear=`uname -a | cut -d ' ' -f 3`
bitInfo=`uname -a | cut -d ' ' -f 12`

if test $bitInfo == "x86_64"; then
    bit=64
else
    bit=32
fi

echo "The $(tput bold)${bit}$(tput sgr0) bit operating \
system is $(tput bold) `head -n 1 /etc/issue`\
$(tput sgr0), Nuclear info is $(tput setaf 1)\
${nuclear}$(tput sgr0)."

echo

echo "The CPU is$(tput setaf 4)`sed -n '5p' /proc/cpuinfo \
| cut -d ':' -f 2 | sed 's/[ ] */ /g'`$(tput sgr0)."

echo

echo "There are $(tput setaf 5)\
`cat /proc/cpuinfo | grep "physical id" | sort | uniq \
| wc -l`$(tput sgr0) physical cpu, \
each physical \
cpu has$(tput setaf 5)`sed -n '12p' /proc/cpuinfo | \
cut -d ':' -f 2`$(tput sgr0) cores,\
$(tput setaf 5)`sed -n '10p' /proc/cpuinfo | \
cut -d ':' -f 2`$(tput sgr0) threads."
```

```
echo

echo "There are $(tput setaf 5)\
`cat /proc/cpuinfo | grep "cpu cores" | wc -l`$(tput sgr0) logical cpu."

mem=`head -n 1 /proc/meminfo | cut -d ':' -f 2 | sed 's/^ *//g' | cut -d ' ' -f 1`
memInM=$(echo "$mem/1024/1024" | bc -l)

echo

echo "The memory of this server is $(tput setaf 5)${memInM}$(tput sgr0)G."

echo

echo "The disk information is :"

