上海交通大学Linux用户组

# GNU/Linux 使用介绍（续）

骆铮 @ SJTUG

2016 年 4 月 10 日

# 目录


1. 文件管理
   - 复习
   - 创建文件
   - 查看目录大小
   - 查找文件
   - 权限与执行
   - 连接
   - 小测验
2. Virtualenv
   - 介绍
   - 安装和使用
   - 配合 requirements.txt
3. Bash
   - Bash 小技巧
   - 管道
   - Shell golf!
4. 进程管理
   - 后台运行
   - 进程管理
   - 前台/后台切换
   - 总结图

# 1 文件管理

*Everything is a file.*

# 复习

- `ls [-l] [-a] [directory]` (list) 查看一个目录
- `cd [directory]` (change directory) 更改当前目录
- `.` 代表当前目录，`..` 代表父目录，`-`代表上一目录，`~`代表当前用户的 home（一般是`/home/username`）
- `mkdir test` 创建目录 (make directory)
- `cp src dst` 和 `mv src dst` 复制 (copy)、移动 (move) 文件/目录
- `rm [-rf] file_or_dir1 ...` 删除 (remove) 文件/目录

# 创建文件

一般来说，创建文件有两种方式：

`touch filename` 若 `filename` 不存在，创建空文件；否则更新已有的 `filename` 的日期。

`command > output.txt` 用 `command` 的输出来作为 `output.txt` 的内容

**Example**

- `date > output.txt`
- `cat output.txt`
- 结果为当前的日期时间

# 查看目录大小

默认的 `ls` 命令并不会显示出目录大小，所以需要使用 `du` (disk usage) 命令

**用法**

```
du [-h] [-s] [File]
```

`-h` human-readable，自动选取 KB,MB 作为单位显示

`-s` summarise，只显示总数，而不显示下面各个子文件夹的大小

# 查找文件

使用 `find` 命令查找文件。

**用法**

`find root_path -name '*.py'`: 查找 `root_path` 下的所有`.py` 文件

Q&A

- 为什么不能是`find root_path -name *.py`
- 因为不带单引号的 `*.py` 默认在命令执行前展开成当前目录下符合要求的文件名
- 即实际执行的是：查找与当前目录下`.py` 文件名相同的文件

# 执行

Linux 并不以扩展名区分文件类型。所有带有 x 权限的文件几乎都可以执行。

**用法**

执行当前目录下名称为 prog 的方式是：./prog

./不可忽略

- 否则假如有人在~目录下放了一个叫做 ls 的病毒，习惯性输入 ls 就会运行当前目录的病毒
- 所以 GNU/Linux 系统要求显式指定目录

# 权限

用 `ls -al` 就能看到当前目录下文件权限

## 常见权限

`r` read，是否可读
`w` write，是否可写
`x` execuate，对于文件（狭义）来说表示可执行，对于文件夹来说表示是否可打开

## 各位含义

文件标识一般有 10 位，常见是 `drwxr-xr-x`

- 第一位如果是 `d` 表示是目录，`-`是普通文件
- 第 2 到 4 位表示所有者的权限。
- 第 5 到 7 位表示拥有该文件 group 中用户的权限
- 第 8 到 10 位表示其他用户的权限

# 修改权限

一般来说，文件拥有者或 root 可以用 chmod 修改一个文件的权限。

参考：`drwxrwxrwx`

**Example**

`chmod u+x file` 给当前用户执行 `file` 的权限，对应着设置 10 位中的第 4 位。

`chmod g-w file` 剥夺同组用户写入 `file` 的权限，对应着去除 10 位中的第 6 位

`chmod a=g file` 让非同组用户拥有和同组用户相同的权限，对应着把第 5 到 7 位复制到了第 8 到 10 位

# 连接

一个文件，多个位置。类似于 Windows 下的“快捷方式”，但更灵活。

**用法**

`ln -s /src/file1 /dst/file2`，创建一个在`/dst/file2`的连接，其指向了`/src/file1`

- `ln` = `link`
- 之后对`/dst/file2` 的操作就像是在`/src/file1` 上一样
- `rm /dst/file2` 只是删除了连接，没有删除`/src/file1`，这就是`-s` 选项创建的软连接 (symbolic link)
- 推荐创建连接时`/src/file1` 采用绝对路径，否则容易产生问题

# 小测验

- 如何将当前目录下所有文件名（每行一个）写入 `filename.txt`?
- 如何查看`/tmp`（注：这是临时文件夹）的大小?
- 根据网上的教程，在执行`./install.sh` 的时候，显示 `bash: Permission Denied: ./install.sh`，应该如何操作?

