

Linux操作系统使用

授课人：董燎原

中国科学院高能物理研究所

教学内容与目的

- ✚ Linux介绍和应用
- ✚ 基本命令使用
- ✚ 常用工具和命令
- ✚ 文本编辑工具
- ✚ 脚本语言编程(shell, awk, sed)
- ✚ GCC编译器使用
- ✚ 程序开发与测试

Linux操作系统简介



Linux是基于Linux内核的Unix-like计算机操作系统的一个统称（中文一般念成：离呢克斯）。

Linux是一个遵循POSIX (Portable Operating System Interface) 标准的免费操作系统。

- 它的核心是一个面向命令行(shell)的操作系统，为实现编程的目的。
- 它也提供了图形用户界面 (GUI) ，配合系统的工作。

Linux操作系统的特色

- Linux操作系统和其提供的工具程序的开发是Linux内核最初的作者Linus Torvalds (1991年10月5日)和众多的用户共同努力的结果。
- 它最大的特色之一就是这个操作系统的**全部源代码是公开的**。
Linux操作系统遵从GNU公共许可证 (GNU Public License, 简称GPL) 的规定，用户可以自由地使用、拷贝、修改甚至重新发布这个软件。

详细介绍见

<https://en.wikipedia.org/wiki/Linux>

Linux操作系统的发行版本

Linux广泛应用于服务器，嵌入系统，手机和超级计算机中。
随着Linux易用性的提高，在个人计算机上的应用越来越广泛。
目前Linux家族已经有了近140个不同的版本。

https://en.wikipedia.org/wiki/List_of_Linux_distributions

常见的版本有（mirror.ihep.ac.cn 可下载）：

桌面系统和服务器：

ubuntu



redhat



桌面系统(界面华丽漂亮，办公首选)：

科学计算服务器：



Scientific Linux

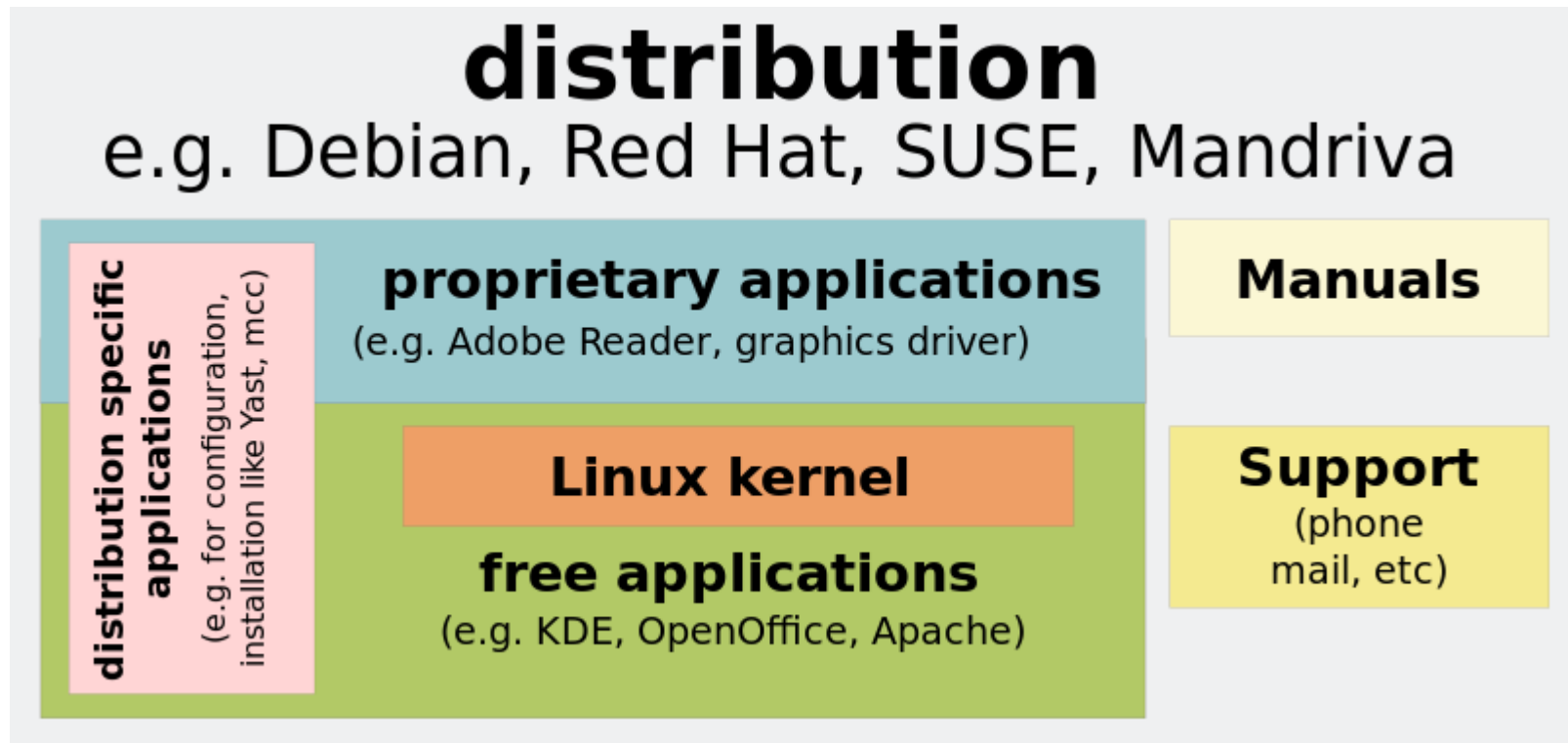
<https://www.scientificlinux.org/>



CentOS

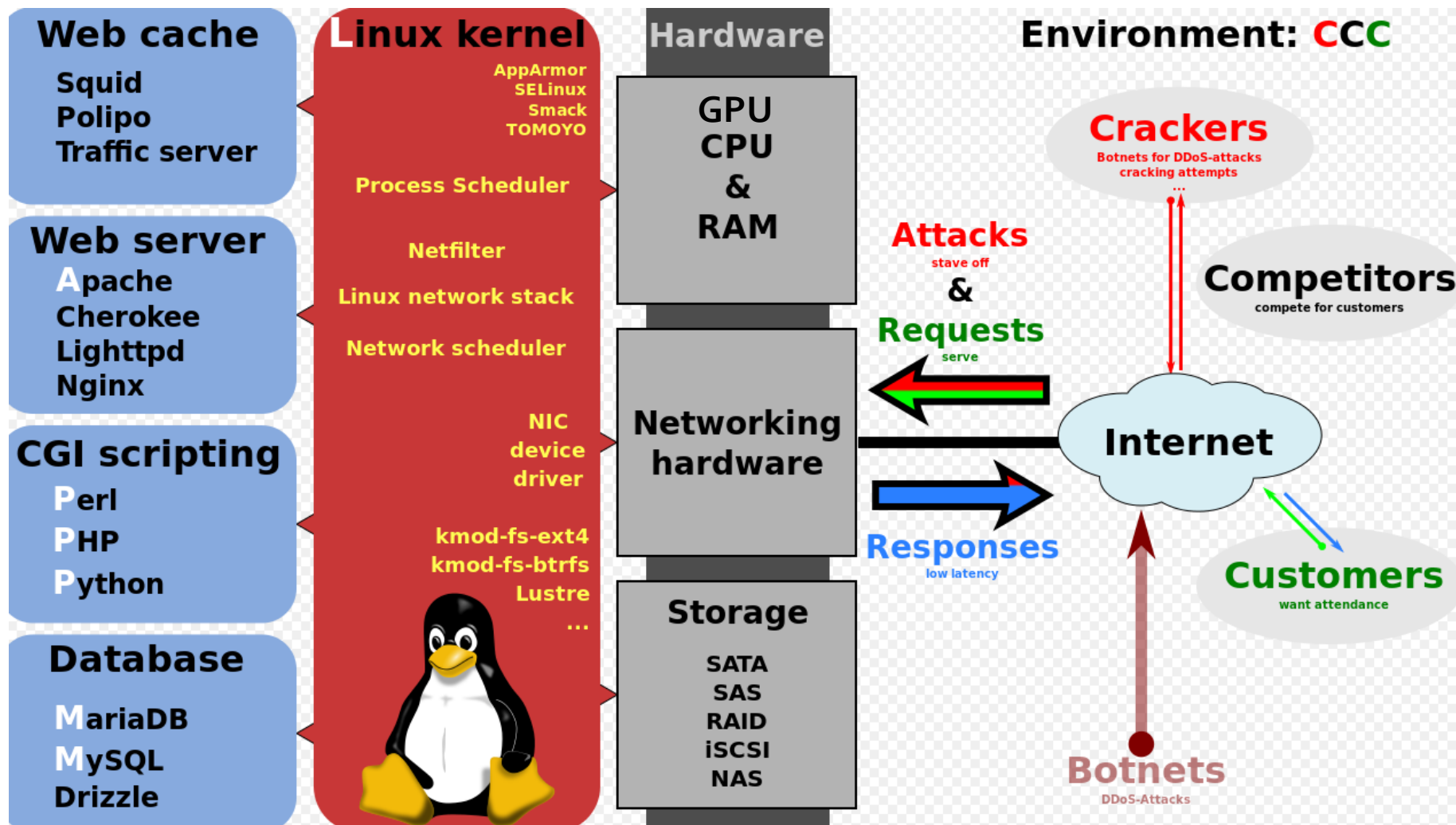
高能所计算中心 准备从 CentOS 7.9 升级到 Alma Linux 9.2 (2024.6)

Linux发行版的内容



Ref: http://en.wikipedia.org/wiki/Linux_distro

Linux提供的服务



Linux可提供网络，存储，数据库，编程，web服务，web cache 等服务！

Linux支持的常用编程语言

GNU Compiler Collection



提供 C, C++, 和 Fortran 程序的编译.



server-side scripting language, 用于web服务的开发!



is a widely used high-level, general-purpose, interpreted, dynamic programming language.

Perl



is a family of high-level, general-purpose, interpreted, dynamic programming languages.



Linux的安装

Installation Settings

1. 下载linux光盘映像，刻录到光盘（也可其他方式ftp，NFS，USB，或者硬盘），
2. 阅读发行版的安装指南文档，按步骤安装，
3. 硬盘分区：\根目录区(50-100GB)，\home区(用户区)， swap (~12GB)交换区，
4. 如果需要桌面环境，可选择KDE和Gnome图形桌面环境，
5. 建立root（超级用户）的密码，以及普通用户账号和密码，
6. 配置网络，需要静态IP的地址，若没有设置动态分配IP，
7. 选择安装所需的软件包，如编译器gcc和所需的语言支持等，
8. 选择所需启用的服务，如需要web和数据库的服务，选apache和mysql软件包，
9. 安装完成后，升级系统，打安全补丁，
10. 选择需要安装的额外软件，如 root，firefox，adobe reader等。

Click a headline to make changes.

Bootimg

- Boot Loader Type: GRUB2
- Status Location: /dev/sda2 ("/")
- Change Location:
 - Do not install bootcode into MBR ([install](#))
 - Install bootcode into "/" partition ([do not install](#))
- Order of Hard Disks: /dev/sda, /dev/sdb, /dev/sdc

Software

- Product: openSUSE
- System Type: KDE Desktop
- Patterns:
 - + Base System
 - + Enhanced Base System
 - + AppArmor
 - + Laptop
 - + YaST System Administration
 - + Software Management
 - + KDE4 Desktop Environment
 - + Multimedia
 - + KDE4 Base System
 - + Office Software
 - + X Window System
 - + Fonts
 - + Graphics
 - + Games
 - + Misc. Proprietary Packages
- Size of Packages to Install: 3.5 GiB

Default systemd target

- Graphical mode

System

- [System and Hardware Settings](#)

Firewall and SSH

- Firewall will be enabled ([disable](#))
- SSH port will be blocked ([open](#))
- SSH service will be disabled ([enable](#))

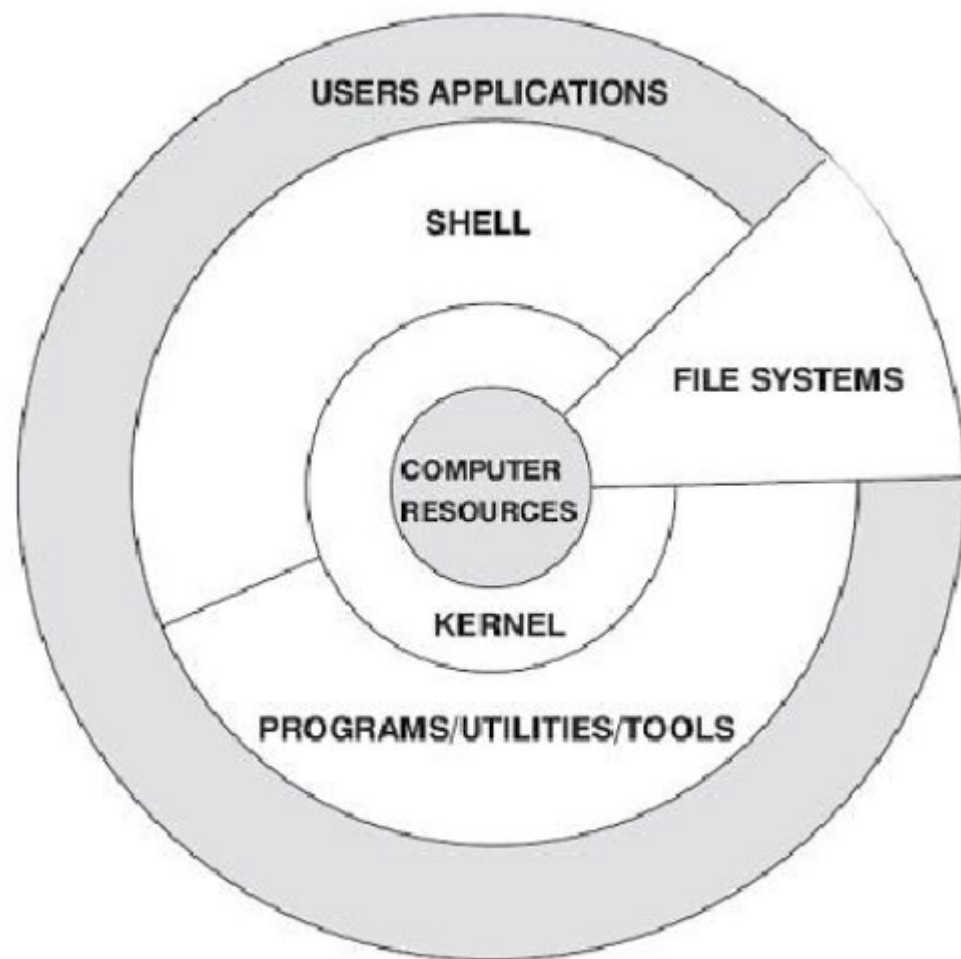
Unix/Linux 系统结构

Linux系统有4个主要部分：

内核：连接应用程序与硬件
Shell（命令行解释器）

文件系统

应用程序：基本系统命令，
编译器（GCC），vi文本编辑器，KDE（窗口管理器）等。



内核（**Kernel**）是操作系统的核心，具有很多最基本功能，它负责管理系统的进程、内存、设备驱动程序、文件和网络系统。

Kernel文件名一般是vmlinuz，放在/boot目录下。

Linux文件类型

- ✚ 普通文件（Regular files, 标识为-）
- ✚ 目录（Directories, 标识为d）：files that are lists of other files.
- ✚ 链接文件（Links, 标识为l）：a system to make a file or directory visible in multiple parts of the system's file tree.
- ✚ 特殊文件（Special files, 标识有c,i,s等等）：如标识c为压缩读出/写入标识，常用在/dev内的设备文件上。

```
[dongly@lxslc601 ~/work]$ ls -la
total 16
drwxr-xr-x  3 dongly physics  2048 Feb 27 16:10 .
drwxr-xr-x 55 root    root    10240 Feb 27 16:04 ..
-rw-r--r--  1 dongly physics    0 Feb 27 16:10 .backup
-rw-r--r--  1 dongly physics   10 Feb 27 16:05 check.sh
lrwxr-xr-x  1 dongly physics    8 Feb 27 16:05 ck.sh -> check.sh
-rw-r--r--  1 dongly physics    0 Feb 27 16:04 count
drwxr-xr-x  2 dongly physics  2048 Feb 27 16:05 results
```

Linux系统文件结构

/bin (binary)

这个目录包含着所有的标准命令和应用程序.

