

# Linux 系统概论

天津医科大学  
生物医学工程与技术学院

2017-2018 学年下学期 (春)  
2016 级生信班

# 第五章 高级 Linux 命令

伊现富 (Yi Xianfu)

天津医科大学 (TJMU)  
生物医学工程与技术学院

2018 年 6 月



# 教学提纲

1

引言

2

正则表达式和元字符

- 元字符
- 正则表达式

3

find 和 grep

- find
- grep

4

文本处理命令

5

sed 和 AWK

- sed

- AWK

6

生物信息学中的应用

7

回顾与总结

- 总结
- 思考题

## 1 引言

## 2 正则表达式和元字符

- 元字符
- 正则表达式

## 3 find 和 grep

- find
- grep

## 4 文本处理命令

### 5 sed 和 AWK

- sed
- AWK

## 6 生物信息学中的应用

## 7 回顾与总结

- 总结
- 思考题



## 已经学习

- 命令的工作原理
- 常用的基本命令

## 即将学习

- 正则表达式和元字符
- 高级命令：find, grep, sed, AWK



## 已经学习

- 命令的工作原理
- 常用的基本命令

## 即将学习

- 正则表达式和元字符
- 高级命令：find, grep, sed, AWK



1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题



## 正则表达式

- 正则表达式 (regular expression) 是具有一定句法的集合或短语，表示某类文本或字符串。
- 优势：使得用户能够利用比较少的预定义字符集合来表示多种字符组合。
- 正则表达式包含元字符或普通字符。

## 元字符

- 元字符 (metacharacter) 是代表一组字符或命令的字符。
- 优势：减少和命令一起使用的文本数量，用最小的字符集表示多组文本。

## 相关命令

less,more; grep,egrep,fgrep; sed,AWK; emacs,vim; ...

# 正则和元字符 | 简介

## 正则表达式

- 正则表达式 (regular expression) 是具有一定句法的集合或短语，表示某类文本或字符串。
- 优势：使得用户能够利用比较少的预定义字符集合来表示多种字符组合。
- 正则表达式包含元字符或普通字符。

## 元字符

- 元字符 (metacharacter) 是代表一组字符或命令的字符。
- 优势：减少和命令一起使用的文本数量，用最小的字符集表示多组文本。

## 相关命令

less,more; grep,egrep,fgrep; sed,AWK; emacs,vim; ...

# 正则和元字符 | 简介

## 正则表达式

- 正则表达式 (regular expression) 是具有一定句法的集合或短语，表示某类文本或字符串。
- 优势：使得用户能够利用比较少的预定义字符集合来表示多种字符组合。
- 正则表达式包含元字符或普通字符。

## 元字符

- 元字符 (metacharacter) 是代表一组字符或命令的字符。
- 优势：减少和命令一起使用的文本数量，用最小的字符集表示多组文本。

## 相关命令

less,more; grep,egrep,fgrep; sed,AWK; emacs,vim; ...

# Regular Expression | Introduction

**Regular expressions** are text strings used for matching a specific **pattern**, or to search for a specific location, such as the start or end of a line or a word. Regular expressions can contain both normal characters or so-called **metacharacters**, such as `*` and `$`.

Many text editors and utilities such as **vi**, **sed**, **awk**, **find** and **grep** work extensively with regular expressions. Some of the popular computer languages that use regular expressions include **Perl**, **Python** and **Ruby**. It can get rather complicated and there are whole books written about regular expressions.

These regular expressions are different from the wildcards (or "metacharacters") used in filename matching in command shells such as **bash**.



