

Linux 学习

Created By: xi_ling

目录

第 1 章 常用指令	4
1.1 文件管理	4
1.2 系统管理	6
1.3 用户管理	7
第 2 章 Windows Subsystem for Linux	8
第 3 章 软件安装	9
2.1 换源	9
2.2 初始化：更新与升级	10
2.3 使用 apt 命令安装包	10
2.4 安装下载的包	10
2.4.1 将安装包上传至服务器（或虚拟机）	10
2.4.2 解压缩并安装(.gz .bz .bz2)	11
2.4.3 安装.sh 文件	11
2.4.4 安装.deb 文件	12
2.5 设置环境变量	12
2.6 覆盖安装	12
第 4 章 环境部署	13
3.1 PYTHON 环境搭建（FOR PYSCF）	13
3.1.1 安装 PYTHON	13
3.1.2 安装 CONDA	13
3.1.3 conda 独立环境	15
3.1.4 通过 CONDA 安装 PYSCF	16
3.2 CPP 环境搭建（FOR HF-CPP）	16
3.2.1 GIT 项目-Hartree-Fock-in-CPP	16
3.2.2 GCC G++ 安装	16
3.2.3 IDE 选择	17
3.3 FORTRAN 环境搭建	17
3.3.1 GFORTRAN 安装	17

第 5 章 文件传输.....	18
4.1 Ubuntu 虚拟机文件传输(GUI).....	18
4.2 通过 SSH 连接(XSHELL)传输文件	18
4.2.1 lrzsz.....	18
4.2.2 python bypy 下载百度网盘文件	18
第 6 章 项目管理.....	20
5.1 GIT.....	20
5.1.1 连接到 GIT-HUB(WIN).....	20
5.1.2 连接到 GIT-HUB(LINUX).....	20
5.2 MAKE 编程基础.....	22
5.2.1 什么是 MAKEFILE	22
第 7 章 脚本编程.....	23
6.1 SHELL SCRIPT.....	23
6.1.1 什么是 SHELL SCRIPT.....	23
6.1.2 SHELL,SHELL SCRIPT 与 BASH.....	23
6.1.3 基于 BASH 的 SHELL SCRIPT 编程	23
6.2 初识 SHELL 编程	23
6.2.1 “hello world!”	23
6.2.2 shell 脚本运行	24
6.3 SHELL 脚本编程	25
6.3.1 变量.....	25
6.3.2 数组.....	28
6.3.3 变量传参.....	29
6.3.4 运算符.....	30
6.3.5 控制语句.....	32
6.4 函数.....	34
6.4.1 定义函数.....	34
6.4.2 函数参数.....	35
6.4.3 函数作用域.....	35
6.5 重定向.....	35
6.5.1 输入重定向.....	36
6.5.2 输出重定向.....	36
6.5.3 标准错误文件重定向.....	36
6.5.4 Here Document	36

6.5.5 禁止输出: /dev/null 文件	36
6.6 文件操作.....	36
6.6.1 文件操作.....	36
第 8 章 常见报错及处理.....	37

第1章 常用指令

#该部分当作一个库文件，请直接 CTRL+F 查找需要的指令

1.1 文件管理

gedit filename	//创建一个文件，GUI 界面下
touch new_file_name	//新建一个文件
mkdir directory_name	//创建一个目录
cd dirname	//进入目录 dirname
cd ..	//返回上层目录
cd ../	//返回上层目录
cd /	//跳转至根目录
cd ~	//跳转至当前用户的 home 目录
cd /home	//跳转至普通用户的 home 目录
ls	//查看当前目录下文件名
ls -ls	//查看当前目录下文件详细信息
ll	//查看文件详细信息，同 ls -ls
dir	//展示当前路径下所有文件名，不区分
rm	//删除文件或目录
mv	//移动文件或目录
chmod	//改变文件或文件夹的权限
./	//执行
exit	//退出程序（在 Xshell 中则断开连接）
wc FILE_NAME	//展示文档字符个数
wc -l FILE_NAME	//展示文档行数
echo "hello"	//在 shell 中打印"hello"
pwd	//显示当前目录
	//`.`： 当前目录
	//`..`： 代表上一级目录
	//`~`： 代表用户主目录
#文本编辑器	
vim	//最常用的 linux 阅读器，建议学习

```

less                                //一种有更多功能的文件阅读器

#查找文件
find FILE_NAME                     //查找文件
find -name file_name               //递归查找文件
find -name file_name*              //递归查找以 file_name 开头的全部文件

#删除文件
rmdir                              //删除空目录
rm                                  //删除文件
rm -r FILE_NAME                    //递归的删除目录及其所有文件
rm -rf FILE_NAME                   //递归的强制删除受保护文件
                                   //经典删库跑路代码，千万别在根目录下运行
                                   //若删除文件不存在，则会删除当前路径

#覆盖式删除文件（难以恢复）
shred FILE_NAME                    //从磁盘覆盖删除文件
shred -u FILE_NAME                 //立即删除文件

#文件操作
cat A B                            //将两个文件连接在一起[strcat]输出
tail -n FILE_NAME                  //展示文件的末尾 n 行
head -n FILE_NAME                  //展示文件的开头 n 行
grep KEYWORD FILENAME              //展示文件中含有 KEYWORD 的行

#解压缩
#不同字段间仅需一个空格，此处使用多个为对齐
tar -zxvf FILE_NAME.tar.gz         //解压缩.gz 文件
tar -zxvf FILE_NAME.tgz            //解压缩.tgz 文件
tar -jxvf FILE_NAME.tbz            //解压缩.tbz 文件
tar -xvf FILE_NAME.tar             //解压缩.tar 文件
unzip FILE_NAME.zip                 //解压缩.zip 文件
7z x FILE_NAME.7z                  //解压缩.7z 文件