/boot

这里存放给系统启动需要使用的一些文件.

/etc

这个目录包含着系统设置文件和其他的系统文件,它在Linux下极为重要:
fstab这个档案相当重要,记录开机要 mount 上来的 filesystem;
passwd (用户信息), shadow(加密的密码,不可见), group (分组信息);
ld.so.conf 系统动态链接共享库的路径,记录一些 library 所在的目录,
用ldconfig可以重新产生。

/home

存放用户主目录的地方,一般说"/home/username"就是用户的主目录.

/lib(library)

存放系统最基本的动态连接库.

/lost+found

这个目录一般都是空的.但当文件系统发生故障,恢复文件用。

/mnt

空目录,是让用户临时挂载文件系统的地方.

/proc

是linux提供的一个虚拟系统,是由系统在系统启动的时候在内存中产生的，用户可以 直接通过访问这些文件来获得系统信息，例如/proc/kcore就是系统运行的时候内存的映象文件。

/root

超级用户主目录（一般用户无法访问）。

/sbin

这个目录存放着super user使用的系统管理程序，如fsck， mount等。

/tmp (temporary)

存放不同的程序执行时产生的临时文件（不要主动把文件存在这个目录）。

/usr (user)

linux系统占地最大的一个目录，用户的很多应用程序和文件都几乎存放在这里。

/var

这个目录中存放着那些不断在扩充着的东西，如log文件等。

使用Linux

本地登录:

输入用户名和密码即可进入系统。

远程登录:

1. 从linux机器登陆到另一台机器:

`ssh -l dongly lxslc7.ihep.ac.cn`

以账户dongly登陆lxslc7的机器

若加-Y参数可将图形显示过来。

2. 从Windows系统登陆:

可用terminal软件:

NetSarang.Xmanager.Enterprise等

关闭系统: 以root身份登陆后,

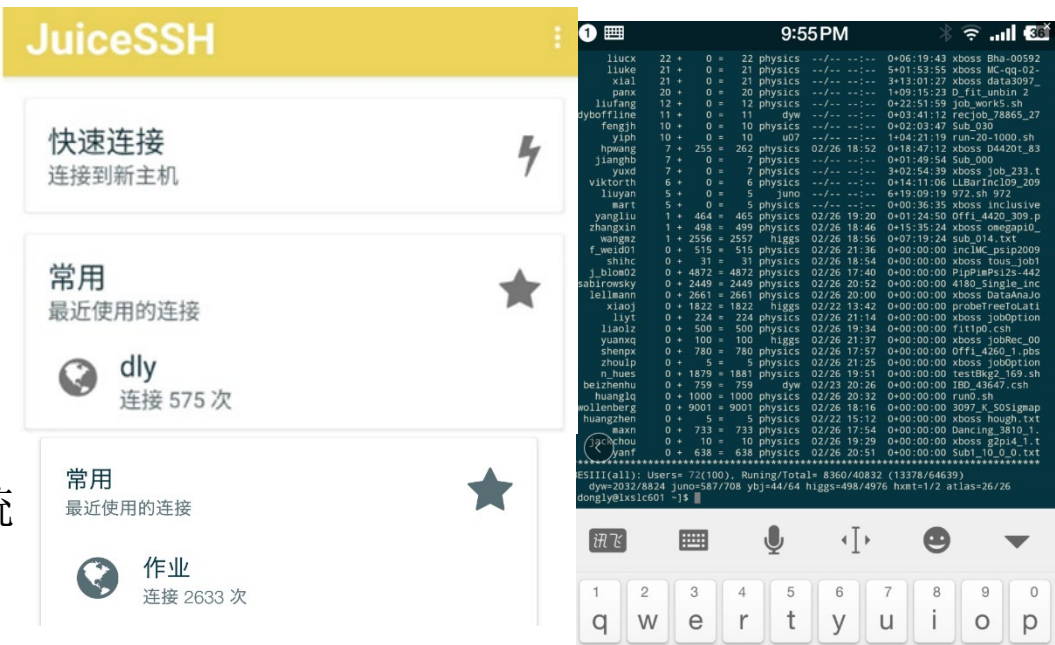
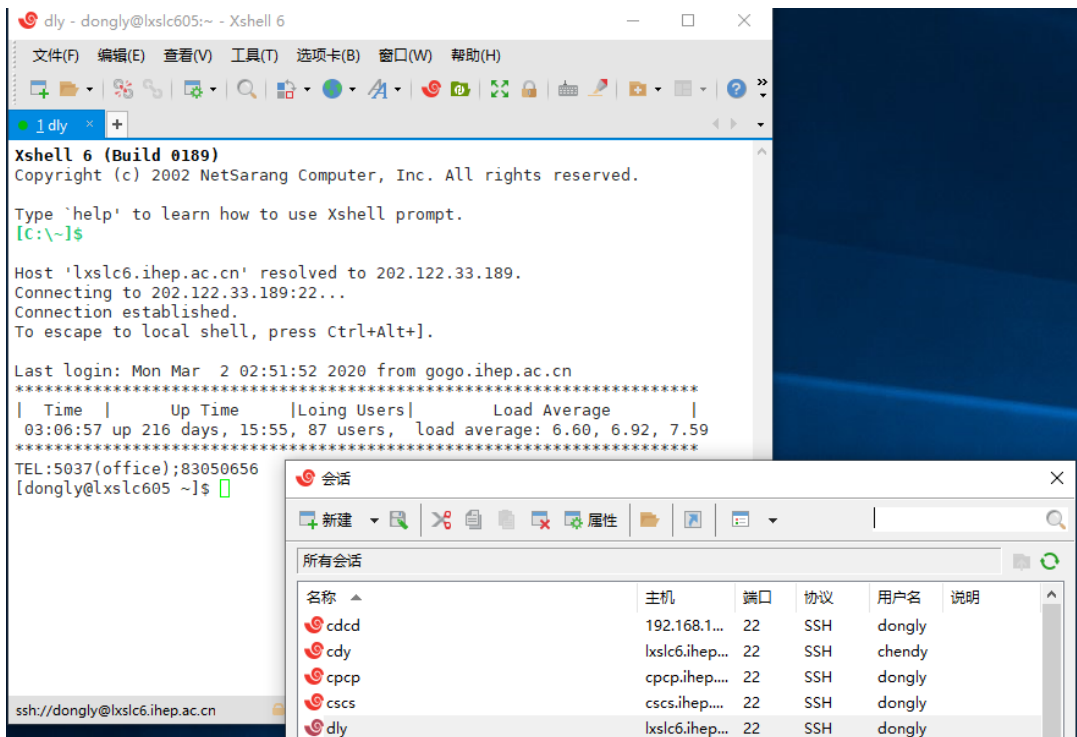
输入: `shutdown -h`

重启系统: 以root身份登陆后,

输入: `shutdown -r`

也可用桌面的菜单关闭或重启系统

3. 从手机登陆: JuiceSSH



Linux命令

- ✚ 基本语法 (Syntax) : `command [options] [arguments]`
- ✚ 命令 (command), 选项 (options) 和参数 (arguments) 之间空格隔开; 方括号 [] 表示可选 (*可有可无*) 。
- ✚ 选项一般情况下以短线 “-” (hyphen) 开头
- ✚ 多个选项 (Options) 可以放在一起:

```
[dongly@lxslc601 ~/work]$ ls -ltr ~/work/      (按时间顺序详细显示)
total 4
-rw-r--r-- 1 dongly physics      0 Feb 27 16:04 count
drwxr-xr-x 2 dongly physics 2048 Feb 27 16:05 results
-rw-r--r-- 1 dongly physics    10 Feb 27 16:05 check.sh
lrwxr-xr-x 1 dongly physics      8 Feb 27 16:05 ck.sh -> check.sh
[dongly@lxslc601 ~/work]$
```

- ✚ 大小写敏感
- ✚ 利用man命令, 可以参看各个命令的使用方法:

```
[dongly@lxslc601 ~/work]$ man ls (按q键退出帮助页面)
```

修改账号密码passwd

✚ 语法: passwd [用户名]

修改自己的密码只需用 passwd,
只有root账号才能输入用户名, 为用户该密码。

```
> passwd
Changing password for dongly.
(current) UNIX password: <输入旧密码>
New UNIX password: <输入新密码(最好为大于6位, 英文字母与数字混合)>
>
Retype new UNIX password: <再输入一次密码>
passwd: all authentication tokens updated successfully.
>
```

文件操作

文件操作命令

Commands	Output/Result
<code>pwd</code>	Prints the current directory path
<code>cd</code>	Changes directory
<code>ls</code>	Lists out the directory contents
<code>touch</code>	Creates a file
<code>cp</code>	Copies a file
<code>rsync</code>	Copies a remote (and local) file
<code>mv</code>	Moves a file to a different directory
<code>ln</code>	Links to a file or directory
<code>mkdir</code>	Creates a new directory
<code>rm</code>	Removes a file
<code>chmod</code>	Changes file permissions
<code>chown</code>	Changes file ownership
<code>du</code>	Estimates file space usage

改变目录cd

- ✚ **pwd** ---- print working directory 显示当前所在目录
- ✚ **cd** ---- changing directory 改变目录

<code>cd work</code>	This moves you to the work directory.
<code>cd .</code>	This moves you nowhere, but stay in your current directory.
<code>cd ..</code>	This moves you to the directory above your current directory.
<code>cd /</code>	This moves you to the root directory.
<code>cd ../..</code>	This moves you to TWO directories above your current directory.
<code>cd ~</code>	This moves you to your user home directory.
<code>cd -</code>	This moves you to the last visited directory.

- ✚ use **/** to connect the subdirectory name: `/home/cp`
- ✚ a single period or **'.'** means the current directory and double periods **'..'** means the directory above

目录中的文件ls

✚ Syntax : **ls** [**options**] [**file**]

✚ default: list files in the current directory.

✚ Options :

✚ **-a**: This lists all files, including hidden files.

✚ **-l**: This lists files in a long listing format.

✚ **--color**: color is used to distinguish file types.

To find more options, use the “**man ls**” command.

✚ Examples of **ls** command in use

<code>ls /home/dongly</code>	This lists all files in that directory.
<code>ls /home/dongly/*.sh</code>	This lists all the files with the file extension of <i>.sh</i> .
<code>ls /home/dongly/t*</code>	This lists all files beginning with the letter <i>t</i> .

常用: `ls -lrt` : 按**修改时间倒序**列出当前目录下的所有文件的详细信息。

文件创立touch

Creating Files

 Syntax: **touch** **[option]** **[file]**

```
[dongly@lxslc601 touch]$ touch DONE
[dongly@lxslc601 touch]$ ls -l
total 0
-rw-r--r-- 1 dongly physics 0 Mar  1 13:42 DONE
```

文件复制cp

Copying Files

✚ Syntax: `cp [option] [source_file] [target_file]`
`cp [option] [source_dir] [target_dir]`

✚ option:

-i	The interactive option. The cp command will prompt the user before actually copying the file(s).
-f	Overwrite an existing destination file. (有危险, 小心操作!脚本中常用)
-r	This forces the cp command to copy the files recursively.
-p	This preserves the specified attributes (default: mode, ownership, timestamps) and security contexts, if possible. Additional attributes: links, all

```
cp -r /dir/* /dir2/ will copy all files and subdirectories omitting dir
cp -r /dir/ /dir2/ will copy all files and subdirectories including dir
```

文件同步rsync

rsync命令是一个远程数据同步工具，可通过网络快速同步多台主机间的文件。

这个命令只传送两个目录的不同部分，而不是每次都整份传送，因此速度相当快。

例子：

```
/usr/bin/rsync -av /bes/dongly/round21 .
```

开关：

-a, --archive 归档模式，表示以递归方式传输文件，并保持所有文件属性。

-v, --verbose 详细模式输出。

功能：将本地/bes/dongly/round21（整个目录及里面的所有子目录和文件）备份到当前目录。

这个命令可以反复使用，直到两个目录完全相同。

文件移动mv

Moving Files or Renaming Files

✚ Syntax: `mv [option] [source_file/dir] [target_file/dir]`

✚ option:

-i	The interactive option. The mv command will prompt the user if the target_file/dir already exists.
-f	Overwrite the existing target file(s). (有危险，小心操作！脚本中常用。)
-b	Backup the existing target file(s) with a "~"-end new file name.
-v	The verbose option, which forces mv to report what's being done.

✚ **mv** command can be used to rename file or directory

建立链接(Linking Files) ln

Syntax: `ln [option] [source_file] [target_file]`

✚ Option: 默认是硬链接

-s: 软链接

✚ hard link: `ln file1 file2`

修改file1和file2任意一个, 另一个文件随之更新; 删除file1, file2依然存在; file1与file2在物理储存上是一致的。

✚ soft link: `ln -s file1 file2`

file2是file1的符号链接; 删除file1, file2将没有任何内容; 符号链接可以创建一个指向目录的链接。

```
# touch foo
# ln -s foo goo
# ln foo boo
```

<code>-rw-r--r--</code>	2	root	root	0	Nov	1	01:43	boo
<code>-rw-r--r--</code>	2	root	root	0	Nov	1	01:43	foo
<code>lrwxrwxrwx</code>	1	root	root	3	Nov	1	01:43	goo -> foo

建立目录mkdir

Creating Directories

✚ Syntax: **mkdir** [**option**] [**directory_name**]

✚ option:

-p	no error if existing, make parent directories as needed
-v	It prints a message when the directory is created.