echo "`df -h`"
```

8 生信教程文章集锦

8.1 生信宝典

生信的作用越来越大，想学的人越来越多，不管是为了以后发展，还是为了解决眼下的问题。但生信学习不是一朝一夕就可以完成的事情，也许你可以很短时间学会一个交互式软件的操作，却不能看完程序教学视频后就直接写程序。也许你可以跟着一个测序分析流程完成操作，但不懂得背后的原理，不知道什么参数需要修改，结果可以出来，却把我不住对还是错。

学习生信从来就不是一个简单的事，需要做好持久战的心理准备。

在学习时，我们都希望由浅入深的逐步深入，不断地练习和实践，这就是为什么我们需要一本书，因为书很系统。但生信发展的历史短于计算机编程的历史，如果想要一门程序设计的入门数据，每种语言都可以找到几本。但想要一个囊括生信的书，就有些难了。本身生信跨领域，需要多学科的知识，而其内部又有不少分子，都囊括了太大，包括的少又有些隔靴搔痒的感觉。

我们当时都是零基础下自学 Linux, 自学 Python，自学 R，自学高通量测序；这些学习经历，之前都零星地记录在博客里。现在回头去看几年前自己记录的东西，觉得好简单，而当时却费了很大的力气。这些零星的随手记，当时也只是为了自己看，到现在确实只有自己能看得懂，不便惠及更多的人。

因此我们创建了生信宝典，希望从不同的角度传播知识。这个不同有三点含义，一是形式上的不同，摒弃之前主编们单人作战想写啥就写啥，而是有组织有计划的内容聚合，提供一系列的教程，由入门到提高。二是内容的不同，不去用网上现有教程的通用数据做例子，而是拿实际生物数据，讲述如何解释生信中普遍碰到的问题，讲述如何处理自己的数据。三是立足点不同。在写作时，我们回到了当年，在回忆中用整个阶段的学习去指导当初的那个小白，从那些会了的人觉得微不足道而不会的人又迈不过的坎入手，直击痛点。知识点的收录依据不是是否炫酷，是否难，而是是否必要。如果必要，再简单，也要提及；如果不必要，再炫酷，也暂不纳入。

通过大量的生信例子、关键的注释和浓缩的语句形成下面的一系列学习教程。每一篇内容都不多，可以当做小说阅读，也可以跟着去练，反复几遍，每读一次都会有不同的收获和体会。

8.1.1 系列教程

- 生物信息之程序
- 如何优雅的提问
- 生信宝典视频教程
- 好色之旅-画图三字经
- 转录组分析的正确姿势
- 生信的系列教程
- 生信的系列书籍
- 文章用图的修改和排版 (1)

- 文章用图的修改和排版 (2)
- 简单强大的在线绘图
- 简单强大的在线绘图-升级版
- 论文图表基本规范
- 学术图表的基本配色方法
- 英语写作常见错误总结和学习视频
- 教育部推出首批 490 门 “国家精品在线开放课程

8.1.2 NGS 分析工具评估

- 39 个转录组分析工具，120 种组合评估 (转录组分析工具哪家强-导读版)
- 39 个转录组分析工具，120 种组合评估 (转录组分析工具大比拼（完整翻译版）)
- 无参转录组分析工具评估和流程展示

8.1.3 宏基因组教程

- 微生物组入门必读 + 宏基因组实操课程
- 扩增子图表解读-理解文章思路
- 扩增子分析流程-把握分析细节
- 扩增子统计绘图-冲击高分文章
- 宏基因组分析教程
- 4500 元的微生物组培训资料

8.1.4 系列宣传

- 转录组分析的正确姿势
- 120 分的转录组考题，你能得多少
- 生物信息作图系列 R、Cytoscape 及图形排版和 Python 编程培训研讨班开课了
- 维密摔倒不可怕，关键时有人搀一把，坚持走下去
- 生物信息作图系列 - R、网络图及文章图形排版
- 易生信转录组培训总结和优惠分享
- 生物信息 9 天速成班 — 你也可以成为团队不可或缺的人
- Python 没有捷径，但可以加速，零基础九天你也可以会编程
- 小学生都学 Python 了，你还不知道怎么开始-资源帖
- 一个月学会 Python 的 Quora 指南和资料放送
- 扩增子分析基本流程和结果解读
- 微生物组——扩增子分析专题实战开课啦
- 如何入门生信 Linux
- 3 分和 30 分文章差距在哪里？

8.1.5 生信生物知识

- 生物研究中不可缺少的数字概念，多少，多大，多快

8.1.6 文献精读

- CRISPR-CAS9 发展历程小记
- 一场大病引起的诺贝尔 2017 年生理学奖角逐
- Science 搞反狗脑 - 人脑和狗脑一样？