```

#压缩文件

zip -r zipped.zip folder_to_zip

1.2 系统管理

#系统信息

htop //查看系统状态（可查看 CPU 以及内存）

uname //展示 UNIX 系统的名称

neofetch //展示本机详细信息

top //查看 CPU 使用率

lshw -class network lshw //查看网络使用情况

df -h //查看磁盘空间 1

sudo fidsk -l //查看磁盘空间 2

nproc //查看可并行核心数

lscpu //查看系统 CPU 信息

lsb_release -a //查看操作系统详细信息(ubuntu or centos)

uname -m //显示系统架构

file /bin/ls //显示系统架构详细信息

#程序信息

which python //展示 python 的安装路径

whatis python //展示程序的一些基本信息

gcc -v //展示 gcc 版本

cmake --version //展示 cmake 版本信息

#进程管理

ps //查看正在运行的进程，所有终端的进程

ps -l //查看正在运行的进程详细信息

jobs -l //查看当前终端运行的所有进程

kill PID //按照 pid 标识终止一个进程

ctrl + c //将一个正在执行的前台应用终止

#进程前后台调度

ctrl + z: 将一个正在执行的前台应用放到后台，并且处于暂停状态。

&: command + &表示将此程序在后台执行

fg + %jobnum: 将后台中的应用调至前台执行
bg: 将一个在后台暂停的命令, 在后台继续执行

#关机重启
shutdown 20:00 //在 20: 00 关闭电脑
shutdown -h now //立即关机
shutdown -s //立即关机
shutdown -c //终止一个关机计划
reboot //重启

1.3 用户管理

Ubuntu GUI 中:
Ctrl+alt+T //在当前路径下打开一个终端
Ctrl+shift+N //新建一个文件

#用户命令
history //展示最近使用的命令
passwd //修改当前用户密码
whoami //展示用户名

#用户操作
#用户信息存储在/etc/passwd 文件中
su USER_NAME //切换至目标用户下, 需要输入密码
su root //特殊的, 切换至 root 用户
#一下指令前均需要加入 sudo: sudo 指使用 root 权限执行命令
adduser USER_NAME //添加用户, 在/home 目录下会创建同名路径
userdel USER_NAME //删除用户
userdel -r USER_NAME //删除用户, 同时删除用户文件
passwd USER_NAME //修改用户密码

groups USER_NAME //查看用户组

#用户组操作
#用户组信息储存在/etc/group 文件中
#关于文件、用户、用户组关系, 见第 0 章 LINUX 系统概述。

批注 [cj1]: 尚未完成, 第 0 章: 用户权限

```
addgroup GROUP_NAME //添加用户组
group GROUP_NAME //查看组内用户名
sudo usermod -aG GROUP_NAME USER_NAME
//将用户添加到用户组
// G: 指定组名，可加入多个用户组，会删除
// a:表示追加，不会从其他用户组删除
sudo usermod -G GROUP_NAME USER_NAME
//将用户从用户组删除
```

第2章 Windows Subsystem for Linux

2.1 何为 WSL
WSL 指 Windows Subsystem for Linux，即 Linux 版的 window 子系统。

第3章 软件安装

2.1 换源

Ubuntu 下载文件时从选中的源下载文件，想要获得更快的下载文件速度，我们必须更换合适我们的文件下载源。

- a) 保存原来的源文件
`sudo cp /etc/apt/sources.list /etc/apt/sources.list_backup`
- b) `sudo vim /etc/apt/sources.list`
- c) 阿里云的源 copy 进去即可（熟练 vim 使用）
`deb http://mirrors.aliyun.com/ubuntu/ trusty main restricted universe multiverse`
`deb http://mirrors.aliyun.com/ubuntu/ trusty-security main restricted universe multiverse`
`deb http://mirrors.aliyun.com/ubuntu/ trusty-updates main restricted universe multiverse`
`deb http://mirrors.aliyun.com/ubuntu/ trusty-proposed main restricted universe multiverse`
`deb http://mirrors.aliyun.com/ubuntu/ trusty-backports main restricted universe multiverse`
`deb-src http://mirrors.aliyun.com/ubuntu/ trusty main restricted universe multiverse`
`deb-src http://mirrors.aliyun.com/ubuntu/ trusty-security main restricted universe multiverse`
`deb-src http://mirrors.aliyun.com/ubuntu/ trusty-updates main restricted universe multiverse`
`deb-src http://mirrors.aliyun.com/ubuntu/ trusty-proposed main restricted universe multiverse`
`deb-src http://mirrors.aliyun.com/ubuntu/ trusty-backports main restricted universe multiverse`
- d) 中科大源：(gcc-9)
`deb https://mirrors.ustc.edu.cn/ubuntu/ focal main restricted universe multiverse`
`deb-src https://mirrors.ustc.edu.cn/ubuntu/ focal main restricted universe multiverse`
`deb https://mirrors.ustc.edu.cn/ubuntu/ focal-updates main restricted universe multiverse`
`deb-src https://mirrors.ustc.edu.cn/ubuntu/ focal-updates main restricted universe multiverse`
`deb https://mirrors.ustc.edu.cn/ubuntu/ focal-backports main restricted universe multiverse`