✚ Example:

```
mkdir documents
```

```
mkdir -p work/foo/num1
```

(建num1目录, 如果work和foo目录不存在, 也一起建好)

文件删除rm

Removing Files

✚ Syntax: **rm** [option] [file1][file2]
 rm [option] [dir1] [dir2]

✚ option:

-i	The interactive option. The rm command will prompt the user before actually removing the file(s) (执行前确认, 这个重要!).
-f	Remove the file(s) without prompt. (有危险, 小心操作! 脚本中常用)
-r	This forces the rm command to remove the files recursively .
-v	The verbose option.

✚ option **-i** is recommended: 提示交互操作

✚ Don't use **rm -fr /*** as root privilege.

```
rm -rf /dir1/* removes all files and subdirectories leaving dir1 empty
```

```
rm -rf /dir1/  removes all files and subdirectories including dir1
```

以上命令删除目录下所有文件, 小心使用!

文件属性

```
[dongly@lxs1c601 ~/work]$ ls -l --color
total 6
-rw-r--r-- 1 dongly physics 10 Feb 27 16:05 check.sh
lrwxr-xr-x 1 dongly physics 8 Feb 27 16:05 ck.sh -> check.sh
-rw-r--r-- 1 dongly physics 0 Feb 27 16:04 count
drwxr-xr-x 2 dongly physics 2048 Feb 27 16:05 results
drwxr-xr-x 2 dongly physics 2048 Mar 1 13:42 touch
[dongly@lxs1c601 ~/work]$
```



Permissions



Owner



Group



File Size



Date/Time



File/Directory Name

-rw-rw-r-- 1 root java 242 Apr 21 2002 data.txt

chmod chown

-	r	W	-	r	W	-	r	-	-
u			g			o			

the permission levels:

r	Read
w	Write
x	Execute
-	No permission

设置文件权限chmod

Changing File Permission

✚ Syntax: **chmod** **[options]** **[file]**

✚ mod:

✚ option:

User	u
Group	g
Others	o
All	a

Read	4
Write	2
Execute	1

Read	r
Write	w
Execute	x

+	Adds permissions 给与权限
-	Removes permissions 取消权限
-R	Recursive. Recursively changes permissions to files and directories. 对当前目录下所有目录和文件操作!

设置文件权限chmod

<code>chmod 755 dir1</code>	对于目录 dir1 ，设定成任何使用者皆有读取及执行的权利，但只有所有者可做修改（ 一般用这个选项 ）。
<code>chmod 700 file1</code>	对于文件 file1 ，设定只有所有者可以读、写和执行的权限（ 文件保密的选项 ）。
<code>chmod u+x file2</code>	对于文件 file2 ，增加当前用户可以执行的权利。
<code>chmod g+x file3</code>	对于文件 file3 ，增加工作组使用者可执行的权利。
<code>chmod o-r file4</code>	对于文件 file4 ，删除其他使用者可读取的权利。

改变文件所有权chown

Changing file ownership

✚ Syntax:

```
chown [OPTION]... OWNER[:[GROUP]] filename  
chown [OPTION]... :GROUP filename
```

✚ `chown dongly:physics file1`

将文件`file1`改为用户`dongly`, 工作组`physics`所有。

✚ `chown -R user dir1`

将目录`dir1`及其子目录下面的所有文件改为用户`user`所有。

查看目录大小du

✚ 语法： `du [-s] 目录`

✚ `du dir1`

显示目录`dir1`的总容量及其子目录的容量(以KB为位)。

✚ `du -s dir1`

显示目录`dir1`的总容量。

✚ `du -sh *` (常用)

显示当前目录下各个子文件夹或文件的大小。

查询与搜索

查找文件

locate: 快速查找文件

 **locate database**

查找系统中所有含有**database**字符串的文件名和目录名。

 **locate -u**

建立或更新**locate**查询数据库

find: 查找文件

 **find /path -name database -print**

查找目录/**path**中含有**database**字符串的文件名和目录名，并显示出来。

例如: `find . "we*.bak" -print`

在当前及所有子目录寻找**we**开头加**.bak**结尾的文件。

 **find功能非常强大，并且可用正则表达式。**

例如: `find dir -name "*.txt"|xargs sed's/round04/round01/g' -i`

将**dir**目录的所有**txt**文件中的“**round04**”改为“**round01**”。

查找命令

which command

显示命令的路径，及使用者所定义的别名。

whereis command

显示命令command相关的重要文件

whatis command

显示命令command的功能摘要。

apropos keyword


在**whatis**数据库中查找与字符串**keyword**匹配的命令，并显示出来。

命令帮助文档

man **command**


组成段	说 明
Name	命令的名称及简单说明
Synopsis	如何使用这个命令及命令行参数
Description	对这个程序命令及其参数的解释
Files	这个命令用到的文件清单和它们存放的位置
See Also	有相互联系的使用手册页的清单
Diagnostics	特殊输出情况的说明
Bugs	编程漏洞
Author	程序的主要编写者和其他维护人员

info **command**

 提供GNU软件程序，自由软件基金会（FSF）及GUN/GPL的详细资料。

文本操作

查看文件内容 (常用)

 **cat file1**


以连续显示方式查看文件名file1的内容

 **more file1**

以分页方式查看文件名file1的内容。

 **less file1**

分页显示方式, 比more的功能更强大

 **head -n file1; tail -n file1**

显示文件file1头n行或者后n行的内容

Sim_001.txt.besslog:

第1行: Job 55454730.0 on RemoteHost = slot4@bws0897.ihep.ac.cn

倒数10行: ApplicationMgr INFO Application Manager Finalized successfully
ApplicationMgr INFO Application Manager Terminated successfully

文件内容查找grep

- ✚ Syntax: `grep string file`

- ✚ **`grep abc file1`** (常用)

查找并显示文件`file1`中包含字符串`abc`的行的文本内容。


- ✚ `grep`功能非常强大，支持正则表达式。

文件统计和比较

 **wc file**

统计和显示文件**file**的字节数，单词数和行数。

 **diff**

 **diff file1 file2** (常用)

比较文件**file1**与**file2**内各行的不同之处。

 **diff -r dir1 dir2**

比较目录**dir1**与**dir2**内各文件的不同之处。

 **diff -i file1 file2**

忽略大小写的差异

 **vimdiff file1 file2**

在vim编辑器中比较文件的差异

文本内容操作

✚ **sort files**

对文本文档的行进行排序。参考man帮助。

✚ **split files**

把文件分割为几个小文件。参考man帮助。

✚ **cut file**

从文件的每行中去除一些区域。参考man帮助。

✚ 其他相关：重定向符(>, >>, <), awk, sed等。后面将会介绍。

✚ 文本编辑器：vi, joe, emacs等

文件存档与压缩

tar

- ✚ `tar -xvf foo.tar`
verbosely extract `foo.tar`
- ✚ `tar -xzf foo.tar.gz`
extract gzipped `foo.tar.gz` (展开gzip压缩后的打包文件)
- ✚ `tar -czf foo.tar.gz bar/`
create gzipped tar archive of the directory `bar`
(将`bar`目录打包后压缩, 生成`foo.tar.gz`)

✚ 存档工具tar命令功能强大, 经常用于系统备份, 文章投稿。

gzip, gunzip, bzip2, bunzip2 (压缩和解压缩)

- ✚ `gzip file1` 压缩文件`file1`
- ✚ `Bzip2 file2` 压缩文件`file2`
- ✚ `gunzip file1.gz` 解压缩文件`file1.gz`
- ✚ `bunzip2 file2.bz2` 解压缩文件`file2`. 保证

进程管理

Linux是个多用户、多任务的操作系统。Linux操作系统将负责管理多个用户的请求和多个任务。用户运行一个程序，就会启动一个或多个进程。进程的启动有两种方式：**手工启动**和**调度启动**。

手工启动有前台启动和后台启动。前台启动是用户直接运行命令时并等待这个命令结束。前台进程的特点是进程不结束，用户不能再执行别的任务。用户在输入命令行后加上“&”字符，则启动后台。如

```
find / -name myfile -print > /root/test &
```

调度进程指用户事先设定好(如在某个时间)，让系统自行启动进程的方法。（使用crontab命令）

进程管理

查看系统进程： `ps [Options]`

✚ 选项：

a： 显示当前控制终端的进程 (包括其他用户的)。

u： 显示进程的用户名和启动时间等信息。

x： 显示没有控制终端的进程。

-e： 显示所有的进程。

✚ 例： `ps`或`ps x`查看系统中，属于自己的进程。

`ps au`查看系统中，所有用户的进程。

`ps aux`查看系统中，包含系统内部的及所有用户的进程。

✚ `ps`常和管道符 `|` 一起使用：

`ps -e | grep sshd` 查找`sshd` (SSH服务守护进程) 进程的信息，如进程号等。

进程管理

 **kill [Options] PID1 PID2 ...**


前台进程运行时，可用<Ctrl-c>来终止。对后台进程可以使用kill命令向进程发送强制终止信号来达到目的。

 **kill -9 PID** 强制终止进程号为PID的进程。

 **kill PID** 终止进程号为PID的进程。

 **killall -s "signal command"**

 **Killall** 可以根据进程名来发送信号。

 **killall -9 vim** 终止所有vim会话。

进程管理 (常用)

✚ top 实时监控进程状况, 系统资源使用和占用情况

✚ bg、jobs、fg

✚ 前台进程运行中时, <Ctrl-z>键可暂停进程的执行。

✚ bg命令用于把暂停的进程放到后台。

✚ 使用jobs命令可以看到在后台运行的进程。

✚ fg命令则可以把在后台运行的进程放到前台。

```
> du -a / | sort -rn > ~/log <按Ctrl-z>
[1]+  Stopped                  du -a / | sort -rn >~/log
> jobs
[1]+  Stopped                  du -a / | sort -rn >~/log
> bg %1
[1]+ du -a / | sort -rn >~/log &
> jobs
[1]+  Running                  du -a / | sort -rn >~/log &
> fg
```

网络相关命令

✚ `whoami` 和 `groups` `userid`
查看当前用户ID和组ID

✚ `finger`, `who` 和 `w`
查看当前在线用户

✚ `ssh -X user@host`
`ssh -Y user@host`

登陆服务器`host`, 并打开X11转发 (将远程机器显示图传到本地屏幕)

✚ `scp file1 user@host2:file2` (常用)
`scp -r user@host1:dir1 user@host2:dir2`
安全复制文件

也可用 `rsync` 文件同步

例子: `rsync -avz -e ssh --delete * webadmin@hnbcs.ihep.ac.cn:/www/html/`

其他常用工具和命令

- sleep 延迟指定时间: `sleep 1800` (延迟1800秒)
- bc 支持任意精度的交互式计算器:
 - 交互式输入: `bc -l`
 - 管道输入: `echo "scale=2;1/3" | bc`
- wget 从指定的URL下载文件 (支持断点续传)
- xargs 给其他命令传递参数的一个过滤器
- tee 输出到屏幕同时也输出到文件: **tee -a file (同时将输出追加到文件file中)**
 - > file 输出到file, >>追加到file (>和>> 看不到屏幕输出)**
- xpdf pdf 文档阅读器: `xpdf xx.pdf`
- latex 排版系统,生成很多具有书籍质量的文档(投稿, 毕业论文, 演示报告)。
- pdflatex 用来编译用LaTeX格式写的tex文件, 生成PDF文件。
- gv Postscript文档阅览器 `gv xx.ps; gv xx.eps`
- eog 图片查看器, 支持JPG, PNG, GIF, TIFF or XPM等图片格式
- display 图片浏览工具

修改文章用: `latexdiff draft1.tex draft2.tex > diff.tex`
`pdflatex diff.tex` 看2个tex文件的区别

文本编辑程序

vi 和 emacs

Vim

- ✚ `vi`是unix世界上最通用的全屏编辑器，linux中常用的是`vi`的加强版`vim`，`vim`同`vi`完全兼容，`vi`就是“visual interface”的缩写。它可以执行输出、删除、查找、替换、块操作等众多文本操作，而且用户可以根据自己的需要对其进行定制。
- ✚ `vi`有三种状态：普通模式，插入模式和可视模式。
- ✚ 普通模式：打开`vim`后首先进入命令模式。用户的任何输入都被当作命令来解释执行。
- ✚ 插入模式：用户输入直接呈现在文件内容中。普通模式下按“`i`”键进入插入模式；按 “`ESC`”键回到普通模式。
- ✚ 可视模式：按 “`v`” 键；按 “`ESC`”键返回普通模式。

vi的启动和退出

 **vi test.txt** 首先进入命令模式


```
~  
~  
~  
"test.txt" [New File] 0,0-1 All
```

 行首有波浪符号“~”，表示此行是空行。

 按“i”键，键入编辑模式。然后输入

```
Hello, this is a test. :)
```

 按Esc键，回到命令模式

 按“:”键进入命令行，键入“wq”保存并退出。 **(注意区分大小写)**

```
Hello, this is a test. :)  
~  
~  
:wq
```

vi命令

- ✚ vi的指令分为两种：长指令和短指令。
- ✚ 长指令以冒号开头，键入冒号后，在屏幕的最末尾一行会出现冒号提示符，等待用户键入指令，输入完指令后回车，vi就会执行该指令。
- ✚ 短指令和快捷键相似，键入短指令之后，vi不会给任何提示就直接执行。

vi常用命令

命令	说明
:	进入命令行
v	进入可视模式
i	在光标前输入文本
h, j, k, l	分别是光标左移, 下移, 上移, 右移 (一般来说, 按方向键就可以了)
x	删除光标对应的一个字符
dd	删除光标所在的一行
J	将下一行并入本行末尾
r字符	替换光标对应字符为新字符
p或P	在当前位置粘贴剪贴板的内容, p粘在光标所在字符后面, P粘在前面
数字G	一动光标到第若干行, 如果直接按G则移动到最后一行