# Regular Expression | Pattern

Search Patterns	Usage
. (dot)	Match any single character
a   z	Match a or z
\$	Match end of string
*	Match preceding item 0 or more times



# Regular Expression | Example

## Sentence

the quick brown fox jumped over the lazy dog

Command	Usage
<i>a..</i>	matches azy
<i>b. j.</i>	matches both br and ju
<i>..\$</i>	matches og
<i>l.*</i>	matches lazy dog
<i>l.*y</i>	matches lazy
<i>the.*</i>	matches the whole sentence



1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式
- find 和 grep
- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

- 7 回顾与总结
- 总结
  - 思考题

# 正则和元字符 | 元字符 | 字符

元字符	功能	表达式	匹配示例
一般字符	匹配自身	abc	abc
.	匹配除换行符(\n)外的任意一个字符	a.c	abc
[]	匹配字符集中的任意一个字符	a [bcd]e	abe,ace,ade
[a-z]	匹配任意一个小写字母	a [a-z]e	aae,abe,...,aye,aze
[0-9]	匹配任意一个数字	a [0-9]e	a0e,a1e,...,a8e,a9e
[^]	不匹配字符集中的任意一个字符	a [^bcd]e	除 abe,ace,ade 外
\	转义字符（删除紧跟其后的字符的特殊意义）	a \.c	a .c



# 正则和元字符 | 元字符 | 字符集

元字符	功能	表达式	匹配示例
\d	数字, [0-9]	a\dc	a1c
\D	非数字, [^\d]	a\Dc	abc
\s	空白字符, [\t\r\n\f\v]	a\sc	a c
\S	非空白字符, [^\s]	a\Sc	abc
\w	单词字符, [A-Za-z0-9_]	a\wc	abc
\W	非单词字符, [^\w]	a\Wc	a c



# 正则和元字符 | 元字符 | 量词

元字符	功能	表达式	匹配示例
?	匹配前一个字符零次或一次	abc?	ab,abc
*	匹配前一个字符零次或多次	abc*	ab,abccc
+	匹配前一个字符一次或多次	abc+	abc,abccc
{m}	匹配前一个字符 m 次	ab{2}c	abbc
{m, n}	匹配前一个字符 m 至 n 次	ab{1,2}c	abc,abbc



# 正则和元字符 | 元字符 | 边界

元字符	功能	表达式	匹配示例
^	匹配字符串开头	^abc	[开头]abc
\$	匹配字符串末尾	abc\$	abc[末尾]



# 正则和元字符 | 元字符 | 其他

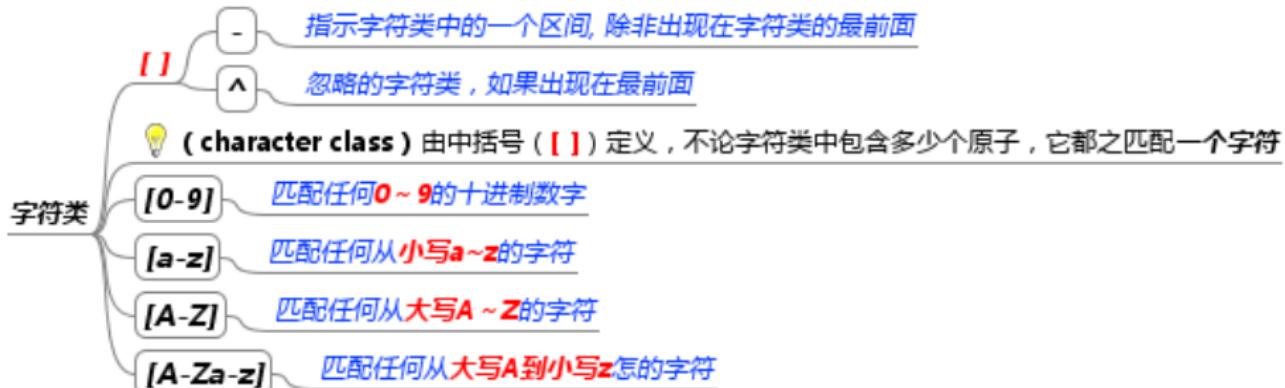
元字符	功能	表达式	匹配示例
( )	分组, 创建用于匹配的子串	ab(cd)?	ab,abcd
	或, 两边的项目二选一	ab(cd ef)	abcd,abef