# 小测验

- 如何将当前目录下所有文件名（每行一个）写入 filename.txt?
  答案：`ls > filename.txt`
- 如何查看/tmp（注：这是临时文件夹）的大小?
  答案：`du -hs /tmp`
- 根据网上的教程，在执行./install.sh 的时候，显示 bash: Permission Denied: ./install.sh，应该如何操作?
  答案：有两种方法：
  - chmod u+x install.sh，然后再执行./install.sh
  - 或者 source install.sh，这样的话就不算运行 install.sh 了，所以也不需要 x 权限了

# 2 Virtualenv

---

*Don’ t let the noise of others’ ~~opinions~~ Python packages break your own ~~inner voice~~ dependencies.*

# VitualEnv: 介绍

## 背景

- 越来越多的 Python 程序
- 越来越多的 Python 第三方包通过 `easy_install/pip install` 安装
- 同一个用户乃至同一个系统中所有的 Python 包是共享的。容易出现版本问题

## VirtualEnv 介绍

VirtualEnv 提供了一种隔离措施：即进入一个虚拟环境，之后安装的所有包都会保存在这个目录下面

# 安装与使用 VirtualEnv

1. Ubuntu 下使用 `apt-get install python-virtualenv` 安装
2. 使用 `virtualenv ENV1` 在当前目录下创建一个叫做 ENV1 的文件夹（即虚拟环境）
3. 使用 `cd ENV1`，然后再 `source bin/activate` 方式进入虚拟环境
4. 之后，用 `easy_install` 或 `pip install` 安装包，用 python 运行 Python
5. 使用 `deactivate` 退出虚拟环境，所有虚拟环境中安装的包都不会影响主机

# 配合 requirements.txt

VirtualEnv 配合 `requirements.txt` 能够大幅度减少依赖的问题。

**Example**

我开发了一个 Python 项目，它需要 `bs4` 和 `requests` 等第三方库作为依赖

在本机，新建一个 `requirements.txt`，其内容是：

```
bs4
requests
```

# 配合 requirements.txt

## Example

然后不论是在本机还是服务器，我都能如下跑起这个 Python 程序，同时不会造成包污染：

1. `virtualenv MyAppEnv`
2. `cd MyAppEnv`
3. `source bin/activate` 进入虚拟环境
4. `pip install -r prog_path/requirements.txt` 或 `easy_install `cat prog_path/requirements.txt`` 安装依赖
5. `python prog_path/my_prog.py` 运行
6. `deactivate`

# 3 Bash

---

*Shell n. the covering or outside part of a fruit or seed especially when hard or fibrous.*

# Bash 小技巧

记住一些 Shell 下常用的按键能大幅提升操作效率。

| 命令 | 作用 |
| --- | --- |
| Ctrl-b | backward, 光标左移一位，相当于 Left |
| Ctrl-f | forward, 光标右移一位，相当于 Right |
| Alt-b | 光标左移一个单词 |
| Alt-f | 光标右移一个单词 |
| Ctrl+p | previous, 查看上一条命令，相当于 Up |
| Ctrl+n | next, 查看下一条命令，相当于 Down |
| Alt+backspace | 删除一个词 |
| Ctrl+a | 跳到行首 (a 是第一个字母，所以代表行首) |
| Ctrl+e | 跳到行尾 (end) |

# 管道

管道是将一个程序的输出作为另一个程序的输入。

**复习**

```
ls *.c | grep malloc
```

1. `ls *.c` 输出了当前目录下所有 .c 文件的名称（每个一行）到标准输出
2. 然后管道把这些文件名作为 `grep` 的输入
3. 因为 `grep` 只指定了一个参数（待查找的正则表达式 `malloc`），所以它从标准输入读入数据
4. 最后 grep 向标准输出输出结果

Tip：绝大多数命令不指定文件名都会从标准输入读入

# Shell golf!

## 背景

`cut -d' ' -f 1` file 输出 file 中以空格分开的第一列

`uniq -c` 将连续的重复的 k 行，变成 k line_content 的形式

`sort` 将各行按字母序排序

`sort -nr` 根据每一行第一个的数字倒序排序

~/.bash_history 存储了输过的命令，其格式为：

```
ls /home/lz
kill 5962
ls .
```

## 要求

输出最常用的命令及次数。例如上例中应该输出

```
2       ls
1       kill
```

# Answer

## 问题

- cut -d' ' -f 1 file输出 file 中以空格分开的第一列
- uniq -c命令，将连续的重复的 k 行，变成 k line_content 的形式
- sort 命令：将各行按字母序排序。sort -nr 命令：根据每一行第一个的数字倒序排序
- ~/.bash_history 的格式：

```
ls /home/lz
kill 5962
ls .
```

要求：输出最常用的命令及次数，例如上例中应该输出

```
2       ls
1       kill
```

## 答案

```
cut -d' ' -f 1 ~/.bash_history  |
sort |
uniq -c |
sort -nr
```

# 4 进程管理

---

*All members want to accomplish different things. However, to achieve some of these diverse ends, concerted, interdependent actions are required.*

# 问题

- 不知道怎么在后台运行程序?
- 已经运行的程序怎么转到后台运行?
- 在后台重复运行了两个程序?
- 为什么跑起来这么卡?