- 一篇压根不存在的文献被引用 400 次 ?! 揭开” 幽灵文献” 的真面目
- 基于人工智能的文献检索，导师查找，更聪明
- GeenMedical：文献查询、筛选、引用排序、相似文献、全文下载、杂志分区、影响因子、结果导出、杂志评述、直接投稿，一站服务
- YANDEX 搜索，不翻墙稳定使用近谷歌搜索
- Nature 我的研究对后人毫无用途：21% 的学术论文自发布后从未被引用
- SCI-HUB 镜像, SSH 隧道访问学校内网
- 为了速成生物学，一位程序员探索了 “爆款” 基因背后的秘密

8.1.7 Linux

- Linux-总目录
- Linux-文件和目录
- Linux-文件操作
- Linux 文件内容操作
- Linux-环境变量和可执行属性
- Linux - 管道、标准输入输出
- Linux - 命令运行监测和软件安装
- Linux-常见错误和快捷操作
- Linux-文件列太多，很难识别想要的信息在哪列；别焦急，看这里。
- Linux-文件排序和 FASTA 文件操作
- Linux-应用 Docker 安装软件
- Linux 服务器数据定期同步和备份方式
- VIM 的强大文本处理方法
- Linux - Conda 软件安装方法
- 查看服务器配置信息
- Linux - SED 操作，awk 的姊妹篇
- Linux - 常用和不太常用的实用 awk 命令
- Bash 概论 - Linux 系列教程补充篇

8.1.8 CIRCOS 系列

- CIRCOS 圈图绘制 - circos 安装
- CIRCOS 圈图绘制 - 最简单绘图和解释
- CIRCOS 圈图绘制 - 染色体信息展示和调整
- CIRCOS 增加热图、点图、线图和区块属性

8.1.9 R 统计和作图

- 在 R 中赞扬下努力工作的你，奖励一份 CheatShet
- 别人的电子书，你的电子书，都在 bookdown
- R 语言 - 入门环境 Rstudio
- R 语言 - 热图绘制 (heatmap)
- R 语言 - 基础概念和矩阵操作
- R 语言 - 热图简化
- R 语言 - 热图美化
- R 语言 - 线图绘制
- R 语言 - 线图一步法
- R 语言 - 箱线图（小提琴图、抖动图、区域散点图）
- R 语言 - 箱线图一步法
- R 语言 - 火山图
- R 语言 - 富集分析泡泡图（文末有彩蛋）
- R 语言 - 散点图绘制
- 一文看懂 PCA 主成分分析
- 富集分析 DotPlot，可以服
- R 语言 - 韦恩图
- R 语言 - 柱状图
- R 语言 - 图形设置中英字体
- R 语言 - 非参数法生存分析
- 基因共表达聚类分析和可视化
- R 中 1010 个热图绘制方法
- 还在用 PCA 降维？快学学大牛最爱的 t-SNE 算法吧, 附 Python/R 代码
- 一个函数抓取代谢组学权威数据库 HMDB 的所有表格数据
- 文章用图的修改和排版
- network3D: 交互式桑基图
- network3D 交互式网络生成

8.1.10 扩增子三步曲

- 1 图表解读-理解文章思路

- 2 分析流程-把握分析细节

- 扩展 1：视频教程-夯实分析思路
- 扩展 2：QIIME2 教程-了解分析趋势
- 3 统计绘图-冲击高分文章

8.1.11 宏基因组分析专题

- 1 背景知识-Shell 入门与本地 blast 实战
- 2 数据质控 fastqc, Trimmomatic, MultiQC, khmer
- 3 组装拼接 MEGAHIT 和评估 quast
- 4 基因注释 Prokka
- 5 基于 Kmer 比较数据集 sourmash
- 6 不比对快速估计基因丰度 Salmon
- 7bwa 序列比对, samtools 查看, bedtools 丰度统计
- 8 分箱宏基因组 binning, MaxBin, MetaBin, VizBin
- 9 组装 assembly 和分箱 bin 结果可视化—Anvio
- 10 绘制圈图-Circos 安装与使用
- MetaPhlAn2 分析有参宏基因组

8.1.12 NGS 基础

- NGS 基础 - FASTQ 格式解释和质量评估
- NGS 基础 - 高通量测序原理
- NGS 基础 - 参考基因组和基因注释文件