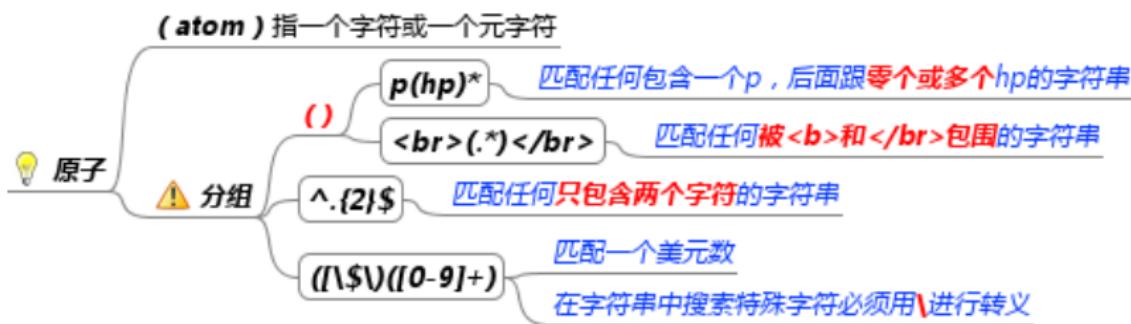
# 正则和元字符 | 元字符

正则表达式	描    述	示    例
^	行起始标记	^tux 匹配以tux起始的行
\$	行尾标记	tux\$ 匹配以tux结尾的行
.	匹配任意一个字符	Hack. 匹配Hackl和Hacki，但是不能匹配Hackl2和Hackil，它只能匹配单个字符
[]	匹配包含在[字符]之中的任意一个字符	coo[kl] 匹配cook或cool
[^ ]	匹配除[^字符]之外的任意一个字符	9[^01] 匹配92、93，但是不匹配91或90
[ - ]	匹配[]中指定范围内的任意一个字符	[1-5] 匹配从1~5的任意一个数字
?	匹配之前的项1次或0次	colou?r 匹配color或colour，但是不能匹配colouur
+	匹配之前的项1次或多次	Rollno-9+ 匹配Rollno-99、Rollno-9，但是不能匹配Rollno-
*	匹配之前的项0次或多次	co*1 匹配cl、col、coool等
( )	创建一个用于匹配的子串	ma(tri)? 匹配max或maxtrix
{n}	匹配之前的项n次	[0-9]{3} 匹配任意一个三位数，[0-9]{3} 可以扩展为[0-9][0-9][0-9]
{n, }	之前的项至少需要匹配n次	[0-9]{2, } 匹配任意一个两位或更多位的数字
{n, m}	指定之前的项所必需匹配的最小次数和最大次数	[0-9]{2,5} 匹配从两位数到五位数之间的任意一个数字
	交替——匹配 两边的任意一项	Oct (1st   2nd) 匹配Oct 1st或Oct 2nd
\	转义符可以将上面介绍的特殊字符进行转义	a\.b 匹配ab，但不能匹配ajb。通过在 . 之间加上前缀 \，从而忽略了 . 的特殊意义

# 正则和元字符 | 元字符

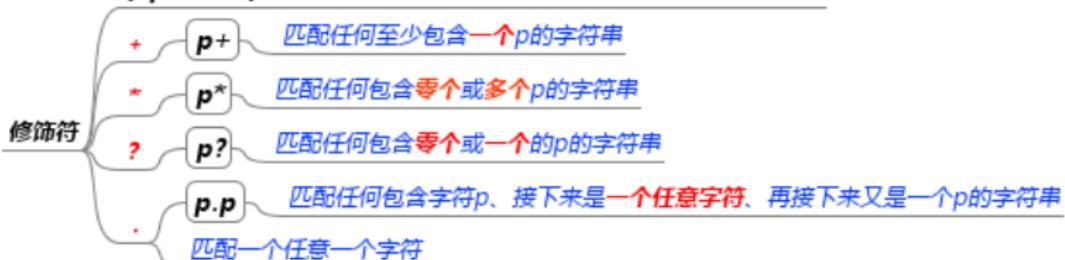


元字符 ( metacharacter ) 是指有特殊含义 ( 不同于其字面含义 ) 的字符

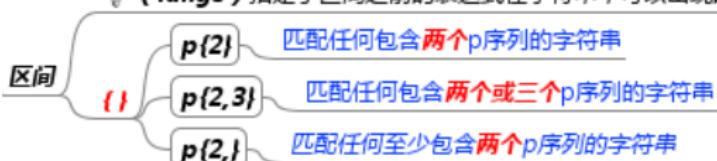


# 正则和元字符 | 元字符

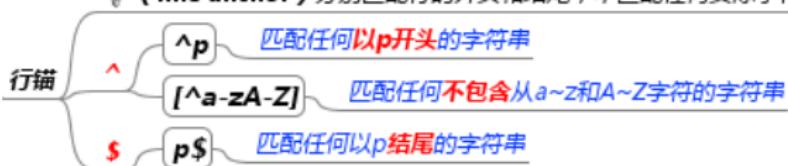
( **qualifier** ) 用于限制匹配时该修饰符前面的表达式可以出现的次数