命令	说明
:q	退出.
:q!	强行退出。(! 为强制执行)
:w [文件名]	存盘
:wq	存盘退出
:n [文件名]	新建文件(同时关闭当前文件), 如果暂时不指定文件名也行
:n,mw 文件名	该将第n~m行的文本保存到指定的文件filename中。
:help	查看帮助. 可以在help后面加某个帮助主题的名称, 如:help dd 和:help help

ref: <http://tnerual.eriogerg.free.fr/vimqrc.pdf>

vi一般命令

命令	说明
a	在光标后输入文本
A	在当前行末尾输入文本
i	在光标前输入文本
I	在当前行开始输入文本
o	在当前行后输入新一行
O	在当前行前输入新一行

命令	说明
B	在光标后输入文本
e	在当前行末尾输入文本
w	在光标前输入文本

命令	说明
dw	删除光标所在的单词
d\$ 或 D	删除光标至行尾的所有字符

命令	说明
r	替换光标所在的字符
R	替换字符序列
cw 或 ce	替换一个单词
cb	替换光标所在的前一字符
c\$ 或 C	替换自光标位置至行尾的所有字符
cc	替换当前行

vi命令

命令	说明
/abc	向后查询字串“abc”
?abc	向前查询字串“abc”
n	重复前一次查询
N	重复前一次查询，但方向相反
:起始行,结束行s/搜索串/替换串/g	从起始行到结束行,把所有的搜索串替换为替换串

命令	说明
u	取消上一次的操作
U	可以恢复对光标所在行的所有改变
J	把两行连接到一起
:set	用来设置或浏览vi系统当前的选项
:X	对所编辑的文件进行简单加密

命令	说明
y	将光标选定的部分拷入剪贴板
yw	将光标所在单词拷入剪贴板
y\$ 或 Y	将光标至行尾的字符拷入剪贴板
yy	将当前行拷入剪贴板
p	将剪贴板中的内容粘贴在光标后
P	将剪贴板中的内容粘贴在光标前

命令	说明
:split 文件名	切分出一个新窗口, 打开指定文件. 如果省略文件名, 则仍显示当前文件, 可用于同时观察文件的不同部分.(注意跟:n的区别)
<C-W>f	切分显示光标所指的文件名, VIM会在path中搜索该文件名, 比如常用它打开#include语句中的文件
<C-W><C-W>	当同时打开几个文件时, 按<C-W><C-W>在各窗口之间切换
<C-W>_	当同时打开几个文件时, 按<C-W>_使当前窗口最大化

vi命令

- ✚ 搜索字符串用的是正则表达式, 其中许多字符都有特殊含义:

\	取消后面所跟字符的特殊含义. 比如\[vim\]匹配字符串"[vim]"
[]	匹配其中之一. 比如[vim]匹配字母"v", "i"或者"m", [a-zA-Z]匹配任意字母
.	匹配任意字符
*	匹配前一字符大于等于零遍. 比如vi*m匹配"vm", "vim", "viim"....
\+	匹配前一字符大于等于一遍. 比如vi\+m匹配"vim", "viim", "viiim"....
\=	匹配前一字符零遍或者一遍. 比如vi\=m匹配"vm"或者"vim"
^	匹配行首. 例如/^hello查找出现在行首的单词hello
\$	匹配行末. 例如/hello\$查找出现在行末的单词hello
\()	括住某段正规表达式
\数字	重复匹配前面某段括住的表达式. 例如\(hello\).*\1匹配一个开始和末尾都是"hello", 中间是任意字符串的字符串

- ✚ 对于替换字符串, 可以用"&"代表整个搜索字符串, 或者用"\数字"代表搜索字符串中的某段括住的表达式.
- ✚ 举一个复杂的例子, 把文中的所有"abc.....xyz"字符串替换为"xyz.....abc"可以有如下写法:
:%s/abc\(.*\)xyz/xyz\1abc/g
:%s/\(abc\)\(.*\)\(xyz\)/\3\2\1/g
- ✚ 其它关于正则表达式搜索替换的更详细准确的说明请看"help".

vi命令

命令	说明
i<C-P>	向上搜索, 补全一个词. 例如, 上文中出现过filename这个词, 当你想再输入filename时, 只要按f<C-P>即可. 假如VIM向上搜索, 找到以f开头的第一个匹配不是filename, 你可以继续按<C-P>搜索下一个匹配进行补全. 当然, 如果你想一次<C-P>就成功, 你可以多输入几个字符比如filen再按<C-P>补全
i<C-N>	向下搜索, 补全一个词
i<C-X><C-L>	补全一行. 比如你写过一行for (int i=0;i<100;i++), 你想再写一模一样的一行, 只要输入for<C-X><C-L>即可. 如果补全出来的不是你想要的那一行, 你可以按<C-P>或<C-N>选择上一个或下一个匹配行
i<C-X><C-F>	在文件系统中搜索, 补全一个文件名

命令	说明
:!命令行	执行一条外部命令
i<C-Y>	把上一行对应列的字符抄下来
i<C-E>	把下一行对应列的字符抄上来
K	看光标所指标识符的man帮助页
:X	对当前文本文件加密

Vim的定制

✚ `vi`编辑器的行为可以通过设置编辑参数来定义，并且有许多种方法可以进行这种设置。最直接的方法是使用`vi`的`set`命令进行设置。这种情况下，`vi`在进行设置前必须处于指令状态。使用这种方法的用户可以设置任何选项，但是选项的改变是临时的，并且只在用户当前编辑会话下有效。当用户退出`vi`编辑器时，设置会被丢弃。

✚ 在UNIX版本的VIM中，这个文件一般可以放在用户的个人主目录下，文件名为“`.vimrc`”。VIM启动时将会把`.vimrc`文件中的每一行作为命令行依次执行，我们可以在该文件中加入若干命令，使VIM启动时自动开启一些有用的功能，定义一些常用的快捷键等。

以“.”(点)开头的文件被称为隐藏文件，用`ls -a`查看。

可以从系统目录copy: `cp /etc/vimrc ~/.vimrc` 然后编辑。

Vim的定制

✚ **设置vi环境:** `set`设置

✚ **缩写操作符:** `ab/unab`设置

`:ab lc linux course` 将lc指定为linux course的缩写

✚ **宏操作符:** 宏操作符 (`map/unmap`) 使用户能将一系列键绑定/解定给某一键。如同缩写操作符给用户一个文本输入模式下的捷径一样, `map`给用户一个在指令模式下的捷径。

`:map! <F10> <Esc>d di` 表示插入模式下按<F10>就相当于连续按了<Esc>d di, 这将会使VIM退回到普通模式, 删除一行, 再进入插入模式。

注意, 如果定义`:map d di`这将引起循环定义错误. 这时, 需要使用`:noremap d di`来定义。

Vim的定制

Set定制:

选项	缩写	功能
autoindent	ai	将新行与前一行的开始对准
ignorecase	ic	在搜索选项下，忽略大小
number	nu	显示行号
report	-	告知用户最后一个命令作用行的行号
shiftwidth	sw	设定缩进空格数，一般与autoindent一同使用
show mode	smd	在屏幕右角显示vi编辑器模式

宏操作符:

:nmap	键只对普通模式有效
:imap	键只对插入模式有效
:vmap	键只对可视模式有效
:cmap	键只在命令行下有效
:map	键在普通模式和可视模式都有效
:map!	键在插入模式和命令行下都有效

Vim的定制

下面是.vimrc文件的示例:

复制/home/cp/codes/linux/vimrc到个人目录下，并更名为.vimrc

```
" 设置在哪些模式下使用鼠标功能，mouse=a表示所有模式
set mouse=a
" 设置路径，在<C-W>f等命令中涉及此参数
" 打开光标的行列位置显示功能
set ruler
" 设置跳格距离
set tabstop=4
" 设置自动缩进格数
set shiftwidth=4
" 打开自动缩进功能
set smartindent
" 根据当前文件语法自动变色。VIM识别上百种文本文件的语法，如html, c++, java等
syntax on
"在普通模式下，映射F2键为删除一行
nmap <F2> dd
"编译代码文件
:map <F9> :w<CR>:!gcc -o %< %<CR>
"运行编译后的程序
:map <F5> :!./%<
```

Emacs

- ✚ 其缩写名称 `Editor MACroS` (编辑器宏指令)
Emacs Make All Computing Simple (Emacs使所有的计算简单)
- ✚ Emacs编辑器不仅仅是一个文本编辑器, 它被设计为一个全能的 (`do it all`) 的程序。由于该程序具有一种编程语言 (`Lisp`) 作为其基础, 因此Emacs用户可以对该编辑器进行自定义或进行功能扩展而继续使用它, 而不是使用现有的命令或创建一个新的多用途工具。自定义文件 `.emacs` 可以相互共享。
- ✚ 该软件一直是由Stallman维护, 他在emacs中内置了可以电子邮件、网络新闻、`shell`命令、编译程序并分析错误消息、运行并调试这些程序以及游戏等命令。最终该程序几乎无所不能, 以至于计算机用户几乎可以整天呆在emacs环境中而不出来, 用户在其中可以在窗口和窗口之间、文件和文件之间来回切换。
- 无模式编辑器: 可关闭/开启的专门功能以支持对文本、`TeX`、`Lisp`、`C/C++`和`Fortran`编程语言等的编辑

Emacs的启动和退出

- ✚ `emacs test.txt` 使用XWindow
- ✚ `emacs -nw test.txt` 无X模式
- ✚ 每次启动时，将从用户的主目录中加载`.emacs`的Lisp代码文件。
 - ✚ `-q` 忽略`.emacs`文件，启动时不加载它。这是一种跳过某个效果不好的`.emacs`文件的方式。
 - ✚ `-u userid` 从`userid`所指定的主目录中加载`.emacs`文件。
 - ✚ `.emacs`启动文件通常只考虑了键绑定和选项设置等内容，另外用户也可以直接在其中写入Lisp语句。
- ✚ 可直接进入文本编辑模式
- ✚ 保存：`<Ctrl-x><Ctrl-w>`
- ✚ 退出：`<Ctrl-x><Ctrl-c>`

Emacs基本命令和按键组合

光标位置

缓存区是否已经改变
窗口当前所查看的缓存区名称
当前何种主模式
光标所在行数
窗口当前位置距离缓存区底部多远

模式行

回显区

```
log - emacs@gucasfarm0.ihep.ac.cn
File Edit Options Buffers Tools Help

Total 224
drwxr-xr-x 5 caogf stubes 4096 Nov 6 08:33 caogf
drwxr-xr-x 21 dusx stubes 4096 Dec 23 14:29 dusx
drwxr-xr-x 20 Huangb stubes 4096 Nov 26 14:44 Huangb
drwxr-xr-x 24 hujf stubes 4096 Feb 20 09:13 hujf
drwxr-xr-x 16 jixb stubes
drwxr-xr-x 7 lil stubes
drwxr-xr-x 16 liubj stubes
drwxr-xr-x 4 liucx stubes
drwxr-xr-x 8 liuy stubes
drwx----- 2 root root
drwxr-xr-x 17 lvqw stubes
drwxr-xr-x 9 mcl stubes
drwxr-xr-x 4 mjni stubes
drwxr-xr-x 12 ningfp stubes
drwxr-xr-x 6 offline stubes
drwxr-xr-x 12 shizf stubes
drwxr-xr-x 10 sunss stubes
drwxr-xr-x 36 sunxd stubes
drwxr-xr-x 9 tianhl stubes
drwxr-xr-x 9 wangj stubes
drwxr-xr-x 13 wangzy stubes
drwxr-xr-x 13 wulh stubes
drwx----- 20 xiaorui xiaorui 4096 Dec 24 17:18 wulh
drwxr-xr-x 22 xieyg stubes 4096 Feb 13 10:21 xiaorui
drwxr-xr-x 7 xugf stubes 4096 Jan 28 00:25 xieyg
drwxr-xr-x 22 zhaohs stubes 4096 Oct 13 11:21 xugf
drwxr-xr-x 7 zhengyh zhengyh 4096 Feb 20 08:16 zhaohs
drwxr-xr-x 7 zhengyh zhengyh 4096 Sep 6 22:25 zhengyh

--- log (Fundamental) --L1--All-----
For information about the GNU Project and its goals, type C-h C-p.
```

Emacs基本命令和按键组合

C - b = 按住Ctrl键，再按下字母b

动 作

M - v = 按住Alt键，再按下字母v

键 组 合

放弃当前操作

C - g

光标后移

C - b

光标下移

C - n

光标前移

C - f

光标上移

C - p

删除字符

C - d

删除行

C - k

删除单词

M - d

到文件头

M - <

Emacs基本命令和按键组合

C - b = 按住Ctrl键，再按下字母b

动 作

M - v = 按住Alt键，再按下字母v

键 组 合

到行的开始

C - a

到文件尾

M - >

到行的结束

C - e

帮助

C - h

打开文件

C - x C - f

向后翻页

C - v

向前翻页

M - v

退出

C - x C - e

文件换名存储

C - x C - w

保存文件

C - x C - s

教程

C - h t

撤消前一个操作(Undo)

C - _ 或者 C - x, u

ref: <http://moss.csc.ncsu.edu/~mueller/EmacsReferenceCard.pdf>

Emacs光标移动命令

键	操 作
CONTROL-F	以字符为单位向前移动光标
CONTROL-B	以字符为单位向后移动光标
META-f	以单词为单位向前移动光标
META-b	以单词为单位向后移动光标
META-e	移动光标到语句结尾
META-a	移动光标到语句开始
META->	向前移动光标到缓存区结尾
META-<	向前移动光标到缓存区开始
CONTROL-ESCAPE	移动光标到行尾
CONTROL-A	移动光标到行首
CONTROL-N	以行为单位向下移动光标
CONTROL-P	以行为单位向上移动光标
CONTROL-L	清除并重新绘制屏幕，滚动当前行到窗口中间
META-r	移动光标到中间行的开始

Emacs编辑搜索命令

键	操 作
CONTROL-D	删除光标下的字符
DELETE	删除光标左边的字符
META-d	向前杀死从光标位置到当前单词结尾的字符
META-DELETE	向后杀死从光标位置到上一个单词开始的字符
META-k	向前杀死到某个语句结尾的字符
CONTROL-X DELETE	向后杀死到某个语句开始的字符
CONTROL-K	向前杀死从光标到以NEWLINE结尾的行之间的文本(不包括NEWLINE), 如果光标和NEWLINE之间没有文本, 将杀死NEWLINE
CONTROL-S	提示用户对某个字符串进行增量搜索, 并向前搜索一个匹配项
CONTROL-S RETURN	提示用户输入一个完整的字符串, 并向前搜索一个匹配项
CONTROL-R	提示用户对某个字符串进行增量搜索, 并向后搜索一个匹配项
CONTROL-R RETURN	提示用户输入一个完整的字符串, 并向后搜索一个匹配项
META-CONTROL-S	提示用户输入一个正则表达式进行增量搜索, 并向前搜索一个匹配项
META-CONTROL-S RETURN	提示用户输入一个完整的正则表达式, 并向前搜索一个匹配项
META-x isearch-backward-regexp	提示用户输入一个正则表达式进行增量搜索, 并向后搜索一个匹配项
META-x isearch-backward-regexp RETURN	提示用户输入一个完整的正则表达式, 并向后搜索一个匹配项