- NGS 基础 - GTF/GFF 文件格式解读和转换
- 本地安装 UCSC 基因组浏览器
- 测序数据可视化 (一)
- 测序文章数据上传找哪里
- GO、GSEA 富集分析一网打进
- GSEA 富集分析 - 界面操作
- 去东方，最好用的在线 GO 富集分析工具
- 生信软件系列 - NCBI 使用

8.1.13 癌症数据库

- UCSC XENA - 集大成者 (TCGA, ICGC)
- ICGC 数据库使用
- TCGA 数据库在线使用

8.1.14 Python

- Python 学习 - 可视化变量赋值、循环、程序运行过程
- Python 极简教程（一）
- Python 教程（二）
- Python 教程（三）
- Python 教程（四）
- Python 教程（五）
- Python 教程（六）
- Pandas，让 Python 像 R 一样处理数据，但快
- Python 解析 psiBlast 输出的 JSON 文件结果
- 为啥我的 Python 这么慢 - 项查找 (二)
- 为啥我的 Python 这么慢 (一)
- Python 资源
- 关于 Python 中的 __main__ 和编程模板

8.1.15 NGS 软件

- Rfam 12.0+ 本地使用（最新版教程）
- 轻松绘制各种 Venn 图
- ETE 构建、绘制进化树
- psRobot：植物小 RNA 分析系统
- 生信软件系列 - NCBI 使用
- 去东方，最好用的在线 GO 富集分析工具

8.1.16 Cytoscape 网络图

- Cytoscape 教程 1
- Cytoscape 之操作界面介绍
- 新出炉的 Cytoscape 视频教程

8.1.17 分子对接

- 来一场蛋白和小分子的风花雪月
- 不是原配也可以-对接非原生配体
- 简单可视化-送你一双发现美的眼睛
- 你需要知道的那些前奏

8.1.18 生信宝典之傻瓜式

- 生信宝典之傻瓜式 (一) 如何提取指定位置的基因组序列
- 生信宝典之傻瓜式 (二) 如何快速查找指定基因的调控网络
- 生信宝典之傻瓜式 (三) 我的基因在哪里发光 - 如何查找基因在发表研究中的表达
- 生信宝典之傻瓜式 (四) 蛋白蛋白互作网络在线搜索
- 生信宝典之傻瓜式 (五) 文献挖掘查找指定基因调控网络

8.1.19 生信人写程序

- 生信人写程序 1. Perl 语言模板及配置
- 生信人写程序 2. Editplus 添加 Perl, Shell, R, markdown 模板和语法高亮

8.1.20 小技巧系列

- 参考文献中杂志名字格式混乱问题一次解决

8.1.21 招聘

- 易汉博欢迎您加入

8.2 宏基因组

http://mp.weixin.qq.com/s/5jQspEvH5_4Xmart22gjMA

宏基因组/微生物组是当今世界科研最热门的研究领域之一，为加强本领域的技术交流与传播，推动中国微生物组计划发展，中科院青年科研人员创立 “宏基因组” 公众号，目标为打造本领域纯干货技术及思想交流平台。

本公众号每日推送，工作日分享宏基因组领域科研思路、实验和分析技术，理论过硬实战强；周末科普和生活专栏，轻松读文看片涨姿势。目前经过近半年发展，分享过百篇原创文章，已有 14000+ 小伙伴在这里一起交流学习，感兴趣的赶快关注吧。

8.2.1 精选文章推荐

5000+

- 微生物组入门必读 + 宏基因组实操课程
- 你想要的生信知识全在这—生信宝典
- 生物信息 9 天速成班—成为团队不可或缺的人
- 3 分和 30 分文章差距在哪里？
- 肠道菌群在人体中的作用
- 看完此片我想把身上的细菌寄生虫供起来
- 岛国科普—生命大跃进
- 我们的未来在哪里？
- 论文图表基本规范
- DNA 提取发 Nature
- 实验 vs 数据分析，谁对结果影响大?

3000+

- 扩增子图表解读-理解文章思路
- 扩增子分析流程-把握分析细节
- 扩增子统计绘图-冲击高分文章
- 宏基因组分析教程
- 4500 元的微生物组培训资料
- Co-occurrence 网络图在 R 中的实现
- 学术图表的基本配色方法
- 一文读懂进化树
- 自学生信-biostar handbook
- 漱口水增加糖尿病，高血压风险

1000+

- 微生物组——扩增子分析专题培训开课啦 ！！！
