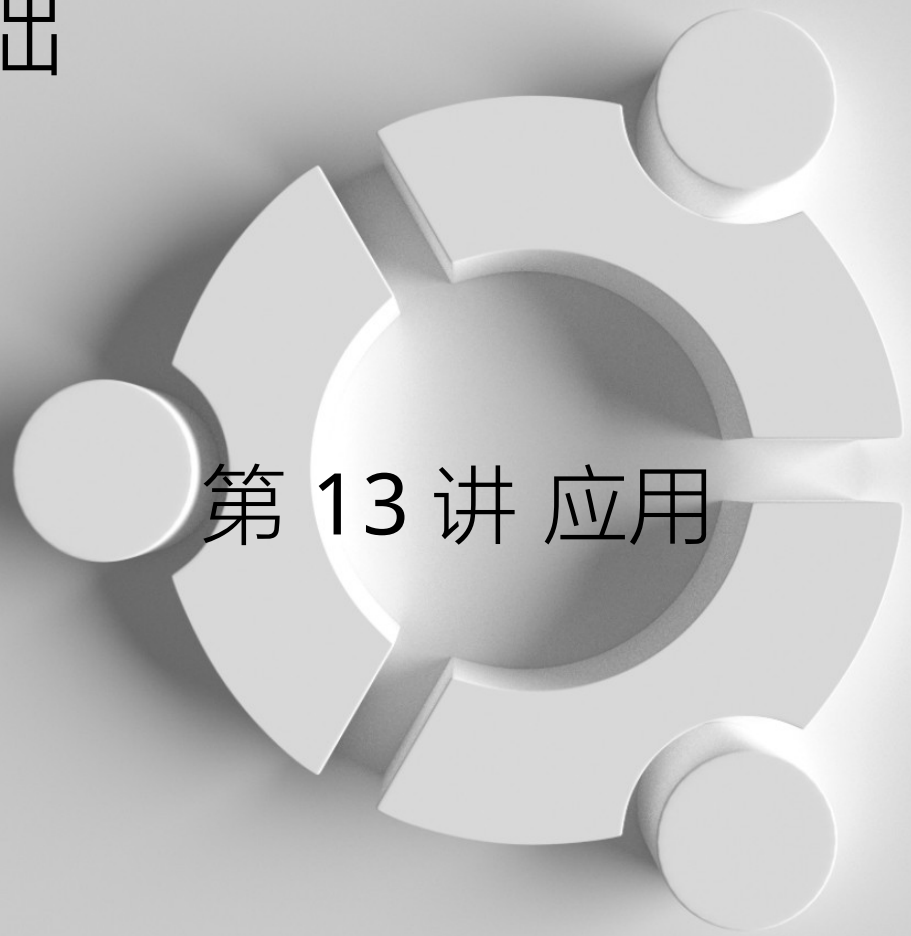
# Linux 基础



## 第 13 讲 应用

# 正则表达式

# 正则表达式

- 如果列举计算机史上的伟大发明，正则表达式一定会在列。其应用范围遍布各个角落，即使是非 IT 人员，只要他经常使用软件，多少都会直接或间接接触到正则表达式。

- 几乎所有的编程语言都支持正则表达式，所有的编辑软件都支持正则表达式。

# 正则表达式

- 正则表达式是对字符串结构模式的一种描述形式。更直白的说，正则表达式主要是为了按照规则匹配文本。
- 因为最开始只是为了处理正规文本，因此得名：正则表达式。而现在正则表达式早已突破了限制。

查找含有 linux，并且格式为 pdf 的文件：

```
linux.*\.pdf$
```

# 正则表达式

- 即使正则表达式已经出现有 50 年，但是这项技术即使在今天，我们所处的教学环境，对正则表达式仍然不够重视。
- 而且掌握正则表达式确实不太容易。

正则表达式示例： `^(12|13|14|15|16|17|18|19)[0-9]{9}$`

因为到目前国内主要软件环境仍然在微软体系内，微软在长期的产品开发中都没有对这项技术有足够重视。而正则表达式诞生于 UNIX 文化体系中。事实上，正是 UNIX 之父肯 · 汤姆森（Ken·Thompson）最早应用正则表达式。

# 元字符

- 正则表达式那些稀奇古怪的特殊字符组合总是让首次看到它们的人一脸茫然并知难而退。
- 但是很快你就会了解它们的作用。之所以会有很多特殊字符，是因为这些字符是元字符。
- 比如 . 表示任意字符，* 前一个模式出现任意次数，^ 表示要从开头就要匹配。下表给出了常用的元字符。

# 元字符

| 元字符 | 说明 |
| --- | --- |
| ^ | 匹配开头，当 ^ 出现在正则表达式中间往往不作为元字符，但是这要看程序如何处理。 |
| . | 匹配任意字符。 |
| * | 前一个模式匹配任意次数。 |
| $ | 尾匹配，表示匹配结束正好是文本的末尾。 |
| + | 前一个模式出现 1 次或多次。 |
| \ | 转义，可以让元字符作为普通字符。 |
| {} | 次数限制， {n}, {n,m}, {n,} 表示前一个模式出现 n 次， n 到 m 次， n 次及以上。 |
| [] | 范围集合， [0-9] 匹配 0 到 9 的字符， [0-9a-z] 匹配 0 到 9 或 a-z 的字符。 |
| \| | 逻辑或， x\|y 匹配 x 或 y 。 |