💡 ( **range** ) 指定了区间之前的表达式在字符串中可以出现的次数



💡 ( **line anchor** ) 分别匹配行的开头和结尾，不匹配任何实际字符



转义 \ ( **escape** ) 在原子前面使用

OR | 元字符用作为正则表达式中的或操作符

1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题

# 正则和元字符 | 正则表达式 | 元字符



# 正则和元字符 | 正则表达式 | 实例

```
1 |Christian Scott lives here and will put on a Christmas party. |
2 |There are around 30 to 35 people invited. |
3 |They are: |
4 |                                         Tom |
5 |Dan |
6 | Rhonda Savage |
7 |Nicky and Kimberly. |
8 |Steve, Suzanne, Ginger and Larry. |
```

a. `/^A-Z].$/` 查找文本中所有以大写字母开头、后跟两个任意字符，再跟一个换行符的行。查找结果是第 5 行的 Dan。

b. `/^A-Z][a-z ]*3[0-5]/` 查找所有以大写字母开头、后跟零个或多个小写字母或空格，再跟数字 3 和一个 0~5 之间的数字的行。查找结果是第 2 行。

c. `/[a-z]*\./` 查找包含跟在零个或多个小写字母后的句点的行。结果是第 1、2、7、8 行。

d. `^ *[A-Z][a-z][a-z]$` 查找以零个或多个空格开头(注意：制表符不算空格)，后跟一个大写字母、两个小写字母和一个换行符的行。结果将是第 4 行的 Tom 和第 5 行的 Dan。

e. `/^A-Za-z]*[^,][A-Za-z]*$/` 查找以零个或多个大/小写字母开头，后跟一个非逗号的字符，再跟零个或多个大/小写字母和一个换行符的行。结果是第 5 行。



## 正则表达式

- `\d+\.\d+\.\d+\.\d+`
- `(^\d{15}$) | (^$\d{17} ([0-9]|X)$)`
- `^(0[0-9]{2,3}\d){2}([2-9][0-9]{6,7})+(\d-[0-9]{1,4}){2}$`
- `/^[_a-zA-Z]([_a-zA-Z0-9]*[_a-zA-Z]?[_a-zA-Z0-9]+)*@[_a-zA-Z]*[_a-zA-Z]?[_a-zA-Z0-9]+@[\.][_a-zA-Z]{2,3}([\.\.][_a-zA-Z]{2}){2}$/i`

## 正则含义

- IP 地址
- 身份证号码（15 位或 18 位）
- 固定电话：3-4 位区号，7-8 位座机号码，1-4 位分机号
- 邮箱地址

# 正则和元字符 | 正则表达式 | 复杂实例

## 正则表达式

- `\d+\.\d+\.\d+\.\d+`
- `(^\d{15}$) | (^$\d{17} ([0-9]|X)$)`
- `^(0[0-9]{2,3}\d){2}([2-9][0-9]{6,7})+(\d-[0-9]{1,4}){2}$`
- `/^[_a-zA-Z]([_a-zA-Z0-9]*[_a-zA-Z]?[_a-zA-Z0-9]+)*@[_a-zA-Z]*[_a-zA-Z]?[_a-zA-Z0-9]+@[\.][_a-zA-Z]{2,3}([\.\[_a-zA-Z]{2}){2}$/i`

