

---

# Linux系统基础

---

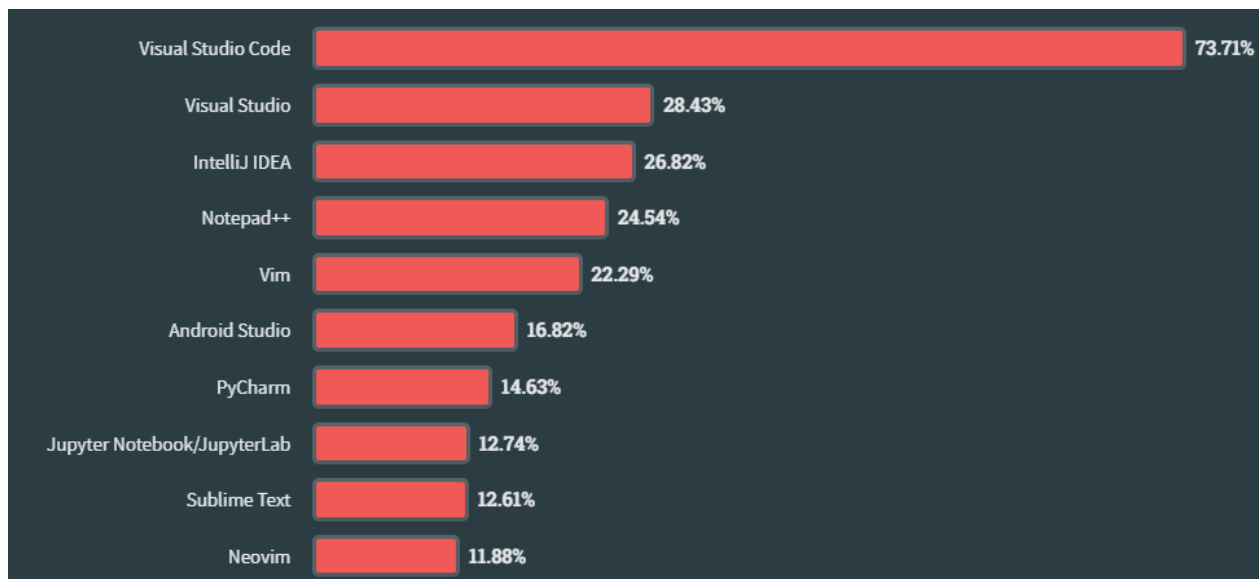
第二天

陈健  
2024年

---

# 该学哪个代码编辑器

---



stack overflow 2023调查

<https://survey.stackoverflow.co/2023/#worked-with-vs-want-to-work-with-new-collab-tools-worked-want>

# vim历史

---

- 1976年, Bill Joy发布vi
- 1991年, Bram Moolenaar发布vim
- 有“模式”的编辑器



Ken Thompson



Bill Joy



Bram Moolenaar

# vim的设计哲学

---

**Vim**通过模式的转换、命令的组合和数以万计的插件，保证程序员在编程的过程中，双手尽可能保留在键盘中央的区域，并且不需要用到鼠标。

# vim的安装

---

- 查看默认安装的vim版本
  - `apt list --installed | grep vim`
- 安装文本界面版vim
  - `apt install vim-nox`
- 安装GNOME桌面版
  - `apt install vim-gtk3`

# vim的模式

---

- 正常模式
  - 缺省模式，在任何其他模式中，都可以通过键盘上的**Esc**键返回正常模式
- 插入模式
  - 输入文本时使用。在正常模式下键入**i**或**a**即可进入该模式
- 替换模式
  - 替换文本时使用。在正常模式下键入**r**或**R**即可进入该模式
- 可视化模式
  - 用于选定文本块。在正常模式下键入**v**来按字符选定，Vim也提供其他不同的选定方法，包括按行(**s-v**)和按列块(**c-v**)
- 命令模式
  - 用于执行命令。在正常模式下键入冒号(**:**)即可进入该模式。使用斜杠(**/**)或问号(**?**)开始搜索也算作命令模式。命令模式下的命令要输入回车键(**Enter**)才算完成

# vim的基本操作

---

- ❑ 在正常模式下，键入i进入插入模式，输入内容，按下<ESC>返回正常模式
- ❑ 在正常模式下，输入冒号:进入命令模式，输入q退出vim编辑器，或输入w保存文件
- ❑ 在正常模式下，输入冒号:进入命令模式，输入help key|command查询某个按键或命令的帮助

# vim的缓存、标签页和窗口

---

- Vim可以打开多个标签页，每个标签页可以包含多个窗口，每个窗口对应某个缓存（即打开的文件）
  - 窗口和缓存不是一一对应的关系
- 窗口操作
  - 水平分隔 :sp
  - 上下分隔 :vsp
  - 切换窗口 ctrl-w w
- 标签操作
  - 打开新标签 :tabnew
  - 切换标签 :tabnext或:tabn或快捷键gt
  - 关闭标签 :tabclose或:tabc