Emacs窗口操作命令

键	操 作
CONTROL-X b	切换窗口所查看的缓存区
CONTROL-X 2	将当前窗口垂直拆分为两个
CONTROL-X 3	将当前窗口水平拆分为两个
CONTROL-X o	选择其他窗口
META-CONTROL-V	滚动其他窗口
CONTROL-X 4b	提示用户输入一个缓存区名，并在其他窗口选择它
CONTROL-X 4f	提示用户输入一个文件名，并在其他窗口选择它
CONTROL-X 0 (零)	删除当前窗口
CONTROL-X 1	删除除当前窗口之外的所有窗口
META-x shrink-window	使当前窗口缩短一行
CONTROL-X ^	使当前窗口变长一行
CONTROL-X }	使当前窗口变宽一个字符
CONTROL-X {	使当前窗口变窄一个字符

Emacs替换命令

键	操 作
META-x replace-string	提示用户输入“字符串”和“新字符串”。使用“新字符串”替换每一个“字符串”。在该命令的开始设置标志
META-%或META-x query-replace	和以上一样，但在替换“字符串”之前将询问用户是否确认。参见以下的响应表
META-x replace-regexp	提示用户输入一个正则表达式和“新字符串”。使用“新字符串”替换每一个正则表达式。在该命令的开始设置标志
RETURN	不再进行替换，退出
SPACE	进行此次替换并继续
DELETE	不进行此次替换，跳过它并继续
, (逗号)	进行此次替换，显示替换结果并等待其他的命令，任何命令都是合法的，除了DELETE作为SPACE命令来对待，并且不能取消修改
. (句点)	进行此次替换并退出搜索
! (惊叹号)	不再询问用户，无条件地替换所有剩余的相应字符串

.emacs文件

复制/home/cp/codes/linux/emacs到个人目录下，
并更名为.emacs

```
(set-default-font "9x15")
;; ===== Set colors =====
(set-cursor-color "red")
;; Set region background color
(set-face-background 'region "blue")
;; Set emacs background color
(set-background-color "white")
;; Set foreground color
(set-foreground-color "black")
;; Show line-number in the mode line
(line-number-mode 1)
;; Show column-number in the mode line
(column-number-mode 1)
;; ===== Line by line scrolling =====
(setq scroll-step 1)
;; ===== F5 to evoke speed bar =====
(global-set-key [(f5)] 'speedbar)
;; ===== Set the highlight current line minor mode =====
(global-hl-line-mode 1)
;; ===== Set standard indent to 2 rather than 4 =====
(setq standard-indent 2)
```

脚本编程语言

Scripting Language: UNIX shell, php, Perl ...

- ✓ 为缩短编写-链接-运行过程而创建的编程语言
- ✓ 数据没有类型
- ✓ 解释式，非编译
- ✓ 开发速度快，易用性好

System Programming Language: fortran, C/C++, Java ...

- ✓ 从头设计数据结构和算法
- ✓ 数据具有类型
- ✓ 编译式
- ✓ 复杂的功能和算法，运行速度快

shell编程

✚ shell是一个命令语言解释器，它拥有自己内建的shell命令集，shell也能被系统中其他应用程序调用。用户在提示符下输入的命令都由shell解释后传给Linux核心。

✚ Linux中的shell有多种类型，其中最常见的：

✚ Bourne shell (sh)

✚ C shell (csh)

✚ Korn shell (ksh)

✚ Turbo C shell: tcsh

✚ Bourne Again shell: bash (*Linux的标准shell*)

我们主要介绍**Bash**编程。

Bash

特性

- 命令行编辑模式：更容易回馈和定位错误或修改以前的命令
- 作业控制：同时停止，启动和暂停任意数目的命令
- 更多用于定制的新选项和变量，可编程特性已扩展到函数定义，更多的控制结构，整数运算，高级I/O控制等。

参看与更改shell:

- `echo $SHELL`
- `chsh`

文件名、通配符和路径名扩展

- 基本通配符

- ✓ `?` 匹配任意单个字符; `*` 匹配任意字符串
- ✓ `[set]` `set`中任意字符
- ✓ `[!set]` 不在`set`中的任意字符

<code>[abc]</code>	a,b或c
<code>[a-c]</code>	a,b或c
<code>[a-z]</code>	所有小写字母
<code>[!0-9]</code>	所有非数字
<code>[0-9!]</code>	所有数字和惊叹号

- 大括号扩展: 可选前缀 + `{ }` + 可选后缀

- ✓ `echo b{ed,olt,ar}s` 显示 `beds`, `bolts`和`bars`
- ✓ `ls *. {c,h,o}` 打印源文件, 头文件和目标文件

输入与输出

数据过滤器

- 标准I/O:
 - 0:标准输入stdin:键盘
 - 1:标准输出stdout:屏幕(默认)
 - 2:标准错误输出stderr:屏幕
- I/O重定向
 - 重定向输出符号 (>)
 - 追加输出符号 (>>)
 - 重定向标准输入 (<)
 - 管道 (|)

命令	功能
cat	将输入复制到输出
grep	在输入中检索字符串
sort	在输入中将行排序
cut	在输入中抽取列
sed	对输入执行编辑操作
tr	将输入字符转换成其他字符

🚩 /dev/dull 为空设备，可将不想看见的输出重定向到此。

```
ls -l readme.txt 1> /dev/null    若readme.txt存在，无显示；    不存在，显示错误。
ls -l readme.txt 2> /dev/null    若readme.txt存在，显示文件；    不存在，不显示错误。
ls -l readme.txt  > /dev/null 2>&1  不管readme.txt是否存在，都无任何显示。
```

这里， 2>&1 表明 2等同于1

命令行编辑

- 提供`emacs`和`vi`编辑模式。`emacs`为默认模式
- 选择编辑模式: `set -o emacs (vi)`
- 将流行编辑器的键盘习惯转到`shell`
- 历史文件: `.bash_history`
- 参见`emacs`编辑命令
- 主要功能键:
 - ✓ `Ctrl+a` 移动到行首 `Ctrl+e` 移动到行尾
 - ✓ `Ctrl+d` 删除字符 `Ctrl+k` 删除至行尾字符
 - ✓ `Ctrl+r` 搜索历史命令 `Tab` 文字补全

定制用户环境

- 特殊文件：
 - ✓ `.bash_profile`: 登录shell 时执行
 - ✓ `.bashrc`: 开启子shell时执行
 - ✓ `.bash_logout` 退出shell时执行
- 别名: 用户对命令或命令字符串所定义的同义词
 - ✓ `alias name='comand line'`
`alias rm='rm -i'` (将rm命令加确认选项, 以免误删)
- 选项: `set - (+) o [optionname]`
 - ✓ `-`: 打开 `+`: 关闭
 - 如: `set -o vi` (让bash模拟vi的操作。默认的是模拟emacs)
- 变量: `varname=value`
 - ✓ 使用变量: `$varname` 或 `${varname}`
 - 如: `g139="gpu@192.168.237.139:/home/gpu/GPU05/"`

变量

- 使用双引号不能影响\$替代
- 单引号或反斜杠防止\$替代
- `unset varname`
- 使用`export`设置环境变量：
使变量被其子shell使用
- 内置环境变量：

```
> person=lvxr001
> echo $person
lvxr001
> echo "$person$"
lvxr001$
> echo '$person'
$person
> echo \ $person
$person
> unset person
> echo "name $person"
name
```

常用环境变量	功能
HOME	用户的主目录
PATH	执行命令时所搜索的目录
PS1	在shell命令行的提示符
HISTSIZE	命令历史文件最大行数

echo -e (将转义后的内容输出):

```
Txt=`echo -e "$Txt\nrm -f aa.bosserr"`
```

```
Txt=`echo -e "$Txt\nrm -f aa.bosslog"`
```

```
echo "$Txt" > Run.txt
```

将所有要输出的内容放在\$Txt, 最后将\$Txt的内容保存到 Run.txt (\n为换行)

输出带颜色的字符:

```
echo -e "G=$iii R=$kkk B=\033[1;5;31m$jjj\033[0m" (用红色, 闪烁, 显示数字)。
```

echo -n 表示不换行输出 (用于程序运行状态条等)

创建shell脚本

- shell脚本是指使用用户环境shell提供的语法所编写的脚本。
- 显示用户信息例程：

showinfo.sh

```
#!/bin/bash
#This script is a test!
echo -n "Date and time is:"
date
echo "The Executable path is : $PATH "
echo "Your name is : `whoami` "
echo -n "Your Current directory is : "
pwd
#end
```

- 首行#! 指定运行的shell; echo -n 不换行输出。
- 以#开头的行为脚本中的注释行，不执行命令操作
- 运行前用 **chmod u+x showinfo.sh** 使自己有运行权限。

执行shell脚本

- 运行方式:

- ✓ `> source scriptname` 或 `./scriptname`
- ✓ `> sh scriptname`
- ✓ 直接执行脚本: `> path/scriptname`
需要可执行权限 `> chmod +x scriptname`

```
> ./showinfo.sh
Date and time is: Thu Mar  2 01:06:36 CST 2017
The Executable path is :
/usr/lib64/mpi/gcc/openmpi/bin:/home/dongly/bin:/usr/local/bin:/usr/bin
:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/usr/lib/mit/bin:/usr/lib/
mit/sbin
Your name is : dongly
Your Current directory is : /home/dongly
```

- 输入命令时: 命令分隔符“;”使系统按顺序执行这些命令
- 使用“\”分行输入一个长命令

shell变量

- 命令行参数：
IFS定义SHELL分隔符，默认为空格
- set命令：改变参数字符串
- shift命令：
命令行参数向前移动一格
- read命令：
从键盘读入变量
- 命令替换
\$(shell command)
`shell command`

变量	功能
\$#	存储命令行参数的个数
\$?	存储上一个命令的返回值
\$0	存储Shell程序的程序名
\$n	第n个命令行参数
\$*	存储Shell程序的所有参数
@	存储所有命令行输入的参数
\$\$	存储Shell程序的PID
!	存储上一个后台命令的PID

条件测试

- 语法:

`test expression`

或

`[expression]`

(注意: `expression`两边有空格)

- 字符串比较操作符
- 数值比较操作符
- 文件属性操作符
- 逻辑操作符

字符串比较操作符

- 用来判断字符串表达式的真假

<code>str1 = str2</code>	如果str1和str2相同, 则为真。
<code>str1 != str2</code>	如果str1和str2不相同, 则为真。
<code>str1 < str2</code>	如果str1小于str2, 则为真。
<code>str1 > str2</code>	如果str1大于str2, 则为真。
<code>-n str</code>	如果str的长度大于零, 则为真。
<code>-z str</code>	如果str的长度等于零, 则为真。

注意: 比较操作符两边要有空格

数值比较操作符

- 用来判断数值表达式的真假

<code>int1 -eq int2</code>	如果 <code>int1 = int2</code> ，则为真。
<code>int1 -ge int2</code>	如果 <code>int1 >= int2</code> ，则为真。
<code>int1 -gt int2</code>	如果 <code>int1 > int2</code> ，则为真。
<code>int1 -le int2</code>	如果 <code>int1 <= int2</code> ，则为真。
<code>int1 -lt int2</code>	如果 <code>int1 < int2</code> ，则为真。
<code>int1 -ne int2</code>	如果 <code>int1 != int2</code> ，则为真。

```
> [ 100 -gt 200 ]  
> echo $?  
1 < 其表达式为假，返回值1 >
```

文件属性操作符

- 用来判断文件是否存在、文件类型及属性

<code>-d filename</code>	如果filename为目录，则为真。
<code>-e filename</code>	如果filename存在，则为真。
<code>-f filename</code>	如果filename为普通文件，则为真。
<code>-r filename</code>	如果filename为只读，则为真。
<code>-s filename</code>	如果filename长度大于零，则为真。
<code>-w filename</code>	如果filename为可写，则为真。
<code>-x filename</code>	如果filename为可执行，则为真。
<code>file1 -nt file2</code>	如果file1比file2新，则为真。
<code>file1 -ot file2</code>	如果file1比file2旧，则为真。

逻辑操作符

- 用来结合表达式或取得表达式相反值

`! expr` 取反。

`expr1 -a expr2` 与关系

`expr1 -o expr2` 或关系

`statement1 && statement2`

执行statement1, 如果成功, 则运行statement2

`statement1 || statement2`

执行statement1, 如果失败, 则运行statement2

`\(expr \)` 逻辑分组

流程控制if/else

```
if expr1
then
    commands
elif expr2
then
    commands
else
    commands
fi
```

- ✚ **最简形式：**
没有elif和else
- ✚ **返回状态：** 0为**无错退出状态**；其他（1~255）通常表示**错误**。
- ✚ **返回：** 脚本默认返回最后一个命令的状态、
 - ✚ **return N：**
 主要用在函数中
 - ✚ **exit N：**
 退出整个脚本

if/else示例

- 在键盘上读取一个字符，然后判断字符的值

`read_character.sh`

```
#!/bin/bash
#This is the example about if/else.
echo -n "Please input the answer: "
read ans
echo You input is $ans
if [ $ans = y ]
then
    echo "The answer is yes."
elif [ $ans = n ]
then
    echo "The answer is no."
else
    echo "Bad Input."
fi
#end of the example
```

键盘一次读入变量时，用语句`read var1 var2`。变量之间用空格隔开。输入文本过长，shell将所有的超长的部分赋给最后一个变量。

流程控制case

✚ 用来从很多测试条件中选择符合的条件执行。

```
case string in                //测试string字符串
    str1)                    //若str1符合
        commands;;          //则执行这些命令
    str2)
        commands;;
    ...
esac                          //结束case语句，是case英文反写。
```

✚ 可选的str可以有管道字符|分隔几个模式组成。

case示例

- 使用case语句建立一个菜单

✓ check_option.sh

```
#!/bin/bash
#display an option menu
echo "-----"
echo "1 Restore"
echo "2 Backup"
echo "3 Upload"
echo
#read and execute the user's selection
echo -n "Enter Choice: "
read CHOICE
case "$CHOICE" in
    1|R) echo "1 Restore";;
    2|B) echo "2 Backup";;
    3|U) echo "3 Upload" ;;
    *)   echo "sorry $CHOICE is not a valid choice! "
        exit 1 ;;
esac
#end
```

流程控制for

如果in list省略，默认列表为“\$@”

```
for name [in list]
do
    commands
done
```

将参数中的文件内容转换成大写字母，然后另存为“.caps”的同名文件中：capitalize.sh

```
#!/bin/bash
#This is example for capitalization.
[ $# -eq 0 ] && echo "no file to be processed!" && exit 1
for file in "$@"
do
    tr a-z A-Z < $file >> file.caps
done
#end
```

*tr*命令将第一个参数的字母范围对应到第二个参数的范围。

tr 将输入字符转换成其他字符，**\$#** 存储命令行参数的个数；**\$@**存储所有命令行输入的参数；

流程控制while和until

- while: 测试条件为真时循环执行
- until: 测试条件为真时循环执行

```
while condition
do
    commands
done
```

```
until condition
do
    commands
done
```