## 正则含义

- IP 地址
- 身份证号码（15 位或 18 位）
- 固定电话：3-4 位区号，7-8 位座机号码，1-4 位分机号
- 邮箱地址

# 正则和元字符 | 正则表达式 | 复杂实例

## 正则表达式

- `\d+\.\d+\.\d+\.\d+`
- `(^\d{15}$) | (^$\d{17} ([0-9]|X)$)`
- `^(0[0-9]{2,3}\-)?([2-9][0-9]{6,7})+(\-\-[0-9]{1,4})?$_`
- `/^[_a-zA-Z]([_a-zA-Z0-9]*[_a-zA-Z]?[_a-zA-Z0-9]+)*@[_a-zA-Z0-9]*[_a-zA-Z]?[_a-zA-Z0-9]+@[\.][_a-zA-Z]{2,3}([\.\.][_a-zA-Z]{2})?$/i`

## 正则含义

- IP 地址
- 身份证号码（15 位或 18 位）
- 固定电话：3-4 位区号，7-8 位座机号码，1-4 位分机号
- 邮箱地址

# 正则和元字符 | 正则表达式 | 复杂实例

## 正则表达式

- `\d+\.\d+\.\d+\.\d+`
- `(^\d{15}$) | (^$\d{17}([0-9]|X)$)`
- `^(0[0-9]{2,3}\-)?([2-9][0-9]{6,7})+(\-\-[0-9]{1,4})?$_`
- `/^[_a-zA-Z]([_a-zA-Z0-9]*[_a-zA-Z]?[_a-zA-Z0-9]+)*@[_a-zA-Z0-9]*[_a-zA-Z]?[_a-zA-Z0-9]+@[\.][_a-zA-Z]{2,3}([\.\.][_a-zA-Z]{2})?$/i`

## 正则含义

- IP 地址
- 身份证号码（15 位或 18 位）
- 固定电话：3-4 位区号，7-8 位座机号码，1-4 位分机号
- 邮箱地址

# 正则和元字符 | 正则表达式 | 复杂实例

## 正则表达式

- `\d+\.\d+\.\d+\.\d+`
- `(^\d{15}$) | (^$\d{17}([0-9]|X)$)`
- `^(0[0-9]{2,3}\-)?([2-9][0-9]{6,7})+(\-\-[0-9]{1,4})?$/`
- `/^[_a-zA-Z]([_a-zA-Z0-9]*[_a-zA-Z]?[_a-zA-Z0-9]+)*@[_a-zA-Z]*[_a-zA-Z]?[_a-zA-Z0-9]+[\.\.][_a-zA-Z]{2,3}([\.\.][_a-zA-Z]{2})?$/i`

## 正则含义

- IP 地址
- 身份证号码（15 位或 18 位）
- 固定电话：3-4 位区号，7-8 位座机号码，1-4 位分机号
- 邮箱地址

# 正则和元字符 | 正则表达式 | 复杂实例

## 正则表达式

- `\d+\.\d+\.\d+\.\d+`
- `(^\d{15}$) | (^$\d{17}([0-9]|X)$)`
- `^(0[0-9]{2,3}\-)?([2-9][0-9]{6,7})+(\-\-[0-9]{1,4})?$/`
- `/^[_a-zA-Z]([_a-zA-Z0-9]*[_-]?[_a-zA-Z0-9]+)*@[_a-zA-Z0-9]*[_-]?[_a-zA-Z0-9]+)[\.\.][_a-zA-Z]{2,3}([\.\.][_a-zA-Z]{2})?$/i`

## 正则含义

- IP 地址
- 身份证号码（15 位或 18 位）
- 固定电话：3-4 位区号，7-8 位座机号码，1-4 位分机号
- 邮箱地址

# 正则和元字符 | 正则表达式 | 复杂实例

## 正则表达式

- `\d+\.\d+\.\d+\.\d+`
- `(^\d{15}$) | (^$\d{17}([0-9]|X)$)`
- `^(0[0-9]{2,3}\-)?([2-9][0-9]{6,7})+(\-\-[0-9]{1,4})?$/`
- `/^[_a-zA-Z]([_a-zA-Z0-9]*[_-]?[_a-zA-Z0-9]+)*@[_a-zA-Z0-9]*[_-]?[_a-zA-Z0-9]+)[\.\.][_a-zA-Z]{2,3}([\.\.][_a-zA-Z]{2})?$/i`