# Vim的正常模式的本质

---

Vim的正常模式就是vim的编程接口，本质上就是一种编程语言。

# 正常模式——移动

## □ 基本移动 hjkl（左、下、上、右）

!	"	#	\$	%	&	'	(	)	0	*	=	{	}	Home ~ ^
1	2	3	4	5	6	7	8	9	0	:	-	[	]	
Esc	Q	W	E	R	T	Y	U	I	O	P	Line Feed	Enter ↵	Here is	
Ctrl	A	S	D	F	G	H ←	J ↓	K ↑	L →	+	@	 \	Rub -	Break
Shift ⬆	Z	X	C	V	B	N	M	<	>	?	Shift ⬆	Repeat	Clear	

- 按词移动 w（下一个词）b（上一个词）e（词尾）
- 行操作 0（行首）\$（行尾）^（行首第一个非空格字符）
- 翻页 <C-B><C-F> 翻半页 <C-U> <C-D>
- 文件 gg（文件头）G（文件尾）
- 查找
  - f{字符} 跳转到光标所在行中，在光标之后的第一个字符位置
  - t{字符} 跳转到光标所在行中，在光标之后的第一个字符前一个位置

# 正常模式——编辑

---

- i 插入模式
- a 在当前光标之后进入插入模式
- o/O 在当前行之下/之上插入行
- d{移动命令} 删除{移动命令}
  - dw 删除词 d\$ 删除到行尾 d0 删除行首 dd 删除整行
- c{移动命令} 修改{移动命令}
  - cw 修改词 cc 修改整行
- x 删除字符
- r 替换光标下的字符
- u 撤销 <C-r> 恢复
- y{移动命令} 复制
  - yy 复制整行 yw 复制词
- p 粘贴
- . 重复上一次的修改操作

# 正常模式——计数

---

- 4j 向下移动4行
- 3e 向后移动三个词到词尾
- 2dw 删除2个词

# 正常模式——修饰语

---

## □ i 内部

- ci( 修改当前括号内的内容
- ci[ 修改当前方括号内的内容

## □ a 周围

- da' 删除单引号字符串，包括单引号

# 自定义vim

---

## □ Vim的配置文件

- ~/.vimrc

# Vim插件

---

- ❑ ctrlp.vim 模糊文件查找
  - `mkdir -p ~/.vim/pack/plugins/start`
  - `git clone --depth=1`  
`https://github.com/ctrlpvim/ctrlp.vim.git`  
`~/.vim/pack/plugins/start/ctrlp`
- ❑ ack.vim 代码搜索
  - `git clone --depth=1`  
`https://github.com/mileszs/ack.vim.git`  
`~/.vim/pack/plugins/start/ack`
- ❑ nerdtree 文件浏览器
- ❑ 插件仓库
  - `https://vimawesome.com`

# Vim练习

---

- 完成vimtutor
- 将我们提供的vimrc文件保存为`~/.vimrc`，阅读这个文件中的注释，观察vim在新设置下有哪些不同之处



# Shell脚本

## ——变量

---

- 变量赋值
  - `foo=bar`
  - `foo = bar` 可以吗?
- 获取变量值
  - `$foo`
  - `echo $foo`
- 单引号'和双引号"
  - `echo "$foo"`
  - `echo '$foo'`

# Shell脚本

## ——函数

---

```
mcd () {  
    mkdir -p "$1"  
    cd "$1"  
}
```

### 方法1

直接在**shell**提示符中输入函数内容

**\$ mcd** 目录名

### 方法2

将函数内容保存为文件

**\$ source** 文件名

**\$ mcd** 目录名

# Shell脚本

## ——常见特殊变量

---

\$0 脚本名

\$1-\$9 脚本参数

\$? 上一个命令的退出状态

\$\_ 上一个命令的最后一个参数

!! 完整的上一个命令，包括参数

特殊变量的完整列表见网址

<https://tldp.org/LDP/abs/html/special-chars.html>

# Shell脚本

## ——退出码结合&&、||

---

false || echo "Oops,fail"

true || echo "Will not be printed"

true && echo "Things went well"

false && echo "Will not be printed"

false ; echo "This will always run"

# Shell脚本

## ——命令替换、进程替换

---

### □ 命令替换 \$(CMD)

- `foo=$(pwd)`
- `echo $foo`
- `echo "We are in $(pwd)"`

### □ 进程替换 <(CMD)

- `diff <(ls) <(ls ..)`

# Shell脚本示例

---

```
#!/bin/bash
```

```
echo "Starting program at $(date)"
```

```
echo "Running program $0 with $# arguments with pid $$"
```

```
for file in "$@"; do
```

```
    grep testonly "$file" > /dev/null 2> /dev/null
```

```
    if [[ $? -ne 0 ]]; then
```

```
        echo "File $file does not have any testonly, adding one"
```

```
        echo "# testonly" >> "$file"
```

```
    fi
```

```
done
```