<pre>deb-src https://mirrors.ustc.edu.cn/ubuntu/ focal-backports main restricted universe multiverse deb https://mirrors.ustc.edu.cn/ubuntu/ focal-security main restricted universe multiverse deb-src https://mirrors.ustc.edu.cn/ubuntu/ focal-security main restricted universe multiverse deb https://mirrors.ustc.edu.cn/ubuntu/ focal-proposed main restricted universe multiverse deb-src https://mirrors.ustc.edu.cn/ubuntu/ focal-proposed main restricted universe multiverse</pre> <p>e) 测试</p> <pre>sudo apt update</pre> <p>#图形界面(GUI): software and updates->Download from;</p>
2.2 初始化：更新与升级
<pre>sudo apt upgrade sudo apt update</pre> <p>#更新并升级所有已安装的包</p>
2.3 使用 apt 命令安装包
<pre>#安装文件名为 PACKAGE_NAME 的包 sudo apt install PACKAGE_NAME</pre> <pre>#查找文件名为 PACKAGE_NAME 的包 sudo apt search PACKAGE_NAME</pre> <pre>#参数-y 安装过程中所有询问都选择 yes sudo apt install -y PACKAGE_NAME</pre>
2.4 安装下载的包
2.4.1 将安装包上传至服务器（或虚拟机）
<pre>#在/home/username 下新建一个目录 packages mkdir packages</pre>

<pre>#进入该目录 cd pcackages #将安装包下载至该目录下(.gz .bz .sh .tar) 利用 lrzsz 下载至服务器 利用 bypy 下载至服务器</pre>
2.4.2 解压缩并安装(.gz .bz .bz2)
<pre>#解压缩文件 #根据压缩包类型解压缩文件 (* 代表压缩包名称) tar -zxvf ****.tar.gz tar -jxvf ****.tar.bz (或 bz2) unzip ****.zip #其他解压缩命令见“1.1 文件管理”末尾 #用 CD 命令进入解压缩后的目录 #输入编译文件命令 ./configure (有的压缩包已经编译过, 这一步可以省去) #命令 make 安装文件命令 make install #安装完毕 #若需要 build, 则在解压缩后目录内 mkdir build cmake .. make make install</pre>
2.4.3 安装.sh 文件
<pre>#有 root 权限 1. 获取 root 权限 su root #输入 root 用户密码</pre>

<p>2. 找到.sh 文件</p> <p>3. 赋予权限</p> <pre>chmod +x FILE_NAME</pre> <p>4. 执行文件</p> <pre>./FILE_NAME</pre> <p>#无 root 权限</p> <p>1. <code>sudo chmod +x FILE_NAME</code> 输入当前用户用户密码</p> <p>2. <code>./FILE_NAME</code></p>
2.4.4 安装 .deb 文件
<pre>sudo apt install dpkg</pre> <pre>sudo dpkg -i FILE_NAME</pre>
2.5 设置环境变量
<p>环境变量：将程序所在路径放在系统或用户环境变量中，即可在任意位置下执行程序。具体定义及作用见 google。</p> <p>系统环境变量位置： /etc/profile</p> <p>用户环境变量： ~/.bashrc</p> <p>修改环境变量时，使用 <code>vim</code> 修改相应文件，在文件末尾加上环境变量位置，格式一般为：</p> <pre>export PATH=/usr/local/path:\$PATH</pre> <p>或</p> <pre>export LIBRARY_PATH=\$LIBRARY_PATH:/usr/local/openssl/lib</pre>
2.6 覆盖安装
<p>#在安装命令后添加</p> <pre>-u</pre>

第4章 环境部署

3.1 PYTHON 环境搭建（FOR PYSCF）

3.1.1 安装 PYTHON

- 1) 使用 apt 安装 python

```
sudo apt install python3
sudo apt install python3-pip
```
- 2) 设置 python 版本（将默认的 python 版本换为 python3）
#查看当前有多少 python 版本

```
sudo update-alternatives --config python
```


#设置 python3 为默认版本

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3 150
```


#设置 python2 为默认版本

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python2 100
```
- 3) 安装 python 常用库

```
pip install numpy
pip install scipy //科学计算
pip install opencv
pip install matplotlib //作图
pip install pandas
pip install sklearn //机器学习库
```

3.1.2 安装 CONDA

- 1) 下载 conda 镜像
#NJU 镜像网站
<http://mirrors.nju.edu.cn/anaconda/>
选择 miniconda 最新版本下载至本机
- 2) 使用 rz 或 bypy 上传至虚拟机
见“2.4.1 将安装包上传至云服务器（或虚拟机）”
- 3) 安装 [Miniconda3-py39_4.12.0-Linux-x86_64.sh](#) 文件（若为 x86 架构）
见“[2.4.3 直接安装\(.sh\)](#)”

在选择安装地址时

/*

Miniconda3 will now be installed into this location:

/root/miniconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

[/root/miniconda3] >>>

*/

在>>>后填写: /bin/miniconda

PS: 准确填写以防安装位置错误

4) 激活 conda

cd /bin/miniconda/bin

source activate (若提示无权限, chmod +x FILE_NAME 给权限)

5) 添加镜像 (channels 后无回车)

conda config --add channels

https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/free/

conda config --add channels

https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/main/

conda config --add channels

https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/

conda config --add channels

https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/bioconda/

[关于国内 anaconda 镜像站点看这一篇就够啦 - 知乎 \(zhihu.com\)](#)