# 关于元字符的说明

- 在一般的使用上，以上元字符都可以正常运行。但是具体到不同编程语言，不同命令则需要查看相应的手册。
- 因为不同编程语言或是命令工具采用的正则库是不同的。
- 现在正则表达式的实现支持的元字符集和模式太复杂。开发者往往经常在具体使用手册中研究如何编写针对不同编程语言、不同命令的正则表达式。

# 正则表达式流派

- 在正则表达式漫长的发展历史中，出现了各种各样的工具，实现方式和基本思想都一样，只是在处理各种模式上有所区别，支持的功能也有区别。
- 然而并没有什么标准区遵循，标准总是晚来的，谁也不能未卜先知，知道大家需要什么，先制定标准再去实现。
- 结果就是，各种流派，大家各自为战。

# 正则表达式流派

- 后来 POSIX 开始制定标准，但是因为历史问题，实现上还是粗略的归为 3 大类。
- 所以，我们现在面对的情况尽管有所好转，但是仍然是换个环境就要研究一下它的手册。
- 比如， grep 不支持 [0-9A-Z] 这种多个范围并列的情况，而 JS 中的正则引擎则支持。

# grep 命令

- grep 是 Linux/Unix 上用于正则匹配的命令。
- grep 支持的正则引擎比较复杂，同时支持早期的和后来新型的匹配引擎，属于混合类型。
- 默认使用 grep 支持的元字符有限，比如不支持 | 。
- 要使用功能更强的匹配则需要使用 egrep 或 grep -E 。

# grep 示例

- 默认使用 grep [ 正则表达式字符串 ] 启动后，会等待用户输入并给出匹配结果，这可以用来做匹配测试。
- 查找目录 /etc 中所有文件包含 SYS_ 的行：

```
grep -R ‘SYS_’ /etc 2> /dev/null
```

- 查找 /usr/share 中包含 background 的配置文件：

```
grep -R ‘background’ /usr/share
```

# grep 示例

- 查找目录 /usr/include 中包含 struct stat 的结构体，并搜索整个结构体的声明：

```
egrep -Rz 'struct stat \{[^\}].*\}' /usr/include
```

- 选项 z 表示多行匹配，[^\}] 则表示只要不包含 } 则匹配成功。

# 其他

- `egrep` 就是 `grep -E`。
- `fgrep` 就是 `grep -F`。
- 正则表达式需要在实战中提升技能，需要在编程中不断地提升，仅仅是书本和课堂的学习是不够的。

# 了解 bash 启动和退出

# bash 启动

- bash 在首次会话登录的启动时，要去 /etc/ 中加载 /profile 配置文件。（bash --login 方式启动 ）
- 此外， bash 还会在当前用户主目录中按照顺序寻找：

  .bash_profile .bash_login .profile

- 之后读取 /etc/bash.bahsrc 以及用户主目录的：

  .bash_bashrc .bashrc

- 并运行其中的命令。

# 配置 bash

- 所以要更改 bash 启动的行为或是添加一些环境变量则可以在 .profile 或 .bashrc 中进行。
- 在 CentOS 这一发行版上，使用的配置文件是：

  .bash_profile .bash_bashrc

# 注意事项

- 如果你要更改 PATH 环境变量，要去 .profile 中或是全局的 profile 中去改，如果在 .bashrc 中加入以下命令：

  export PATH="$PATH:/usr/local/node/bin"

- 那么每次启动新的 bash 都会运行此项，你的 PATH 数据量会成倍增加。

# bash 退出

- 当运行 exit 退出 bash， bash 会在退出之前执行 .bash_logout 中的命令。
- 如果需要做一些退出后的处理，可以把命令写在此处。

# 编写自动初始化脚本

# 重新安装 Linux 面临的问题

- 每次重新部署环境，都要把主要的开发配置、系统设置等选项重新走一遍。
- 这是耗时繁重的工作。但是这个情况可以通过自动化脚本来解决。

# 如何建立自动化脚本

- 我们这里给出一个方案，然后你可以在此基础上，扩展功能。
- 对我们来说，最主要的是要安装一系列的软件，还有编辑配置文件，或者是把之前备份的配置文件复制过去。
- 然后还有创建一系列的目录，可能还要把一些保存的代码以及其他文件复制过来。

# 设计方案

- 这里利用 git 来进行仓库管理，每次在新环境只需要通过 git 来 clone 。只有断网的情况不得已复制本地的。
- 在仓库中，创建 uinit.sh 表示 Ubuntu 环境的初始化。
- 然后建立相关目录和文件。

不要有对整个主目录建立 git 仓库的想法，但是如果你真的想这样做，也可一试，请在测试系统上做，然后建立自己的认知。

# 代码和考虑的问题

- 详细的代码和结构请参考课程示例。
- 当你自己设计，有一些需要考虑的问题：
  - 如何切换到程序所在目录，保证运行时工作目录正确。
  - 是不是要强制 `root` 身份运行。
  - 哪些东西可以放在其中，哪些应该舍弃。

# 思考练习

- 如何在执行命令时，也可以在当前目录查找命令，无论当前所在目录是什么？
- 探索 `zip` 和 `unzip` 命令的使用。
- 在自动化脚本中，如果有一些文件不想公开，需要加密处理如何操作。