- 最简单漂亮的免费在线生信绘图工具
- 小学生都学 Python 了，你还不知道怎么开始
- 五彩进化树与热图更配-ggtree 美颜进化树
- 扩增子分析还聚 OTU 就真 OUT 了
- 主流非聚类方法 dada2,deblur 和 unoise3 介绍与比较
- 16S 预测微生物群落功能 0 概述 1KO 通路 PICRUSt 2 元素循环 FAPROTAX 3 表型 bugbase 4KO 通路 Tax4Fun

- 一文读懂微生物组
- 2017 年发展简史和十大热文盘点

8.2.2 培训、会议、征稿、招聘

- 3 月 10-19 日，北京，微生物组——扩增子分析专题培训
- 5 月 11-13 日，北京，中国肠道大会

8.2.3 科研经验

- 如何优雅的提问
- 公众号搜索方法大全
- 科研团队成长三部曲：1 云笔记 2 云协作 3 公众号
- 文献阅读 1 热心肠 2SemanticScholar 3geenmedical
- 生信编程模板 Perl Shell R
- 生物信息之程序学习

- Endnote X8 云同步：有网随时读文献
- 论文 Figures，你不能不知道的秘密
- 转录组分析的正确姿势
- 整个世界都是你的已知条件

8.2.4 软件和数据库使用

- SILVAngs:16S/18S 在线分析1 2
- METAGENassist 帮你搞定宏基因分析的所有图形需求
- Windows 不用虚拟机或双系统，轻松实现 linux shell 环境：gitforwindows
- 一条命令轻松绘制 CNS 顶级配图-ggpubr
- ggbiplot-最好看的 PCA 图
- LDA 分析、作图及添加置信-ggord
- ggrepel-解决散点图样品标签重叠，方便筛选样品
- Alpha 多样性稀释曲线 rarefraction curve
- 微生物组间差异分析神器-STAMP

- 扩增子分析神器 USEARCH
- 微生物扩增子数据库大全

- antiSMASH：微生物次生代谢物基因簇预测

- 微生物网络构建：MENA, LSA, SparCC 和 CoNet

- Cytoscape: MCODE 增强包的网络模块化分析
- FUNGuild：真菌功能注释
- 在线 RaxML 构建系统发育树
- Genevestigator: 查找基因在发表研究中的表达

- psRobot：植物小 RNA 分析系统
- RepeatMasker：基因组重复序列注释

8.2.5 扩增子学习三步曲

8.2.5.1 1 图表解读-理解文章思路

8.2.5.2 2 分析流程-把握分析细节

- 扩展 1：视频教程-夯实分析思路
- 扩展 2：QIIME2 教程-了解分析趋势

8.2.5.3 3 统计绘图-冲击高分文章

8.2.6 宏基因组分析专题

- 1 背景知识-Shell 入门与本地 blast 实战
- 2 数据质控 fastqc, Trimmomatic, MultiQC, khmer
- 3 组装拼接 MEGAHIT 和评估 quast
- 4 基因注释 Prokka
- 5 基于 Kmer 比较数据集 sourmash
- 6 不比对快速估计基因丰度 Salmon
- 7bwa 序列比对, samtools 查看, bedtools 丰度统计
- 8 分箱宏基因组 binning, MaxBin, MetaBin, VizBin
- 9 组装 assembly 和分箱 bin 结果可视化—Anvi'o
- 10 绘制圈图-Circos 安装与使用
- MetaPhlAn2 分析有参宏基因组

8.2.7 R 统计绘图

- 视频教程：R 语言 recharts 包绘制交互式图形
- R 语言聚类分析–cluster, factoextra
- 堆叠柱状图各成分连线画法：突出展示组间物种丰度变化
- R 相关矩阵可视化包 ggcorrplot

8.2.8 实验设计与技术

- 微生物样本取样及微生物基因组 DNA 提取建议
- 样品生物学重复数据选择 1 必要性 2 需要多少重复？
- 样品命名 注意事项
- 扩增子引物选择 16S 结构 16S 单 V4 区是最佳选择?
- 海洋可培养微生物的鉴定与分类
- 怎么取粪便样品
- Rhizosphere、Rhizoplane 根际土如何取