6) 查看已经添加的镜像

conda config --set show_channel_urls yes #显示已经安装的频道

conda config --get channels #查看安装的频道

- 7) 编辑 conda 配置文件（添加环境变量）

```
vim ~/.bashrc
```

在文件最后加入：`export PATH="/bin/miniconda/bin:$PATH"`

```
source ~/.bashrc
```

PS:

3.1.3 conda 独立环境

5. 创建独立环境

不能将需要用的软件或者其他要用的东西装在base环境中

创建一些独立的环境会比较方便我们做项目管理，不会有冗余，**一个独立的小环境就像一间单独的小房间，彼此独立不会相互影响，这样就会可以方便我们管理不同依赖包的软件**

建议：base环境中不要安装任何的包，保持干净，除非你知道这个包会对原生的环境造成什么样的影响

```
conda create -n rnaseq #创建conda小环境 -n 用来指定环境的名称
conda create -n database python=3.7.3 #指定环境中需要带的python的版本

conda activate rnaseq # 启动小环境

conda deactivate #退出小环境

conda env list / conda info --env #查看共有多少个小环境

conda remove -n python --all #删除conda小环境
```

- 1) 创建名为 123 的独立环境
`conda create -name 123`
- 2) 指定环境中带的 python 版本
`conda create -name database python = 3.11`
- 3) 启动独立环境（每次开机时都需要启动）
`conda activate 123`
- 4) 退出独立环境
`conda deactivate`
- 5) 查看有多少个小环境
`Conda env list / conda info --env`

6) 删除 conda 小环境 <pre>conda remove -n python --all</pre>
3.1.4 通过 CONDA 安装 PYSCHF
<pre>#conda 安装 pyscf conda install -c pyscf pyscf #如果 import pyscf 失败, 则 cp /bin/miniconda/lib/python3.7/site-packages/pyscf /bin/miniconda/lib/python3.9/site-packages/pyscf -r</pre>
3.2 CPP 环境搭建 (FOR HF-CPP)
3.2.1 GIT 项目-Hartree-Fock-in-CPP
<p>由睿叔叔编码维护的开源 CPP 项目 Hartee-Fock, 具有优秀的代码架构、极佳的命名规范, 适合用于 HF-CPP 入门。</p> <p>本作仅用于搭建 CPP 环境, 具体项目所需库文件下载见在上方 GIT-HUB。</p>
3.2.2 GCC G++安装
<pre>#安装 GCC sudo apt install gcc #安装 G++ sudo apt install g++ #版本检验 gcc -v g++ -v #更换 gcc、g++版本 sudo apt install gcc-11 g++-11 sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-11 110 --slave /usr/bin/g++ g++ /usr/bin/g++-11</pre>

#安装基本环境 sudo apt install build-essential
3.2.3 IDE 选择
<p>IDE(Integrated Development Environment)，集成开发环境，是用于提供程序开发环境的应用程序，如 windows 下的 DEV C++、VISUAL STUDIO、VISUAL STUDIO CODE……</p> <p>Ubuntu GUI（图形界面）下推荐使用 VISUAL STUDIO CODE 作为 IDE，由于笔者通过 XSHELL 连接服务器，故使用 VIM+GCC。</p>
3.3 FORTRAN 环境搭建
3.3.1 GFORTTRAN 安装
Sudo apt install gfortran

第5章 文件传输

4.1 Ubuntu 虚拟机文件传输 (GUI)

直接将需要上传的文件复制(copy)，在 VMware workstation 中粘贴(paste)即可。

4.2 通过 SSH 连接(XSHELL)传输文件

4.2.1 lrzsz

```
#安装 lrzsz
sudo apt install lrzsz

#在目标路径处
##上传文件至服务器
rz

##下载文件
sz FILE_NAME
```

4.2.2 python bypy 下载百度网盘文件

bypy 是一种利用百度网盘官方 API 下载文件的 python 包。
安装完 bypy 包并绑定百度网盘账号后，百度网盘中会出现名为“我的应用数据->bypy”的文件夹。将待下载文件拷贝到该文件夹中，即可通过 bypy 命令下载到服务器。

```
#安装：
pip install bypy

#第一次使用：
bypy info

指定路径上传：
bypy syncup xxx
# 或者
bypy upload xxx
不加 xxx 时，则上传当前目录所有文件
```

下载单个文件 xxx

`bypy downfile xxx`

指定路径下载

`bypy syncdown`

或者

`bypy downdir /`

`bypy downdir /aaa/bbb` 可以选择文件夹，下载百度网盘/我的应用数据
/bypy/aaa/bbb 文件夹的内容到本地

下载速度可观！

PS：在百度网盘中复制.gz 文件名时需要 `ctrl+a` 两次

第6章 项目管理

5.1 GIT

5.1.1 连接到 GIT-HUB (WIN)

- 1) 浏览器访问 “<https://raw.githubusercontent.com/helloegon/hosts/master/hosts>”，获取最新映射，复制下来。
- 2) 打开 hosts 文件，Windows 文件夹地址栏输入：
C:\Windows\System32\drivers\etc\hosts，回车。
- 3) 粘贴最新映射，保存，`ctrl+R`，输入`cmd`，输入`ipconfig /flushdns`，刷新 DNS。
- 4) 尝试访问[github 官网](<https://github.com/>)，访问不了换网络，换成手机热点等。

5.1.2 连接到 GIT-HUB (LINUX)