- 例程：重复尝试拷贝文件：copy.sh

```
#!/bin/bash
#This is example about copy
[ $# -lt 2 ] && exit 1
while ! cp $1 $2
do
    echo 'Attempt to copy failed. waiting...'
    sleep 5
done
```

流程控制while例程

- 实现计算1 ~ 5的平方：

square.sh

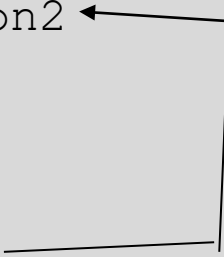
```
#!/bin/bash
#This is example about while.
int=1
while [ $int -le 5 ]
do
    sq=`expr $int \* $int`
    echo $sq
    int=`expr $int + 1`
done
echo "finished"
#end
```

expr为内部命令，实现数值计算功能。

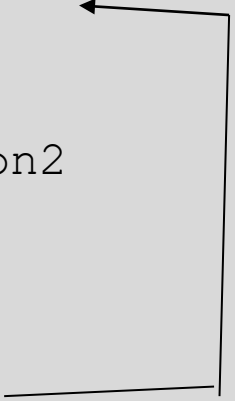
流程控制break和continue

- `break`语句用于中断循环，进入循环结束后的下一个命令
- `continue`语句则忽略之后的命令，直接进入下一步循环

```
while expression1
do
    while expression2
    do
        ...
        continue
        ...
    done
done
```



```
while expression1
do
    while expression2
    do
        ...
        break
        ...
    done
done
```



本地文档<<

- 使用户将输入重新定向到shell脚本或者从shell脚本本身导出
- 格式:

```
command << EndOfFile  
    . . .  
EndOfFile
```

- 分隔符之间的内容都发送给进程作为标准输入
- 对脚本执行需要交互的程序命令很有用

```
> ./check_character.sh << EOF  
> y  
> EOF  
Please input the answer: The answer is yes.
```

shell函数

- 功能：
 - ✓ shell函数存于内存中，运行速度快
 - ✓ 使脚本更小、更易于维护，结构更严谨
 - ✓ 格式：

```
funcname ()  
{  
    shell commands  
}
```

```
function funcname  
{  
    shell commands  
}
```

- 直接调用函数名。函数处理参数的方式与脚本文件类似
- 函数中的变量改变也作用到函数外
- local变量：局部变量

函数例程

- `ascript.sh`

```
#!/bin/bash
#This is example for function
function afunc
{
    echo in function: $0 $1 $2
    var1="var1 in function"
    var2="var2 in function"
    echo var1: $var1
    echo var2: $var2
}
```

```
var1="outside function"
echo var1: $var1
echo $0: $1 $2
afunc funcarg1 funcarg2
echo var1: $var1
echo $0: $1 $2
echo var2: $var2
#end
```

- ✓ `$0`为脚本名
- ✓ 函数中的变量操作也作用在函数外部
- ✓ 可以使用`local`变量限制作用域

```
> ./ascript.sh arg1 arg2
var1: outside function
./ascript.sh: arg1 arg2
in function: ./ascript.sh funcarg1 funcarg2
var1: var1 in function
var2: var2 in function
var1: var1 in function
./ascript.sh: arg1 arg2
var2: var2 in function
```

优先级次序

- shell命令资源的优先级次序：
 - 别名
 - 关键字
 - 函数
 - 内置命令
 - 脚本和可执行程序：按PATH环境变量搜索

awk

- awk是一种程序语言，对文档资料的处理具有很强的功能。awk来自于三个最初设计者的名字：Aho、Weinberger和Kernighan。它借鉴了C程序中的流程控制和语法。
- awk的主要功能是针对文件的每一行（记录），搜寻指定的格式，执行指定的操作，直到文件结束。它经常用在如下的几个方面：
 - 根据要求选择某几行，几列或部分字段以供显示输出。
 - 分析文档中的某一个字出现的频率、位置等。
 - 根据某一个文档的信息准备格式化输出。
 - 以一个功能十分强大的方式过滤输出文档。
 - 根据文档中的数值进行计算。

awk

- 记录：数据文件的一行数据
- 域：记录上被分割开的子字符串。一般以空白字符作为分隔符。'-F'选项可指定分隔符。\$1, \$2, \$3 ... \$n来标识不同域。\$0为所有域。
- 执行awk程序：
 - ✓ `awk 'program' input-file1 ...`
 - ✓ 如： `awk '{print $2}' test1.dat`
 - ✓ 可将单引号内的部分放入一个文件，用-f指定该程序文件执行
 - ✓ `awk -f program-file... input-file ...`

awk

- awk语言命令由两部分组成：表达式 {动作}
 - ✓ 表达式₁ {动作₁}
 - ✓
 - ✓ 表达式_n {动作_n}
- 关系运算符：
>, <, >=, <=, ==, !=, ~, !~
- I/O指令: print, printf(), getline ...
- 流控制指令: if/else, while...
- 标准awk程序包括三部分: BEGIN{ }, { }, END{ }
- 内部变量: \$n, NF, NR, FS, OFS...
- 内部函数: index, length, sub, substr...

awk

如数据文件: student.dat

Tom	116001	FuTian	M	90
John	116005	NanShan	M	85
Mary	116018	LuoHu	W	65
Steven	116030	YanTian	M	78

```
> awk '$0 ~ /LuoHu/' student.dat
```

```
Mary      116018      LuoHu      W      65
```

```
> awk '$0 !~ /LuoHu/' student.dat
```

```
Tom      116001      FuTian      M      90
```

```
John      116005      NanShan      M      85
```

```
Steven    116030      YanTian      M      78
```

```
> awk '{if ($5 > 80) print $1}' student.dat
```

```
Tom
```

```
John
```

awk

```
> awk '{if ($3=="FuTian"&& $4=="M") print $0}' student.dat
Tom          116001  FuTian          M          90
> awk '/^...v/' student.dat
Steven       116030   YanTian          M          78
> awk 'NF>0' student.dat 打印非空白行
> awk 'NR % 2 == 0' student.dat
John         116005           NanShan          M          85
Steven       116030           YanTian          M          78
> awk '{nlines++} END{print nlines}' student.dat
4
> awk 'END{print NR}' student.dat
4
```

awk 应用实例:

删除队列中正在等待运行的作业:

```
hep_q -u | grep I | awk -F ' ' '{print $1}' | xargs hep_rm
```

```
[dongly@lxslc607 ~]$ hep_q -u | more
```

JOBID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
55284338.0	dongly	03/04 02:53	0+18:57:48	R	0	2441.4	xboss sim_dd_001.txt
55284339.0	dongly	03/04 02:53	0+18:57:48	R	0	2441.4	xboss sim_dd_002.txt
55284340.0	dongly	03/04 02:53	0+18:57:48	R	0	2441.4	xboss sim_dd_003.txt
55284341.0	dongly	03/04 02:53	0+18:57:48	R	0	2441.4	xboss sim_dd_004.txt
55284342.0	dongly	03/04 02:53	0+18:57:48	R	0	2441.4	xboss sim_dd_005.txt
55284343.0	dongly	03/04 02:53	0+18:57:48	R	0	2441.4	xboss sim_dd_006.txt
55284344.0	dongly	03/04 02:53	0+18:57:48	R	0	2441.4	xboss sim_dd_007.txt
55284345.0	dongly	03/04 02:53	0+18:57:48	R	0	2441.4	xboss sim_dd_008.txt
55284346.0	dongly	03/04 02:53	0+18:57:48	R	0	2441.4	xboss sim_dd_009.txt

说明: hep_q -u 显示账号所有的作业, 第一列为作业JOBID。
hep_rm JOBID 删除JOBID的作业。

Bash中用awk读取文件的数值进行简单数据处理:

```
#!/bin/bash
declare -a modes
modes[1]='mode400'
modes[2]='mode401'
for ((IDY=1;IDY<=2;IDY++));
do
    SNsig=0
    Schi2ndf=0
    K=0
    mode=${modes[${IDY}]}
    for ((IDX=11;IDX<=38;IDX++));
    do
        Nsig=`grep $mode ${IDX}/*rep | awk -F ' ' '{print $3}'`
        chi2ndf=`grep $mode ${IDX}/*rep | awk -F ' ' '{print $7}'`
        let "SNsig=SNsig+Nsig"
        Schi2ndf=$(echo "scale=2;$Schi2ndf+$chi2ndf"|bc)
        let "K=K+1"
    done
    let "SNsig=SNsig/K"
    Schi2ndf=$(echo "scale=2;$Schi2ndf/$K"|bc)
    echo "$mode ${SNsig} ${Schi2ndf}"
done
```

整数计算用let即可

浮点数计算需要调用 bc 来进行
scale=2, 设置精度为小数点2位

运行结果:

```
[dongly@lxs1c607 ana]$ ./ct1.sh
mode400 31192 1.50
mode401 136983 1.13
```

sed

- a "Stream Editor"
- sed作为重要的Unix工具，可以完成文本的批处理和转换，减少重复的编辑工作。它逐行读入输入文件，并通过执行sed命令或者脚本的操作，然后处理后的结果输出到标准输出。
- sed非常适合于以下情况：
 - ✓ 对于超大的文件，交互编辑器无法正常打开编辑
 - ✓ 对于操作特别繁杂的文件编辑工作
 - ✓ 某些文本整体上适用的修改

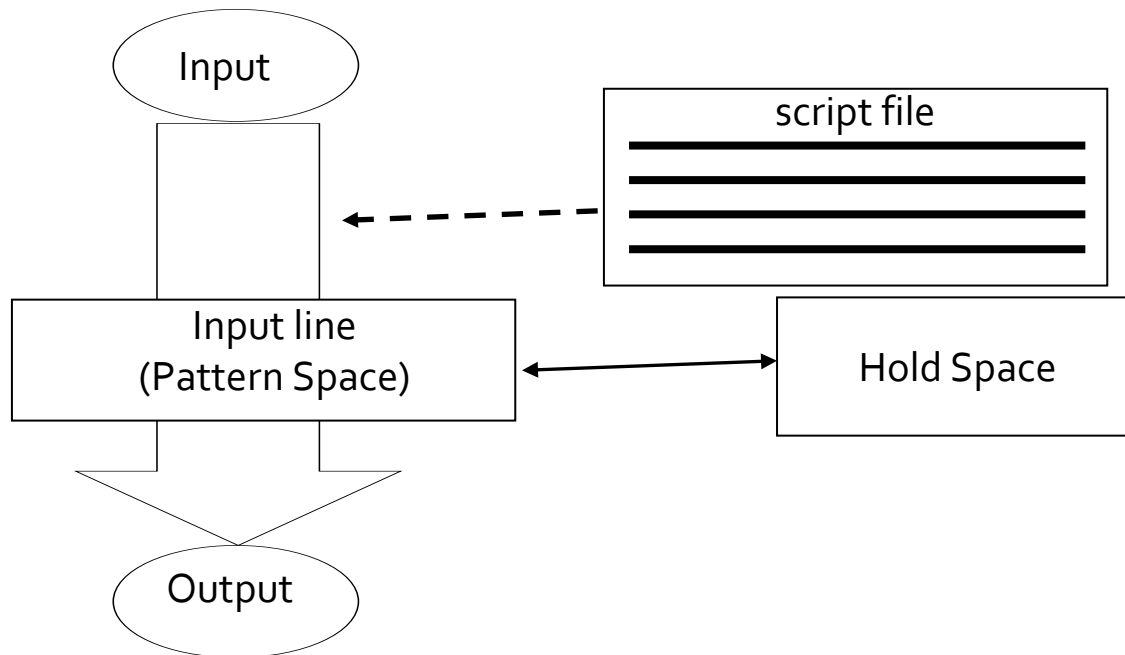
ref: <http://www.grymoire.com/Unix/Sed.html>

sed

- `sed [-e script] [-f script-file] [-n] [files...]`
 - ✓ `-e` 命令行模式脚本
 - ✓ `-n` 默认为显示编辑结果；`-n`设置为不显示，除非脚本内指定
 - ✓ `-f` 从文件中读入sed脚本
 - ✓ `files` 如果不指定文件，sed从标准输入中读入字符串
 - ✓ 如果sed脚本文件开头有“`#n`”，sed将打开`-n`选项
- 启动sed的方法：
 - ✓ `sed -e 'command;command;command' input_file`
 - ✓ `sed -e 'command;command;command' input_file > output_file`
 - ✓ `... | sed -e 'command;command;command' | ...` 使用管道
 - ✓ `sed -f sedcommands input_file > output_file` 使用sed脚本并重定向标准输出到文件

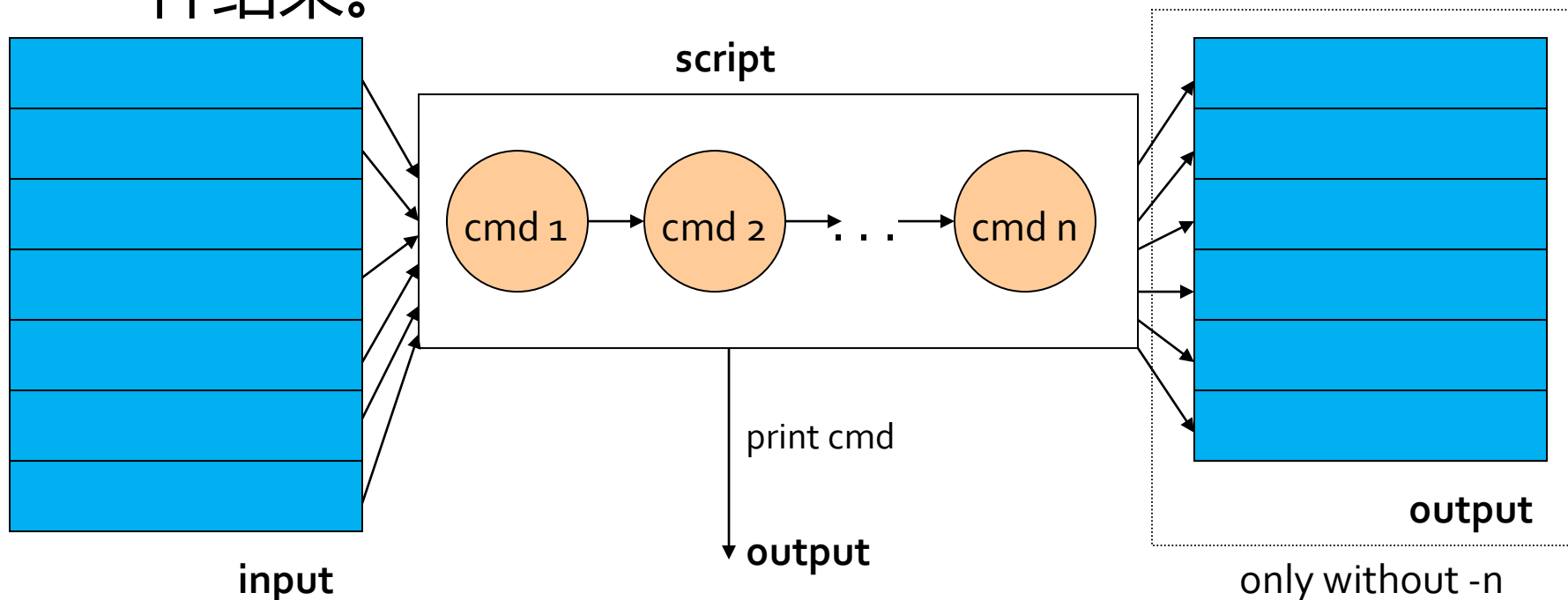
sed

- 所有的编辑命令按照顺序执行
- 如果前一个命令改变了输入的内容，那么接下来的命令将处理pattern space改变后的内容，而不是最先的内容
- sed不改变源输入的内容，只将编辑结果存在pattern space中，完成编辑后输出到标准输出（可以通过重定向储存到文件中）



sed

- `sed`命令由地址和操作组成；地址可以是正则表达式匹配，或是行数。
- 输入的每行，按顺序逐个通过每个命令，直到文件结束。



sed

- 行地址：
 - ✓ N 第N行, N=1,2,...
 - ✓ \$ 最有一行
 - ✓ i,j 第i到j行

sed '53!d' prints only line number 53

sed -n '4,9p' prints only lines 4 through 9
- 匹配字符串或正则表达式地址：

sed '/^\$/d' will delete all empty lines

sed '/./,\$!d' will delete all leading blank lines at the top of file

sed -n '/test/{p;d}' will print the deleted line which has characters “test”
- 命令：

a--append; c--change; d--delete; i--insert; p--print;

s--substitute; r--read; w--write; y--transform;