# 课堂练习

---

请编写两个**bash**函数**savedir**和**cddir**。执行函数**savedir**时，当前的工作目录应当以某种形式保存。执行函数**cddir**时，无论现在处在什么目录中，都**cd**进入当时执行函数**savedir**时所处的目录。

为了方便调试，你可以把代码写在单独的文件**test.sh**中，并通过**source test.sh**命令加载函数。

注：在**bash**中，**source**命令用于读取指定的**shell**脚本文件，并执行该文件中的命令，如同这些命令是直接当前**shell**提示符下输入的一样。使用**source**命令执行的脚本不会启动一个新的**shell**进程，因此任何在脚本中定义的变量或函数都会在当前**shell**环境中可用。

# Shell脚本

## ——通配

---

### ☐ 通配符\*、?

- `rm foo?`
- `rm foo*`

### ☐ 花括号{}

- `convert image.{png,jpg}`
  - ☐ 展开为 `convert image.png image.jpg`
- `cp /project/{foo,bar,baz}.sh /newproject`
  - ☐ 展开为 `cp /project/foo.sh /project/bar.sh /project/baz.sh /newproject`
- `mv *{.py,.sh} folder`
  - ☐ 会移动所有\*.py和\*.sh文件
- `touch {foo,bar}/{a..h}`



# 关于通配和变量的区别

---

假设test目录下有两个文件a.txt和b.txt，我们想将这两个文件转移到test2目录下，请问下面两个方法，哪个会成功？

方法1

```
mv test/*.txt test2
```

方法2

```
FILES="a.txt b.txt"
```

```
mv test/$FILES test2
```

# Shell脚本

## ——脚本静态检查

---


### □ Shellcheck

- <https://github.com/koalaman/shellcheck>
- `apt install shellcheck`

# python脚本

---

```
#!/usr/bin/python
import sys
for arg in reversed(sys.argv[1:]):
    print(arg)
```



```
#!/usr/bin/env python
```

这种做法好吗？

# Shell函数 vs 脚本

---

- ❑ 函数只能与shell使用相同的语言，脚本可以使用任意语言
- ❑ 函数仅在定义时被加载，脚本会在每次执行时被加载
- ❑ 函数会在当前shell环境中执行，脚本会在单独的进程中执行

# Shell工具

## ——查找文件

---

- 查找当前目录及其子目录下所有名为src的目录
  - `find . -name src -type d`
- 查找当前目录及其子目录下名为test的目录下的所有py文件
  - `find . -path '**/test/*.py' -type f`
- 查找当前目录下前一天修改的所有文件
  - `find . -mtime -1`
- 查找当前目录下所有大小在100k至1M的后缀为.tar.gz的文件
  - `find . -size +100k -size -1M -name '*.tar.gz'`
- 删除当前目录下后缀为.tmp的所有文件
  - `find . -name '*.tmp' -exec rm {} \;`
  - `find . -type f -exec rm -- {} +`
  - `find . -type f -exec rm {} \;`

# Shell工具

## ——查找文件

---

### ❑ fdfind

- <https://github.com/sharkdp/fd>
- `apt install fd-find`
- `fdfind .sh`

### ❑ locate

- `locate test`
- `updatedb`负责更新后台数据库

### ❑ find vs locate

- <https://unix.stackexchange.com/questions/60205/locate-vs-find-usage-pros-and-cons-of-each-other>

# Shell工具

## ——查找文件内容

---

### □ grep

- grep text file
- 递归搜索目录
  - grep -R text .
- 获取匹配文本的上下文
  - grep -C 2 text \*
- 输出不匹配的结果
  - grep -v text file

### □ rg

- <https://github.com/BurntSushi/ripgrep>
- apt install ripgrep
- rg test \*.sh

# Shell工具

## ——查找shell命令

---

- history
  - history
  - history | grep find
- ctrl+r
  - 对命令历史记录进行回溯搜索
- 基于历史的命令自动补全
  - zsh
    - 根据最近使用过的开头相同的命令，动态地对当前的shell命令进行补全
- 命令保护
  - 在命令的开头加上一个空格
  - 在用户家目录下的.bashrc中添加如下内容
    - HISTCONTROL=ignorespace



# 作业2提交方法和截止时间

---

- ❑ 实验报告的文件名命名统一为：学号\_lab02.pdf
- ❑ 提交截止时间：2024年7月26日零点
- ❑ 实验报告通过电子邮件发送给  
chenj@nju.edu.cn