- 1) 更换 DNS
apt install resolvconf
vi /etc/resolvconf/resolv.conf.d/base
#格式：
nameserver 8.8.8.8
#PS：将下载的映射版 HOSTS 更换格式

resolvconf -u
#查看是否成功
vi /etc/resolv.conf
- 2) 安装 git
sudo apt install git
- 3) 常用命令
#查看版本号
git -v

#设置用户名和邮箱
git config --global user.name "名称"
git config --global user.email "邮箱"

#查看设置的用户名和邮箱

`git config --global --list`

#查看帮助选项

`git --help`

#克隆一个项目

`git clone`

1) #HTTPS:

`git clone --recursive https://github.com/Walter-Feng/Hartree-Fock-in-CPP.git`

#其中参数 `--recursive` 指拷贝库中所有文件;

#文件保存在当前路径下

2) SSH (推荐使用)

#在终端中使用以下命令生成密钥对

`ssh-keygen -t rsa -b 4096 -C your_email@example.com`

#打开生成的公钥文件, 并将内容复制到剪贴板中

`cat ~/.ssh/id_rsa.pub`

#在 GitHub 上打开你的账户设置

#在左侧菜单中点击 "SSH and GPG keys"

#点击 "New SSH key" 创建一个新的 SSH 密钥

#在 "Title" 字段中为密钥提供一个描述性的名称

#在 "Key" 字段中粘贴之前复制的公钥内容

#点击 "Add SSH key" 保存并添加密钥

#使用 SSH URL 方式来下载文件

`git clone git@github.com:username/repository.git`

#其中`username` 替换为你的 GitHub 用户名, `repository` 替换为你要下载的仓库名称

5.2 MAKE 编程基础

本节参考书：《LINUX C：从入门到精通》

暂时没有写过 MAKEFILE，等 HF 写到了再来补充该部分。

批注 [cj2]: Weiwanc

5.2.1 什么是 MAKEFILE

1) Makefile 关系到整个工程的编译规则，简单来说：“自动化编译”；

第7章 脚本编程

6.1 SHELL SCRIPT

6.1.1 什么是 SHELL SCRIPT

shell 脚本语言是利用 shell 的功能所写的程序，这个程序使用纯文本文件，将一些 shell 的语法与指令写在里面，然后用正则表示法，管道命令以及数据流重导向等功能，以实现目标功能。

其最简单的功能就是将许多指令汇集整理在一起，让使用者很容易地就能够一个操作执行多个命令。另外，shell script 还提供了数组，循环，条件以及逻辑判断等重要功能，让使用者可以直接以 shell 来写程序，而不必使用类似 C 程序语言等传统程序编写的语法。

6.1.2 SHELL, SHELL SCRIPT 与 BASH

#此部分对 shell 与 bash 的定义或许不完全准确。

Shell 是一个命令行解释器，它的作用就是遵循一定的语法将输入的命令加以解释并传给系统。它为用户提供了一个向 Linux 发送请求以便运行程序的接口系统级程序，用户可以用 Shell 来启动、挂起、停止甚至是编写一些程序。

SHELL 脚本是一种将 SHELL 命令组织起来完成指定功能的执行文件。

BASH 是 SHELL 的一种，其为绝大多数 LINUX 发行版默认的 SHELL，其存放位置为：/bin/bash

6.1.3 基于 BASH 的 SHELL SCRIPT 编程

本章 SHELL 脚本编程内容全部基于 BASH

6.2 初识 SHELL 编程

6.2.1 “hello world!”

#下面演示创建一个简单的 shell 脚本并执行

#创建 shell 使用 vim 文本编辑器

1) 创建文档 hello_world.sh

#打开一个 Terminal/（或使用 XSHELL 连接服务器）

#在命令行中输入 vim hello_world.sh

#按下 i 进入 insert 模式，输入以下文本

```
#!/bin/bash
echo "HELLO, WORLD!"
~
```

#按 Esc 退出 insert 模式，输出“:wq”文件并退出。

2) 运行 hello_world.sh

#输入 sh hello_world.sh

```
root@iZgc7g57ntfaz1j313mw24Z:~# mkdir test
root@iZgc7g57ntfaz1j313mw24Z:~# cd test
root@iZgc7g57ntfaz1j313mw24Z:~/test# vi hello_world.sh
root@iZgc7g57ntfaz1j313mw24Z:~/test# sh hello_world.sh
HELLO, WORLD!
root@iZgc7g57ntfaz1j313mw24Z:~/test#
```

屏幕输出“HELLO, WORLD”

解释:

#!/bin/bash	//指定 shell 使用 bash 解释命令
	//单独使用代表“注释”
echo “HELLO WORLD!”	//表示在 shell 中输出 HELLO, WORLD!