=--display line num;

N--append the next line to the current one;

q--quit

sed

- Examples:

Append text following each line matched by address.

```
> cat <an html file> | sed -e '/<body/a Hello  
World '
```

Delete line(s) from pattern space.

```
> cat homework.txt | sed -e '/[Hh]omework/d'> homework.new
```

```
> cat homework.txt | sed -e '1,20d' > homework2.new
```

Print the addressed line(s).

```
> sed -n '/regexp/, $p' somefile
```

Delete ONLY the regular expression and not the entire line

```
> sed -e 's/ <a bad word>//g' somefile > censored_file
```

```
> sed -e 's/ <a bad word>//2g' somefile > censored_file
```

举例：一个BESIII用sed自动生成作业脚本的的shell程序：

```
> cat subjob
#!/bin/sh
SEED=16000

case $1 in
sim|rec)

    DIR=`pwd`
    FILE_NAME=$1
    SAMPLE_NAME="sample_"$FILE_NAME".inp"

    INPUT=1
    until [ $INPUT -gt 400 ]
    do

        if ((INPUT<10))
        then
            IDX="00"$INPUT
        elif ((INPUT>99))
        then
            IDX=$INPUT
        else
            IDX="0"$INPUT
        fi
```

```
        TXT_NAME=$FILE_NAME_"$IDX".txt"
        rm -f $TXT_NAME
        cp -f $SAMPLE_NAME $TXT_NAME

        sed -i "s@xxxx@$SEED@g" $TXT_NAME
        sed -i "s@yyyy@$DIR@g" $TXT_NAME
        sed -i "s@www@$IDX@g" $TXT_NAME

        # submit job
        boss.condor $TXT_NAME

        INPUT=$((INPUT+1))
        SEED=$((SEED+10))
    done

;;

*)
echo "Usage: ./subjob sim for simulation
|| ./subjob rec for reconstruction"
;;

esac
```

事先准备好sample_sim.inp和sample_rec.inp文件，将需要改变的3个位置写为xxxx, yyyy和www，运行./subjob sim(或rec)生成400个模拟（或重建）的作业脚本。

举例：一个BESIII拆分作业脚本的python程序：

```
➤ cat p.py
import sys
if len(sys.argv)==2:
    job = sys.argv[1]
    inFile = "job_"+job+".txt"
    num = 1
elif len(sys.argv)==3:
    job = sys.argv[1]
    inFile = "job_"+job+".txt"
    num = int(sys.argv[2])
else:
    sys.exit(0)

with open(inFile,"r") as f:
    lines=f.readlines()

numDst=sum([line.count('.dst') for line in lines])
if numDst%num == 0:
    numOutFile=numDst/num
else:
    numOutFile=int(numDst/num)+1
if numOutFile==1:
    sys.exit(0)

for index in range(1,numOutFile+1):
    ind=0
    outFile="job_"+job+"_"+str(index)+".txt"
    outfile = open(outFile, "w")
    for line in lines:
        if ".dst" in line:
            ind=ind+1
            if ind>num*(index-1) and ind<=num*index:
                if ind==num*index:
                    outfile.write(line.replace(",",";"))
                else:
                    outfile.write(line)
            else:
                outfile.write(line.replace("Ds_"+job+".root", "Ds_"+job+"_"+str(index)+".root"))
    outfile.close()
```

运行： ./p.py xxx n

将处理 job_xxx.txt 的文件，
将job_xxx.txt文件中依次抽取n个dst，
依次写入拆分的作业脚本 job_xxx_?.txt

```
// Input REC or DST file name
EventCnvSvc.digiRootInputFile = {
"/besfs3/offline/data/703-1/4180/mc/01/dst/qq/qq_round01_251_IHEP.dst",
"/besfs3/offline/data/703-1/4180/mc/01/dst/qq/qq_round01_252_IHEP.dst",
"/besfs3/offline/data/703-1/4180/mc/01/dst/qq/qq_round01_253_IHEP.dst",
"/besfs3/offline/data/703-1/4180/mc/01/dst/qq/qq_round01_254_IHEP.dst",
"/besfs3/offline/data/703-1/4180/mc/01/dst/qq/qq_round01_255_IHEP.dst"};

ApplicationMgr.HistogramPersistency = "ROOT";
NTupleSvc.Output = { "FILE88 DATAFILE='Ds_011.root' OPT='NEW' TYP='ROOT'"};
```

Job_001.txt

运行 p.py 001 2 得到三个文件： Job_001_1.txt，
Job_001_2.txt 和 Job_001_3.txt 文件：

```
// Input REC or DST file name
EventCnvSvc.digiRootInputFile = {
"/besfs3/offline/data/703-1/4180/mc/01/dst/qq/qq_round01_251_IHEP.dst",
"/besfs3/offline/data/703-1/4180/mc/01/dst/qq/qq_round01_252_IHEP.dst"};

ApplicationMgr.HistogramPersistency = "ROOT";
NTupleSvc.Output = { "FILE88 DATAFILE='Ds_011_1.root' OPT='NEW' TYP='ROOT'"};
```

Job_001_1.txt

Linux上程序开发

- 编译器

- ✓ Fortran : g77
- ✓ C : gcc
- ✓ C++ : g++
- ✓ 自动化编译: GNU make
- ✓ 软件版本控制: CVS

} **GCC**

- 调试工具

- ✓ Text : gdb
- ✓ GUI : emacs

- 集成开发环境 (IDE)

- ✓ Anjuta
- ✓ KDevelop
- ✓ source-navigator

还可以开发基于数据库的
web服务程序。

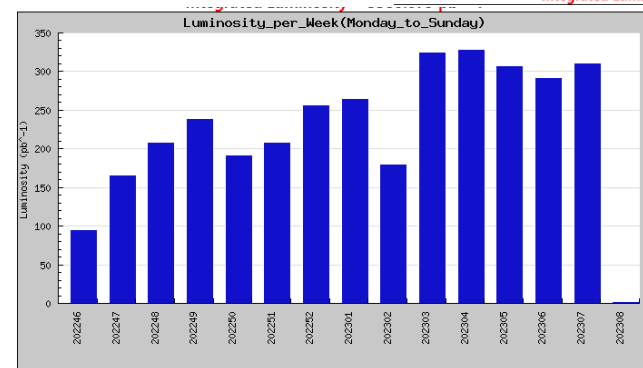
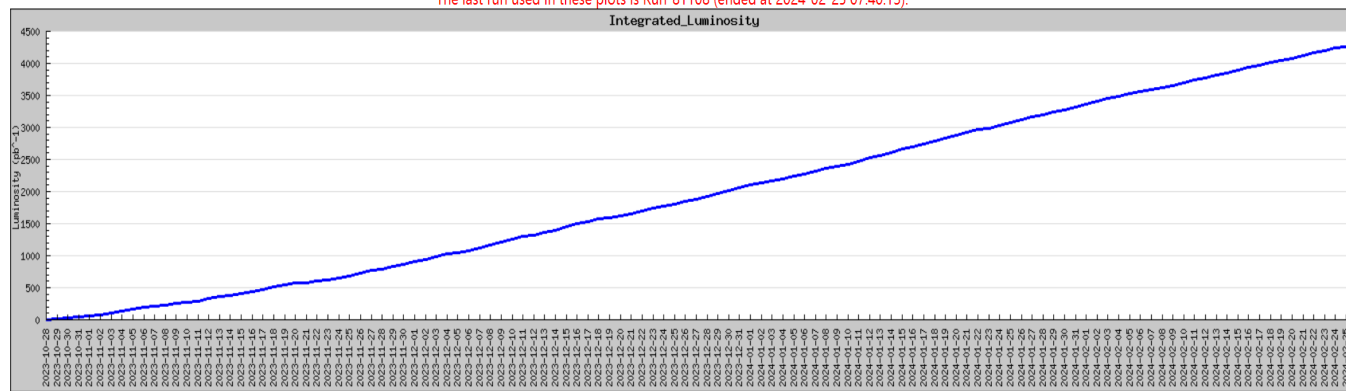


Linux上基于数据库和Web服务器的开发例子:

BESIII 实验取数跟踪示意图的实现:

实现方法:

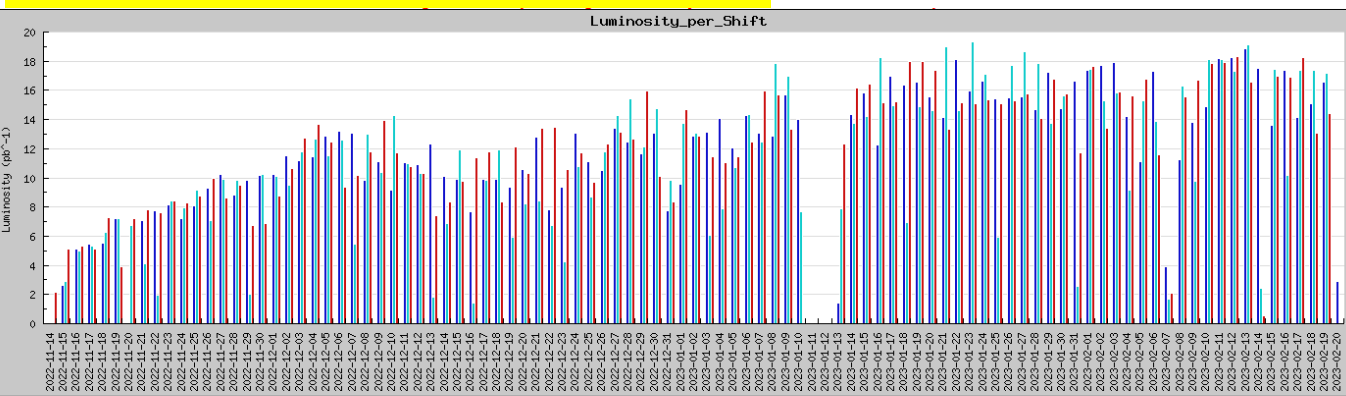
- 读取BESIII 离线数据库中每个run获取的亮度信息。
用到mysql的指令。
- 根据run的开始和结束时间, 统计每天获取的积分亮度, 每班获取的积分亮度。
使用PHP语言来处理。
- 使用PHP语言, 用得到数据做图, 在基于apache的linux web服务器上展示。



每周累计获取数据的积分亮度统计



BESIII亮度破记录!



每班获取数据的积分亮度统计

例子：

BESIII Publication

文章投稿和发表：<https://hnbes3.ihep.ac.cn//publication-status.php>

文章审核进展：<https://hnbes3.ihep.ac.cn//analysis-status.php>

35	494	553	PRL	2022-12-19	2023-01-30 Igor, Ryan	2023-02-28 Haibo	2023-03-23	arXiv:2303.12927	3(3)	Xingsing Xu, Alex Gilman, Xiang Pan	Qiutian Tao	Study of the $f_0(980)$ through the decay $D_s^+ \rightarrow \pi^+ \pi^- e^+ \nu_e$
36	471	600	JHEP	2022-09-16	2022-10-24 Eva, Ryan	2022-11-18 Wolfgang	2022-12-18	arXiv:2212.09043	0(0)	Yu Lu, Jiajia Qin, Xinyu Shan	Amita Lavanaia	Determination of U-spin breaking parameters with an amplitude analysis of the decay $D^0 \rightarrow K_L^0 \pi^+ \pi^-$
37	427	268	PLB	2017-08-07	2017-11-30 name, name	2022-05-11 Wolfgang	2022-06-28	arxiv:2206.13674	2(2)	Weiguo Li, Vindhyawasini Prasad , Alfons Khoukaz	Xingyu Zhou	Measurement of the total and leptonic decay widths of the J/ψ resonance with an energy scan method at BESIII

Nsubmitted = 579, Npublished = 532

BESIII Publication



Accepted/Published

#	PUB	BAM	Target	CWR	PUBcomm	SPapproval	Submitted	arXiv	Accepted/Published	Journal	citation	Author	Title
1	562	683	PRD	2023-09-25	2023-10-17, Liang, Christoph	2023-11-30, Xiaorui	2023-12-20	arXiv:2312.12719	2024-01-26/	Accepted by PRD	0(0)	Junhua Feng	Measurements of Σ electromagnetic form factors in the time-like region using the untagged initial-state radiation technique
2	561	668	PRL	2023-07-20	2023-09-22, Ryan, Yingchun	2023-10-19, Xiaorui	2023-12-18	arXiv:2312.10962	2024-02-14/	Accepted by PRL	0(0)	Bolun Zhang	Observation of significant flavor-SU(3) breaking in the kaon wave function at $\sqrt{s} = 12 \sqrt{2} \text{ GeV}$

GNU Compiler Collection (GCC)