Screen/TMux 都是解决一部分问题的工具，但在这里要介绍的是更加原生的一些命令

# 后台运行

背景：yes 会重复输出 y

**直接运行** `yes` 用 `Ctrl-C` 结束掉。
**后台运行** `yes &`

- Q: 当程序请求输入时会被挂起，但正常输出
  A: 利用重定向 `yes > output.txt &` 之后利用 `less`、`cat` 等命令查看 `output.txt`
- Q：不能用 `Ctrl-C` 结束
  A: `killall yes`
- Q：断开服务器 ssh 的连接后这个进程就结束了
  A：正常现象。如果想在服务器断开后还运行有三种方法：
  - 运行 `nohup yes > output.txt &`。好处是简单，坏处是必须重定向输入输出。
  - 在 `yes &` 运行之后，使用 `disown %job-id`。job-id 一般是 1 位数字，效果同上
  - 使用 `screen`

# 进程管理

`ps` 查看当前终端在运行的进程

`ps -a` 查看所有与终端关联的、且不是会话首进程的进程（基本直接执行的命令都是这类）

`ps -A` 查看所有进程

`ps aux | grep command` 查找运行命令行包括 command 的所有进程

`kill process-id` 结束指定进程号的进程

`kill -9 process-id` 强制结束（可能造成数据丢失）

`top` 可视化查看

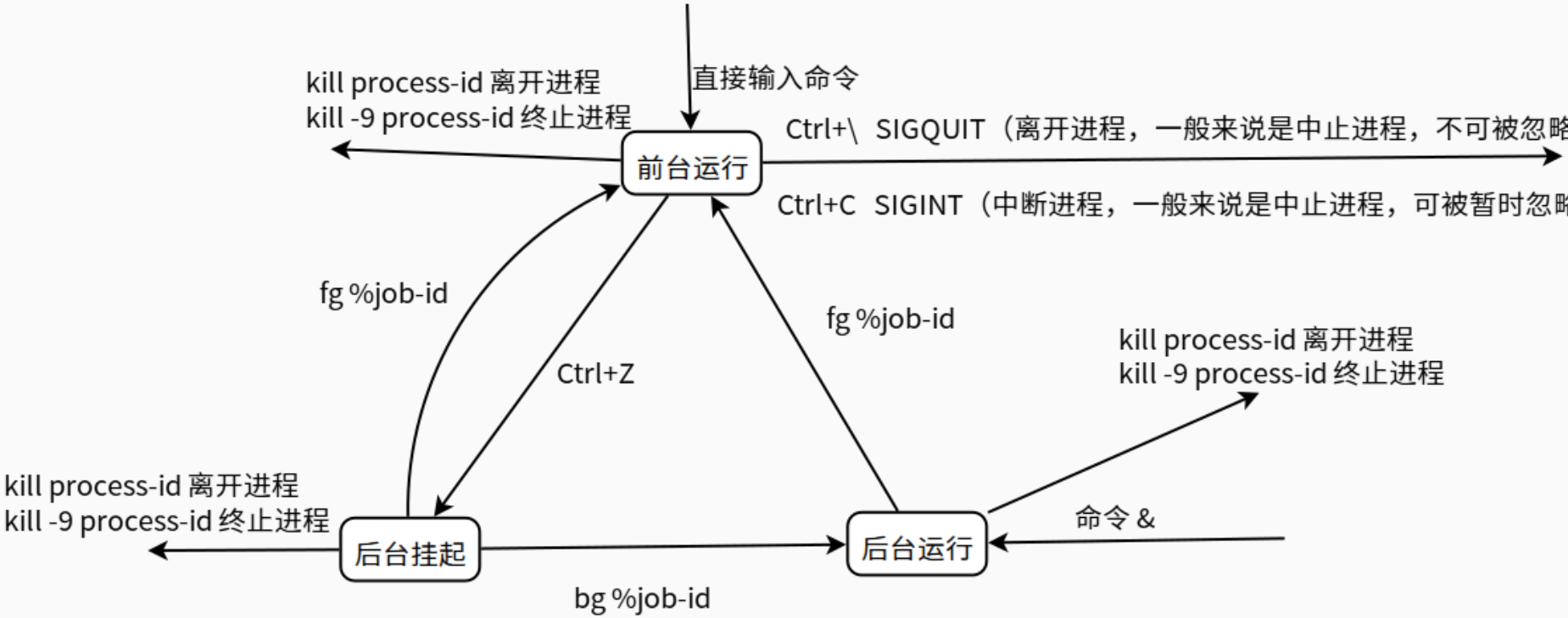
`R` 切换 sort 的列

`k` 杀死选中进程

# 前台/后台切换

| | |
|---|---|
| `jobs` | 查看当前所有后台挂起、后台运行、后台停止的工作 |
| `Ctrl-Z` | 前台 $\rightarrow$ 后台挂起 |
| `bg %job-id` | 后台挂起 $\rightarrow$ 后台运行 |
| `fg` | 后台挂起/后台运行 $\rightarrow$ 前台运行 |

# A picture is worth a thousand words



# 感谢倾听

上海交通大学Linux用户组