6.2.2 shell 脚本运行

#x 权限、bash 和 source 执行脚本的一个区别:

使用 x 权限和 bash 执行脚本，bash 解释器本身会再生成一个 bash 子进程去执行脚本，在脚本执行完之后这个子进程就会消亡;

source 执行脚本时，调用原本的 bash 去执行脚本，执行完脚本后进程不会消亡。

【这种区别可能会造成局部变量的影响，因为全局变量基于当前】，工作中一般都会使用 x 权限的形式执行脚本。

假设文件名为 test.sh

#脚本文件无执行权限

##手动开启指定解释器:

```
root@iZgc7g57ntfaz1j313mw24Z:~/test# sh hello_world.sh
HELLO, WORLD!
```

##直接在当前环境中运行脚本

```
root@iZgc7g57ntfaz1j313mw24Z:~/test# . hello_world.sh
HELLO, WORLD!
root@iZgc7g57ntfaz1j313mw24Z:~/test# source hello_world.sh
HELLO, WORLD!
```

##直接在当前环境下运行脚本

#脚本文件有权限

##赋予脚本文件权限

```
root@iZgc7g57ntfzlj3l3mw24Z:~/test# chmod +x hello_world.sh
root@iZgc7g57ntfzlj3l3mw24Z:~/test# ls
hello_world.sh
```

###本来文件名显示为白色

##绝对路径执行脚本文件

###pwd 显示当前路径

###使用 pwd 得到当前路径后在当前路径下加上文件名即可执行

```
root@iZgc7g57ntfzlj3l3mw24Z:~/test# /root/test/hello_world.sh
HELLO, WORLD!
```

##相对路径执行脚本文件

```
root@iZgc7g57ntfzlj3l3mw24Z:~/test# ./hello_world.sh
HELLO, WORLD!
```

###其中，“.”代表当前所在位置

6.3 SHELL 脚本编程

6.3.1 变量

a. 变量类型

- 1) 局部变量: local, 仅在当前 shell 实例中有效, 其他 shell 启动的程序不能访问。函数中局部变量只能在函数内部访问。
- 2) 环境变量: 所有的程序, 包括 shell 启动的程序, 都能访问的变量。
- 3) shell 变量: shell 程序设置的特殊变量

#

b. 变量操作

#”=”两边不能有空格

1) 创建普通变量

```
name="test"
```

2) 创建局部变量

```
local name="test"
```

#若在函数中创建, 则只能在函数体中使用

3) 创建只读变量

```
name="hello" -> readonly name
```

4) 使用变量

```
echo $name
```

5) 删除变量

`unset name`

c. 字符串变量

1) 字符串变量创建

🚦 单引号创建：字符串常量

`var='test'`

#字符串创建后具有常量特性，且，单引号中不能出现单独的单引号与转义符号

🚦 双引号创建：字符串变量

`var="hello! I'am $name"`

#字符串变量、可出现转义符号（\$name、\n、\t）

2) 拼接字符串

🚦 常量拼接

```
root@iZgc7g57ntfaz1j313mw24Z:~/test# strx='0''1'
root@iZgc7g57ntfaz1j313mw24Z:~/test# echo $strx
01
```

🚦 变量拼接

```
root@iZgc7g57ntfaz1j313mw24Z:~/test# st1="abc"
root@iZgc7g57ntfaz1j313mw24Z:~/test# st2="123"
root@iZgc7g57ntfaz1j313mw24Z:~/test# strx=${st1}${st2}
root@iZgc7g57ntfaz1j313mw24Z:~/test# echo $strx
abc123
```

```
root@iZgc7g57ntfaz1j313mw24Z:~/test# st1="abc"
root@iZgc7g57ntfaz1j313mw24Z:~/test# st2="123"
root@iZgc7g57ntfaz1j313mw24Z:~/test# strx=${st1}"end"
root@iZgc7g57ntfaz1j313mw24Z:~/test# echo $strx
abcend
```

```
root@iZgc7g57ntfaz1j313mw24Z:~/test# st1="abc"
root@iZgc7g57ntfaz1j313mw24Z:~/test# st2="123"
root@iZgc7g57ntfaz1j313mw24Z:~/test# strx="${st1}${st2}"
root@iZgc7g57ntfaz1j313mw24Z:~/test# echo $strx
abc123
```

🚦 命令拼接

```
root@iZgc7g57ntfaz1j313mw24Z:~/test# str='cat'" hello_world.sh"
root@iZgc7g57ntfaz1j313mw24Z:~/test# $str
#!/bin/bash
echo "HELLO,WORLD!"
```

#"\$str"相当于执行"cat hello_world.sh"命令

3) 获取字符串长度

🚦 wc

wc -L : 获取当前行的长度

wc -l : 获取当前字符串内容的行数

: 还可以用于获取文件行数

```
root@iZgc7g57ntfaz1j313mw24Z:~/test# echo "abc" | wc -L
3
```

关于管道命令, 见第 0 章 LINUX 系统概述

批注 [cj3]: 尚未完成, 第 0 章 管道命令

🚦 expr length

```
root@iZgc7g57ntfaz1j313mw24Z:~/test# expr length "abc"
3
root@iZgc7g57ntfaz1j313mw24Z:~/test# strx="abcd"
root@iZgc7g57ntfaz1j313mw24Z:~/test# expr length $strx
4
```

🚦 awk + length 获取字符串长度

echo "Type" | awk '{print length(\$0)}'

🚦 echo\${#str_name}方式

最常用

```
root@iZgc7g57ntfaz1j313mw24Z:~/test# strx="xyzt"
root@iZgc7g57ntfaz1j313mw24Z:~/test# echo ${#strx}
4
```

4) 提取子字符串(substr)

🚦 echo + \${variable +操作+string}

variable: 变量名


\${variable##*string} 从左向右截取最后一个 string 后的字符串

\${variable#*string} 从左向右截取第一个 string 后的字符串

\${variable%%string*} 从右向左截取最后一个 string 后的字符串

`${variable%string*}` 从右向左截取第一个 string 后的字符串

```
root@iZgc7g57ntfaz1j313mw24Z:~# strx="formyfortune"
root@iZgc7g57ntfaz1j313mw24Z:~# echo ${strx##*fo}
rtune
root@iZgc7g57ntfaz1j313mw24Z:~# echo ${strx#*fo}
rmyfortune
```