- GCC是GNU项目发布的一套程序语言编译器。它是类Unix系统上免费软件程序的标准编译器。
- 可以编译C/C++, Java, Fortran等程序语言。

ref: <http://gcc.gnu.org/onlinedocs/gcc-4.3.3/gcc.pdf>

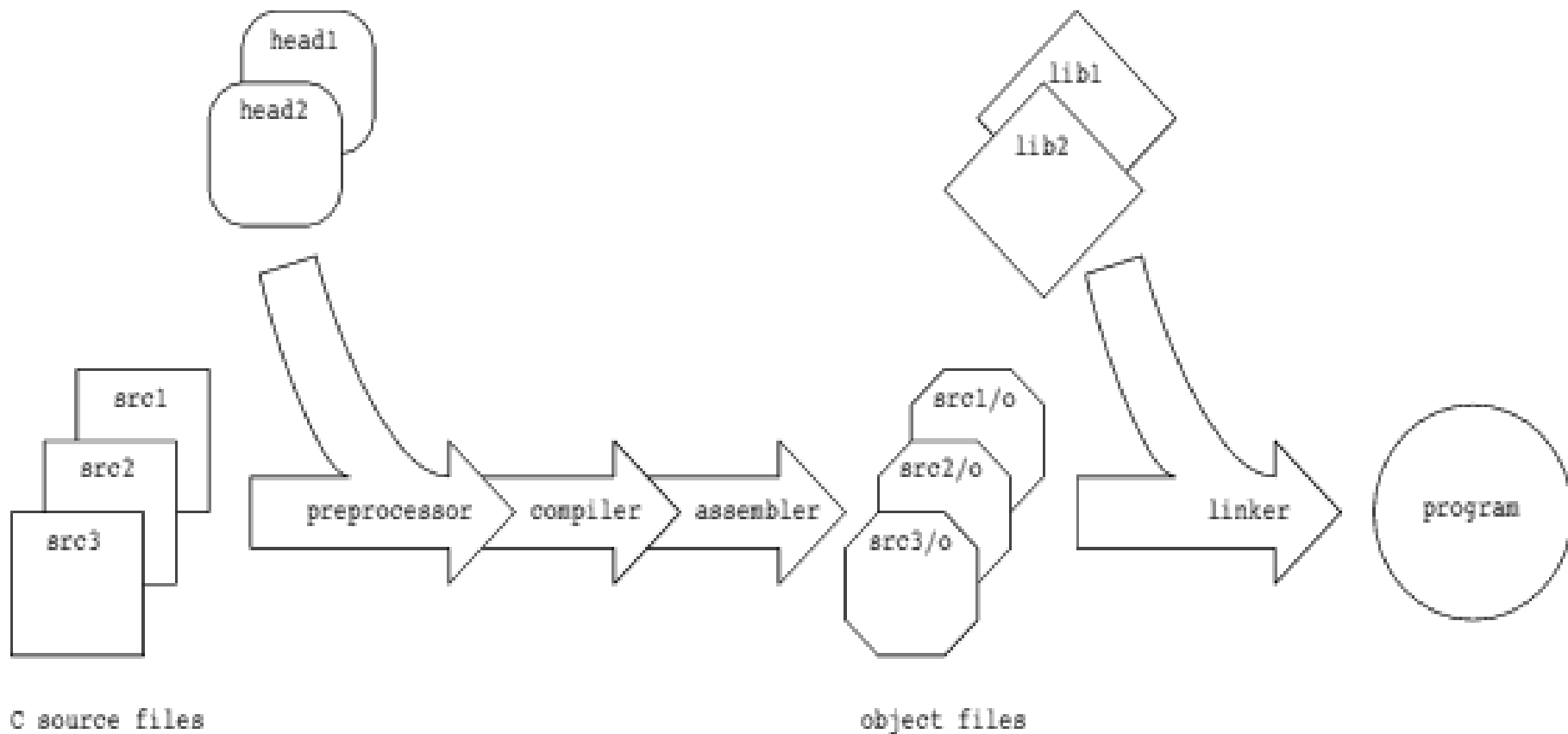
GCC

GCC编译的四个阶段:

阶段	输入	输出
<i>preprocessor</i>	源代码	预处理源代码
<i>compiler</i>	预处理源代码	汇编源代码
<i>assembler</i>	汇编源代码	目标代码
<i>linker</i>	目标代码	可执行文件

headers
(in directory 'h')

libraries



GCC

- Syntax: `gcc/g++ [<options>] <input files>`
编译C程序: **gcc** 编译C++程序: **g++**
- Some useful options:
 - g: 指示编译器在编译的时候, 产生调试信息。如果使用gdb调试程序, 必须使用这个选项。
 - Wall: 显示所有编译警告信息。打开这个选项可以帮助程序员发现非编译错误的问题。
 - o <name>: 指定编译输出文件, 否则输出a.out文件。
- Example: (linux/hello.C)
`g++ -g -Wall -o hello hello.C`

```
/home/cp/codes/linux> g++ -g -Wall -o hello hello.C  
/home/cp/codes/linux> ./hello  
Hello, world!
```

GCC

- GCC编译器重要编译选项：

- ✓ `-c` 只编译并生成目标文件
- ✓ `-D<sym>` 以空字符串定义`<sym>`宏。
`-D<sym>=<def>` 以字符串“`<def>`”定义`<sym>`宏。
- ✓ `-I<directory>` 指定额外的头文件搜索路径`<directory>`。
`-L<directory>` 指定额外的函数库搜索路径`<directory>`。
`-l<library>` 连接时搜索指定的函数库`<library>`。
- ✓ `-ansi` : 只支持ANSI标准的C语法。
- ✓ `-O` or `-O1`, `-O2`, `-O3` : 优化程序。这个选项会减少使编译后的程序的大小和执行时间。但某些时候会在使调试模式出现异常。
- ✓ `-shared` 生成共享目标文件。通常用在建立共享库时。
`-static` 禁止使用共享连接。
- ✓ `-U<sym>` 取消对`<sym>`宏的定义。
- ✓ `-w` 关闭所有的编译警告信息。

举例

- 假如有如下源程序代码文件 (linux/hello_world/) :
hello_1.h hello_1.c hello_2.h
hello_2.c hello.c
- 可以逐个编译源代码文件（.h的头文件不必编译），然后把各个目标文件链接起来：
 > gcc -g -Wall -ansi -c hello.c
 > gcc -g -Wall -ansi -c hello_1.c
 > gcc -g -Wall -ansi -c hello_2.c
 > gcc hello.o hello_1.o hello_2.o -o hello
或
 > gcc -g -Wall -ansi hello.c hello_1.c
 hello_2.c -o hello
- 逐个键入每个源文件太辛苦：使用Makefile

举例 一个root程序编译的例子

1. 一个显示root文件中MC衰变信息的程序:

```
> cat stag.C
#include <iostream>
#include "TFile.h"
#include "TChain.h"
#include "TTree.h"
using namespace std;
int main() {
    TChain * chain = new TChain("truth");
    chain->Add("in.root");
    Int_t          indexmc;
    Int_t          pdgid[100];
    Int_t          motheridx[100];
    chain->SetBranchAddress("indexmc",&indexmc);
    chain->SetBranchAddress("pdgid",pdgid);
    chain->SetBranchAddress("motheridx",motheridx);
    for (Long64_t i=0; i<5;i++) {
        chain->GetEntry(i);
        cout << "event = " << i << endl;
        for ( int k = 0; k != indexmc; k++ ) {
            cout << "    " << k << " id= " << pdgid[k]
                << " mo=" << motheridx[k] << " "
                << pdgid[motheridx[k]] << endl;
        }
    }
}
```

2. 编译成可执行文件:

```
>g++ stag.C `root-config --libs`
`root-config --cflags` -o run
```

3. 运行程序:

```
> ./run
event = 0
    0 id=  433 mo=0    433
    1 id= -431 mo=1   -431
    2 id=  431 mo=0    433
    3 id=   22 mo=0    433
    4 id=  333 mo=2    431
    5 id=  -11 mo=2    431
    6 id=   12 mo=2    431
    7 id=  213 mo=4    333
    8 id= -211 mo=4    333
    9 id=  211 mo=7    213
   10 id=  111 mo=7    213
   11 id=   22 mo=10   111
   12 id=   22 mo=10   111
   13 id=  321 mo=1   -431
   14 id= -321 mo=1   -431
   15 id= -211 mo=1   -431
Event = 1
..... 以下省略
```

举例 一个root程序编译的例子

$e^+ e^- \rightarrow D_s^{*+} D_s^- \rightarrow K^+ K^- \pi^-$
0 1 13 14 15

$D_s^{*+} \gamma$
2 3

$\phi e^+ \nu_e$
4 5 6

$\rho^+ \pi^-$
7 8

$\pi^+ \pi^0$
9 10

$\gamma \gamma$
11, 12

```
> ./run
```

```
event = 0
```

0	id=	433	mo=0	433
1	id=	-431	mo=1	-431
2	id=	431	mo=0	433
3	id=	22	mo=0	433
4	id=	333	mo=2	431
5	id=	-11	mo=2	431
6	id=	12	mo=2	431
7	id=	213	mo=4	333
8	id=	-211	mo=4	333
9	id=	211	mo=7	213
10	id=	111	mo=7	213
11	id=	22	mo=10	111
12	id=	22	mo=10	111
13	id=	321	mo=1	-431
14	id=	-321	mo=1	-431
15	id=	-211	mo=1	-431

```
Event = 1
```

```
..... 以下省略
```

GNU make

- GNU make是一个软件编译自动化工具
 - ✓ 管理一个软件项目的大量源代码文件
 - ✓ 源代码发生改动时，执行尽可能少的编译
 - ✓ 更便利的管理一个项目的结构，相互依赖关系和编译
- Makefile用来告诉make编译哪些文件、怎样编译和何时编译。
- 基本格式为：

targetList: dependencyList

commandList

(command前用<Tab>键，不要使用空格。)

- ✓ targetList: a list of target files
- ✓ dependencyList : a list of files on which the files in targetList depend.
- ✓ commandList : a list of zero or more commands, separated by new lines,

ref: <http://www.gnu.org/software/make/manual/make.html>

Makefile例子

linux/hello_world/Makefile

```
# Simple Makefile with use of gcc could look like this
CC=gcc
CFLAGS=-g -Wall -ansi
OBJS=hello.o hello_1.o hello_2.o
EXE=hello

all: $(EXE)

$(EXE) : $(OBJS)
        $(CC) $(CFLAGS) $(OBJS) -o $(EXE)

.PHONY: clean
clean:
        rm -f $(OBJS) $(EXE)
```

GNU make

- 运行: `"make"` or `"make -f <makefile>"`

```
> make
gcc -g -Wall -ansi -c -o hello.o hello.c
gcc -g -Wall -ansi -c -o hello_1.o hello_1.c
gcc -g -Wall -ansi -c -o hello_2.o hello_2.c
gcc -g -Wall -ansi hello.o hello_1.o hello_2.o -o hello
```

- `make` 做尽量少的执行编译, 只重新编译比目标新的文件
(使用 `touch` 命令测试: `touch -c <filenames>`)

- 不要写成:

```
all: hello.c hello_1.c hello_2.c
    gcc -o hello hello.c hello_1.c hello_2.c
```

因为如果一个文件更新了, 所有的文件都要重新编译一遍。

Makefile常用规则

■ 常用内部变量：

- `$@` 规则中的目标名字
- `$<` 依赖文件中的第一个
- `$^` 所有的依赖文件，重复出现的名字只保留一个

■ 隐含规则：

- `".o"` 目标隐含编译规则：用 `"gcc/g++ -c"` 命令编译
- `".o"` 文件目标的隐含依赖文件：同名的C/C++源文件

■ `.PHONY: target`

- 不是实际的文件目标
- 常用于clean这类并不产生实际文件的目标

静态库和ar命令

- 静态库是预编译的目标文件的集合，它们可被链接进程序。静态库默认以前缀为`lib`，后缀为`".a"`的特殊的存档文件(`archive file`)存储。
- 标准系统库在目录`/usr/lib`与`/lib`下。如C语言的数学库为`/usr/lib/libm.a`。编译时，加`-lm`参数即可被链接。
- 生成自己的静态库使用`ar`命令：

```
ar rcs <libname> <objectfiles> ...
```
- 链接时，需使用`-L`指定库的路径，`-l`指定库文件（库文件格式为去掉前面的`lib`和后面的`.a`后剩下的部分：如`libm.a`为`-lm`）

静态库例子

- 首先，使用-c编译库源文件：

```
gcc -Wall -g -c hello_1.c hello_2.c
```

- 其次，生成静态库文件：

```
ar rcs libhello.a hello_1.o hello_2.o
```

- 编译需要使用libhello.a的主程序文件：

```
gcc -Wall -g -c hello.c
```

- 最后，链接主程序文件和库文件：

```
gcc hello.o -L. -lhello -o hello
```

- 环境变量LIBRARY_PATH指定gcc默认搜索静态库文件的路径

动态库

- 动态库与静态库不同：在静态库中，执行程序完全依赖于程序本身，与其他程序无关，而具有动态库链接的程序运行依赖于它所指向的动态库链接，因为这些动态库是在程序运行时才动态搜索和加载的。当运行时找不到对应的库文件时将报错。
- 优点：
 - ✓ 节省磁盘空间：可执行程序内不包含库文件的函数内容
 - ✓ 库文件的更新，不需要重新编译程序文件，便于大项目的开发
 - ✓ 节省内存：只有一个库文件被加载到内容，多个可执行文件可同时使用该库文件

创建动态库

- 使用-fPIC (Position Independent Code,意思是与文件的位置无关) 编译库源文件:

```
gcc -fPIC -Wall -g -c hello_1.c hello_2.c
```

- 创建动态库:

```
gcc -g -shared -o libhello.so hello_1.o  
hello_2.o
```

动态库默认前缀和后缀是lib 和“.so”

- 链接主程序和动态库, -L选项指定搜索动态库文件的附加路径 (默认在环境变量LD_LIBRARY_PATH中搜索):

```
gcc -Wall -g -o hello hello.c -L. -lhello
```

- 运行: ./hello

/etc/ld.so.conf 有系统动态链接共享库的路径,记录一些 编译链接用library所在的目录,将新的目录输入后用ldconfig可重新刷新,以后链接时自动到指定目录搜索。

GNU Debugger (GDB)

- 调试器能观察程序执行时的内部活动，或程序出错时发生了什么。GDB主要能做四件事：

- ✓ 运行程序
- ✓ 设置断点
- ✓ 查看变量、CPU寄存器等信息
- ✓ 改变你的程序

- 启动GDB：

```
gdb <programfile> [<corefile>|<pid>]
```

- ✓ <programfile> 可执行程序文件
- ✓ <corefile> 程序的core dump文件（程序崩溃记录）
- ✓ <pid> 正在运行程序的进程PID

ref: <http://www.cs.princeton.edu/~benjasik/gdb/gdbtut.html>