## 正则含义

- IP 地址
- 身份证号码（15 位或 18 位）
- 固定电话：3-4 位区号，7-8 位座机号码，1-4 位分机号
- 邮箱地址

# 正则和元字符 | 正则表达式 | 复杂实例

## 正则表达式

- `\d+\.\d+\.\d+\.\d+`
- `(^\d{15}$) | (^$\d{17}([0-9]|X)$)`
- `^(0[0-9]{2,3}\-)?([2-9][0-9]{6,7})+(\-\-[0-9]{1,4})?$/`
- `/^[_a-zA-Z]([_a-zA-Z0-9]*[_-]?[_a-zA-Z0-9]+)*@[_a-zA-Z0-9]*[_-]?[_a-zA-Z0-9]+)+[\.\.][_a-zA-Z]{2,3}([\.\.][_a-zA-Z]{2})?$/i`

## 正则含义

- IP 地址
- 身份证号码（15 位或 18 位）
- 固定电话：3-4 位区号，7-8 位座机号码，1-4 位分机号
- 邮箱地址

# 正则和元字符 | 正则表达式 | 复杂实例

## 正则表达式

- `\d+\.\d+\.\d+\.\d+`
- `(^\d{15}$) | (^$\d{17}([0-9]|X)$)`
- `^(0[0-9]{2,3}\-)?([2-9][0-9]{6,7})+(\-\-[0-9]{1,4})?$/`
- `/^[_a-zA-Z]([_a-zA-Z0-9]*[_-]?[_a-zA-Z0-9]+)*@[_a-zA-Z0-9]*[_-]?[_a-zA-Z0-9]+)[\.\.][_a-zA-Z]{2,3}([\.\.][_a-zA-Z]{2})?$/i`

## 正则含义

- IP 地址
- 身份证号码（15 位或 18 位）
- 固定电话：3-4 位区号，7-8 位座机号码，1-4 位分机号
- 邮箱地址

以下正则表达式中能够抓到 abc 字符串的有：

- ①  $x^*$
- ②  $ax^*$
- ③  $abx^*$
- ④  $ax^*b$
- ⑤  $abcx^*$
- ⑥  $abx^*c$
- ⑦  $ax^*bc$
- ⑧  $bx^*c$
- ⑨  $bcx^*$
- ⑩  $x^*bc$



- 在命令行 (command line) 的位置里，通配符 (wildcard) 只作用于参数 (argument) 的路径 (path) 上
- 正则表达式 (RE) 却只用于“字符串处理”的程序中，这与路径名一点关系也没有
- 正则表达式 (RE) 所处理的字符串，通常是指纯文本或通过 STDIN 读进的内容