 `${variable:n1:n2}`

截取变量 variable 从 n1 到 n2 之间的字符串，下表从 0 开始。

```
root@iZgc7g57ntfaz1j313mw24Z:~# strx="formyfortune"
root@iZgc7g57ntfaz1j313mw24Z:~# echo ${strx:0:4}
form
```

6.3.2 数组

Bash 中只支持一维数组！

a. 定义与引用

 定义数组

#空格分隔


数组名=(元素 1 元素 2 元素 3 ... 元素 n)

 赋值

数组名[下标]=值


 定义时赋初值

数组名=[下标 1]=值 1 [下标 2]=值 2 ... [下标 n]=值 n)

 引用数组对应下标的元素：

`${数组名[下标]}`

b. 遍历数组

 For（或 while）遍历

#以下代码只有在+x 权限下才能正常使用

`#!/bin/bash`

`a=(1 2 3 4 5)`

`for((i=0; i<10; i++))`

`do`

```
echo "a[$i]=${a[$i]}"
done
```

🚦 \${a[*]}或\${a[@]}遍历数组

```
a=(1 2 3 4 5 6)
echo ${a[*]}
echo ${a[@]}
```

c. 获取数组长度

```
echo "a len: ${#a[*]}"
#输出数组长度
```

d. 数组合并

```
a=("hello")
b=("world")
c=(${a[*]} ${b[*]})
```

e. 删除操作

```
#删除数组中元素 a[0]
unset a[0]
#删除数组 a
unset a
```

6.3.3 变量传参

变量	含义
\$0	代表执行的文件名
\$1	代表传入的第 1 个参数
\$n	代表传入的第 n 个参数
\$#	参数个数
\$*	以一个 单字符串 显示所有向脚本传递的参数。
\$@	与 \$* 相同，但是使用时 加引号 ，并在引号中返回 每个参数
\$\$	脚本运行的当前进程号
\$_	后台运行的最后一个进程的 ID
\$?	显示最后命令的退出状态。 0 表示没有错误，其他任何值表明有错误。

常见参数传递见指令。

6.3.4 运算符

a. 算数运算符

运算	shell 中格式
加法	<code>expr \$a + \$b</code>
减法	<code>expr \$a - \$b</code>
乘法	<code>expr \$a * \$b</code>
除法	<code>expr \$b / \$a</code>
取余	<code>expr \$b % \$a</code>
赋值	<code>a=\$b</code>
相等	<code>[\$a == \$b]</code>
不相等	<code>[\$a != \$b]</code>

#使用 `expr` 时需要将命令夹在 ``（反单引号，在键盘上"1"左边）中；

#使用 `[]` 时需要注意条件表达式与 `[]` 之间有空格；

b. 关系运算符

#只支持数字，或字符串中的数值

运算	shell 中的实现	主要符号
检测两个数是否相等	<code>[\$a -eq \$b]</code>	<code>-eq</code>
检测两个数是否不相等	<code>[\$a -ne \$b]</code>	<code>-ne</code>
检测左边的数是否大于右边的	<code>[\$a -gt \$b]</code>	<code>-gt</code>
检测左边的数是否小于右边的	<code>[\$a -lt \$b]</code>	<code>-lt</code>
检测左边的数是否大于等于右边的	<code>[\$a -ge \$b]</code>	<code>-ge</code>
检测左边的数是否小于等于右边的	<code>[\$a -le \$b]</code>	<code>-le</code>

c. 布尔运算符

运算	shell 中的实现	主要符号
非运算	<code>[! false]</code>	<code>!</code>
或运算	<code>[\$a -lt 20 -o \$b -gt 100]</code>	<code>-o</code>
与运算	<code>[\$a -lt 20 -a \$b -gt 100]</code>	<code>-a</code>

d. 逻辑运算符

运算	shell 中的实现	主要符号
逻辑的 AND	<code>[[\$a -lt 100 && \$b -gt 100]]</code>	<code>&&</code>
逻辑的 OR	<code>[[\$a -lt 100 \$b -gt 100]]</code>	<code> </code>

#布尔运算符与逻辑运算符区别：

#1. 布尔：`[]` 逻辑：`[[]]`

#2. 逻辑运算符有短路功能:

#AND 运算中第一个表达式为 false 时不执行第二个表达式; OR 运算中第一个表达式为 true 时不执行第二个表达式。

e. 字符串运算符

运算	shell 中的实现	主要符号
检测两个字符串是否相等	[\$a = \$b]	=
检测两个字符串是否不相等	[\$a != \$b]	!=
检测字符串长度是否为 0	[-z \$a]	-z
检测字符串长度是否不为 0	[-n "\$a"]	-n
检测字符串是否为空	[\$a]	\$

f. 文件测试运算符

运算	shell 中的实现	主要符号
检测文件是否是块设备文件	[-b \$file]	-b file
检测文件是否是字符设备文件	[-c \$file]	-c file
检测文件是否是目录	[-d \$file]	-d file
检测文件是否是普通文件 (既不是目录, 也不是设备文件)	[-f \$file] 返回 true	-f file
检测文件是否设置了 SGID 位	[-g \$file]	-g file
检测文件是否设置了粘着位(Sticky Bit)	[-k \$file]	-k file
检测文件是否是有名管道	[-p \$file]	-p file
检测文件是否设置了 SUID 位	[-u \$file]	-u file
检测文件是否可读	[-r \$file]	-r file
检测文件是否可写	[-w \$file]	-w file
检测文件是否可执行	[-x \$file]	-x file
检测文件是否为空 (文件大小是否大于 0)	[-s \$file]	-s file
检测文件 (包括目录) 是否存在	[-e \$file]	-e file

g. 运算指令

1. (())