- Nat. Biotechnol. 扩增子测序革命—用 16S 及 18S rRNA 全长进行微生物多样性研究

8.2.9 基础知识

- Microbiota, metagenome, microbiome 区别
- 16S 测序，不知道 OTU 你就 out 了！
- 计量宏基因组学数据分析的方法及进展
- 排序方法比较大全 PCA、PCoA、NMDS、CCA
- LEfSe 分析，你真的懂了么
- 宏基因组基础知识梳理
- 扩增子 SCI 套路1 微群落结构差异 2 组间差异分析 3 系统总结
- 环境因子关联分析——我应该选择 CCA 还是 RDA 分析？
- “P 值” 背后那些不可不知的事儿
- Adonis 和 ANOSIM 方法组间整体差异评估原理
- 轻松看懂机器学习十大常用算法
- 一文读懂 “随机森林” 在微生态中的应用
- 你想知道的 “ROC 曲线”
- 人体对微生物的管控
- 简单读懂微生物基因组的泛基因组学

8.2.10 必读综述

- Nature：宏基因组关联分析
- Nature：肠道菌群如何划分肠型
- Nature: 来自细菌的通告——群感效应简介
- Nature：呼吸道菌群—呼吸道健康的守门人
- Nature: 拥抱未知-解析土壤微生物组的复杂性
- Cell：代谢控制中的脑肠轴
- 研究微生物，只靠多组学根本不够

- 中国微生物组计划—农作物微生物组

- Annu Rev：植物微生物组—系统见解与展望

- Annual Reviews| 微生物组与人
- 微生物组学与植物病害微生物防治

- 组学重建真菌现有分类系统

- 微生物应用 | 农业废弃物资源化利用

- 宏基因组学入门1 初识 2 进一步 3 拼接

- 肠道微生物与人类密切相关的方方面面
- Nature: 测序技术的前世今生
- Nature Reviews：全新的益生元定义和范围
- 原核转录组非编码 RNA 研究

8.2.11 高分文章套路解读

- Nature: 培养组学—高通量细菌分离培养鉴定
- SR: 真菌培养组学同揭示人类肠道真菌群落结构
- Nature: 地球微生物组计划首发成果—揭示地球多尺度微生物多样性
- Nature：如何做一篇肠道菌群免疫的顶级文章
- Nat Biotech: 宏表观组—DNA 甲基化辅助宏基因组 binning
- Nature: 甘露糖苷选择性抑制致病性大肠杆菌

- Nature: 拟南芥根微生物组的结构和组成

- Nature: 地球上最古老的热液喷口发现早期生命迹象
- Nature Method: 宏基因组软件评估—人工重组宏基因组基准数据集

- Nature Genetics：微生物基因组如何适应植物？(news & views)
- NC：降低微生物群落复杂度突破组装难题

- NC：自体免疫水泡皮肤病中鉴定基因与微生物组互作

- GigaScience: 植物 MWAS 研究—谷子产量与微生物组关联分析

- Microbiome：微生物组研究中你必须注意的细节

- Microbiome：HiSeq 平台 16S 扩增子文库构建方法

- Microbiome: 简单套路发高分文章–杨树微生物组

- Microbiome：肠道菌群失衡促进高血压

- Microbiome：重新定义 “卫生” 概念

- ME：网络分析揭示微生物群落应对环境扰动的稳定性机制
- SR: 土壤细菌定量方法结合相对丰度分析揭示种群的真实变化

8.2.12 科普视频-寓教于乐

- BBC 人体奥秘之细胞的暗战
- BBC 人体奥秘 Inside.the.Human.Body
- NG 人体内旅行 Inside.the.Living.Body
- NG 子宫日记 Womb

- NHK: 再造人类生命的神奇细胞
- CCTV9 让尸体说话-法医密档
- 豆瓣 8.9，惹哭亿万中国人的纪录片-本草中华
- 2 分钟视频回顾植物学家钟扬的贡献
- 一顿 “寄生虫大餐”，或能治好干净引来的免疫病
- 只要 11 天，浓度 1000 倍的抗生素也无效——超级细菌
- 致命病毒为何疯狂袭击人类？都怪我们那群会飞的远房亲戚
- 土豆上的小霉菌引发百万人死亡和逃难，却造就全球 7 千万后裔
- 看完这些能控制大脑的寄生虫，你会怀疑人类！
- 梅毒狂想曲

8.2.13 友军文章汇总推荐

- 学习生信的系列教程——纯生信一作发 IF>20 的大神