# 教学提纲

1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题



1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题

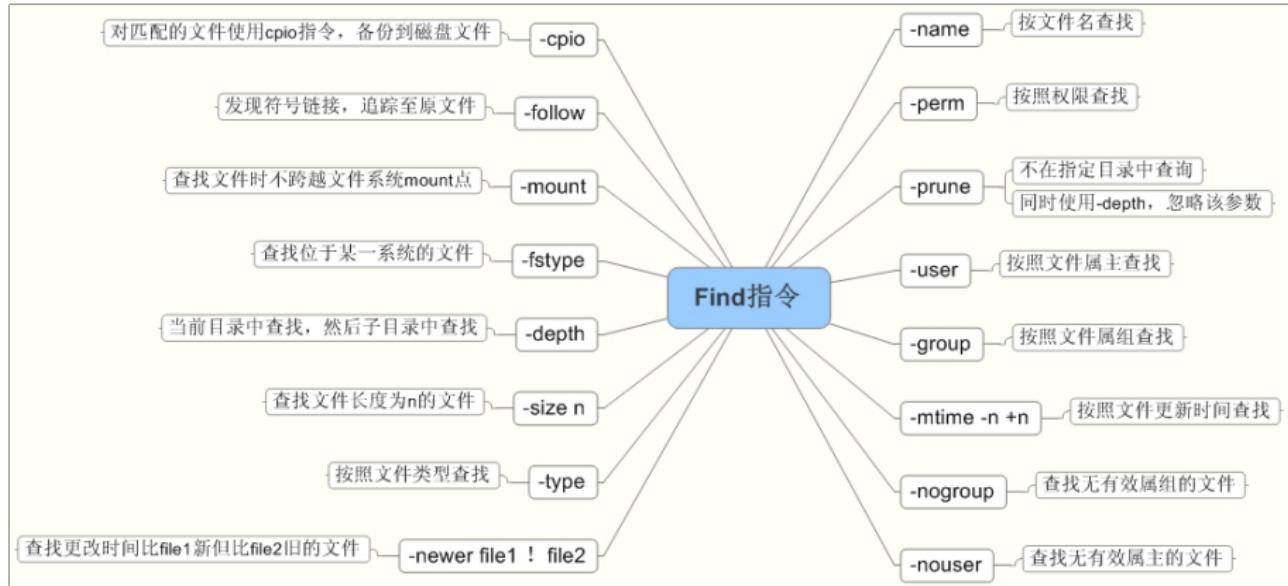


选项	说明
--help	显示帮助信息
--maxdepth n	(最多) 向下搜索到第 n 层子目录
--mindepth m	(至少) 从第 m 层子目录开始搜索
--mount	防止搜索其他文件系统



测试	说明
+n	搜索大于数字 n 的所有内容
-n	搜索小于数字 n 的所有内容
n	搜索准确匹配数字 n 的所有内容
-amin +n	搜索 n 分钟之前访问过的文件
-atime -n	搜索 n 天内访问过的文件
-fstype TYPE	搜索指定类型的文件系统
-gid n	搜索 GID 等于 n 的文件
-group GROUP	搜索组名为 GROUP 的文件
-name FILE	搜索名为 FILE 的文件（可以使用通配符）
-perm MODE	搜索权限设置为 MODE 的文件
-size n	搜索大小等于 n 的文件（可以使用 k、M、G 等）
-type TYPE	搜索 TYPE 类型的文件（如 d、f、l 等）
-uid n	搜索 UID 为 n 的文件
-user USER	搜索用户名为 USER 的文件





## 实际问题

- 找到 /etc 目录下面的 passwd 文件
- 找到用户 USER 在 /home 目录下拥有的所有文件
- 找到 /home 目录下大于 2G 的文件

## 解决方法

- `find /etc -name passwd`
- `find /home -user USER`
- `find /home -size +2G -print`



## 实际问题

- 找到 /etc 目录下面的 passwd 文件
- 找到用户 USER 在 /home 目录下拥有的所有文件
  - 找到 /home 目录下大于 2G 的文件

## 解决方法

- `find /etc -name passwd`
- `find /home -user USER`
- `find /home -size +2G -print`



## 实际问题

- 找到 /etc 目录下面的 passwd 文件
- 找到用户 USER 在 /home 目录下拥有的所有文件
- 找到 /home 目录下大于 2G 的文件

## 解决方法

- `find /etc -name passwd`
- `find /home -user USER`
- `find /home -size +2G -print`



## 实际问题

- 找到 /etc 目录下面的 passwd 文件
- 找到用户 USER 在 /home 目录下拥有的所有文件
- 找到 /home 目录下大于 2G 的文件

## 解决方法

- `find /etc -name passwd`
- `find /home -user USER`
- `find /home -size +2G -print`



## 实际问题

- 找到 /etc 目录下面的 passwd 文件
- 找到用户 USER 在 /home 目录下拥有的所有文件
- 找到 /home 目录下大于 2G 的文件

## 解决方法

- find /etc -name passwd
- find /home -user USER
- find /home -size +2G -print



## 实际问题

- 找到 /etc 目录下面的 passwd 文件
- 找到用户 USER 在 /home 目录下拥有的所有文件
- 找到 /home 目录下大于 2G 的文件

## 解决方法

- find /etc -name passwd
- find /home -user USER
- find /home -size +2G -print



## 实际问题

- 找到 /etc 目录下面的 passwd 文件
- 找到用户 USER 在 /home 目录下拥有的所有文件
- 找到 /home 目录下大于 2G 的文件

## 解决方法

- find /etc -name passwd
- find /home -user USER
- find /home -size +2G -print



# find | 实例

```
find . -name "*.jpg" -exec rm -fv {} \;
```

找出以".jpg"结尾的文件

rm -fv 进行删除操作

} 代表find过滤出来的以".jpg"结尾的文件

";": -exec参数指定的command结束符; "\\"对后面的分号进行转义

## \; vs. ';' -exec vs. -ok

- find -name "\*.swp" -exec rm {} \;
- find -name "\*.swp" -exec rm {} ';'
- find -name "\*.swp" -ok rm {} \;