使用双圆括弧计算其中的内容, 如((var=a+b)), 该指令经常在 if/while 等条件判断中需要计算时使用。

#支持一些 C 语言符号, 如 for 循环

2.let

let var=a+b

3.expr

#常用的计算指令，使用时需要在外部增反引号

var=`expr a+b`

4.\$[]

直接使用这种方式计算中括弧中的内容，如 echo \${3+7}

6.3.5 控制语句

#结构上同 python，学习形式即可

a. If 语句结构

#简单 if 结构

if condition

then

commandN

fi

#if-else 结构

if condition

then

command1

else

command2

fi

#if-else if-else 结构

if condition1

then

command1

elif condition2

then

command2

else

command3

fi

b. for 循环结构

```
for var in item1 item2 ... itemN
```

```
do
```

```
    commandN
```

```
done
```

c. while 循环结构

```
while condition
```

```
do
```

```
    command
```

```
done
```

d. 无限循环

#1.

```
for (( ; ; ))
```

#2.

```
while :
```

```
do
```

```
    command
```

```
done
```

#3.

```
while true
```

```
do
```

```
    command
```

```
done
```

e. Until 循环

```
until condition
```

```
do
```

```
    command
```

```
done
```

f. 跳出循环

continue && break

#多分支结构

g. case-esac

#其中;;代表 break

#模式需要以)结尾

##代表 default

case VALUE in

模式 1)

command

;;

模式 2)

command

;;

*)

command

esac

h. select-in 语句

#适合用作菜单!!

Example:

echo "What is your favourite OS?"

select var in "Linux" "Gnu Hurd" "Free BSD" "Other"; do

break;

done

echo "You have selected \$var"

6.4 函数

6.4.1 定义函数

[function] funname [()]

<pre>{ command [return int;] }</pre> <p>#当函数没有返回值时，默认返回最后一个命令的运行结果作为返回值 #函数头 <code>function</code> 可省略</p>
6.4.2 函数参数
<pre>FunWithParam(){ echo "第一个参数为 \$1 " echo "第十个参数为 \${10} " } FunWithParam 1 2 3 4 5 6 7 8 9 10 11</pre> <p><code>\${0}</code>代表执行文件文件名 #当 <code>n>10</code> 时，需要采用<code>\${n}</code>来获取返回值</p>
6.4.3 函数作用域
<p>#1. <code>Fun</code> 不创建子进程，函数外部或函数内部定义和使用变量的效果相同。 #2. 即在函数外定义的变量函数内可使用 #3. 在函数内定义的变量函数外也可使用 #4. 为了区分函数作用域，在函数内定义变量时使用关键字:<code>local</code></p>
6.5 重定向
<p>标准输入：命令读取输入的 <code>buffer</code>，通常为终端； 标准输出：命令结果输出的 <code>buffer</code>，通常也是终端； 一般情况下，每个 <code>Unix/Linux</code> 命令运行时都会打开三个文件：</p> <ul style="list-style-type: none">a. 标准输入文件(<code>stdin</code>)：<code>stdin</code> 的文件描述符为 <code>0</code>，<code>Unix</code> 程序默认从 <code>stdin</code> 读取数据。b. 标准输出文件(<code>stdout</code>)：<code>stdout</code> 的文件描述符为 <code>1</code>，<code>Unix</code> 程序默认向 <code>stdout</code> 输出数据。c. 标准错误文件(<code>stderr</code>)：<code>stderr</code> 的文件描述符为 <code>2</code>，<code>Unix</code> 程序会向 <code>stderr</code> 流中写入错误信息。 <p>但有些时候我们可能需要将数据从其它文件读入或读出，这就需要我们重定向。</p>

6.5.1 输入重定向
<p>命令原先需要从标准输入 <code>stdin</code> 中获取，转换为从我们的指定文件中获取。</p> <p><code>command < FILE</code></p>
6.5.2 输出重定向
<p><code>command > FILE</code></p> <p>#将 <code>command</code> 命令的输出结果覆盖存储在 <code>FILE</code> 文件中，覆盖原先文件</p> <p><code>command >>FILE</code></p> <p>#将 <code>command</code> 命令的输出结果延后存储在 <code>FILE</code> 文件中，不覆盖原文件</p>
6.5.3 标准错误文件重定向
<p>标准错误文件的文件描述符重定向 <code>stderr</code></p> <p><code>command 2>file</code></p> <p>将 <code>stdout</code> 标准输出文件和 <code>stderr</code> 标准错误文件合并重定向到一个指定文件中</p> <p><code>command > file 2>&1</code></p>
6.5.4 Here Document
<p>一种特殊的多行重定向方式；在构建文件时(特别是命令)很有效，不做详细介绍。</p>
6.5.5 禁止输出：/dev/null 文件
<p><code>/dev/null</code> 是一个特殊的文件，写入到它的内容都会被丢弃；如果尝试从该文件读取内容，那么什么也读不到。</p> <p>即：</p> <p><code>command > /dev/null</code></p> <p>可起到禁止输出的作用</p>
6.6 文件操作
6.6.1 文件操作

第8章 常见报错及处理