# find | 实例

```
find . -name "*.jpg" -exec rm -fv {} \;
```

找出以".jpg"结尾的文件

rm -fv 进行删除操作

} 代表find过滤出来的以".jpg"结尾的文件

";": -exec参数指定的command结束符; "\\"对后面的分号进行转义

## \; vs. ';' -exec vs. -ok

- find -name "\*.swp" -exec rm {} \;
- find -name "\*.swp" -exec rm {} ';'
- find -name "\*.swp" -ok rm {} \;

# 教学提纲

1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题



**定义** Globally search a Regular Expression and Print (全局正则表达式打印)

**功能** 在文件中搜索用户所指定的序列，然后将结果打印出来

**用途** 搜索文件或者在文件中进行查找

**结构** grep StringToSearchFor FileToSearch



grep [options] 'Pattern' FILE

- color=auto 为匹配的项自动添加颜色
- v 反向选取，只显示不符合模式的行。
- o 只显示被模式匹配到的字符串，而不是整个行
- i 忽略大小写
- A n 显示匹配到的行时，顺带显示其后面的n行，A表示after
- B n 前面的n行
- C n 前后的n行
- E 等于egrep，表示使用扩展的正则表达式
- w 字符正则表达式，类似于锚定行首
- r 递归搜索



# grep | 使用方法

Command	Usage
<code>grep [pattern] &lt;filename&gt;</code>	Search for a pattern in a file and print all matching lines
<code>grep -v [pattern] &lt;filename&gt;</code>	Print all lines that do <b>not</b> match the pattern
<code>grep [0-9] &lt;filename&gt;</code>	Print the lines that contain the numbers 0 through 9
<code>grep -C 3 [pattern] &lt;filename&gt;</code>	Print context of lines (specified number of lines above and below the pattern) for matching the pattern. Here the number of lines is specified as 3.



## 实际问题

- 在/etc 目录下面的文件中搜索 root 字符串
- 搜索/etc/passwd 中不含 root 字符串的所有账户
- 搜索/etc/passwd 中包含 root 字符串的所有账户

## 解决方法

- grep root /etc/\*
- grep -v root /etc/passwd
- cat /etc/passwd | grep root



## 实际问题

- 在/etc 目录下面的文件中搜索 root 字符串
- 搜索/etc/passwd 中不含 root 字符串的所有账户
- 搜索/etc/passwd 中包含 root 字符串的所有账户

## 解决方法

- grep root /etc/\*
- grep -v root /etc/passwd
- cat /etc/passwd | grep root



## 实际问题

- 在/etc 目录下面的文件中搜索 root 字符串
- 搜索/etc/passwd 中不含 root 字符串的所有账户
- 搜索/etc/passwd 中包含 root 字符串的所有账户

## 解决方法

- grep root /etc/\*
- grep -v root /etc/passwd
- cat /etc/passwd | grep root



## 实际问题

- 在/etc 目录下面的文件中搜索 root 字符串
- 搜索/etc/passwd 中不含 root 字符串的所有账户
- 搜索/etc/passwd 中包含 root 字符串的所有账户

## 解决方法

- grep root /etc/\*
- grep -v root /etc/passwd
- cat /etc/passwd | grep root



## 实际问题

- 在/etc 目录下面的文件中搜索 root 字符串
- 搜索/etc/passwd 中不含 root 字符串的所有账户
- 搜索/etc/passwd 中包含 root 字符串的所有账户

## 解决方法

- grep root /etc/\*
- grep -v root /etc/passwd
- cat /etc/passwd | grep root



## 实际问题

- 在/etc 目录下面的文件中搜索 root 字符串
- 搜索/etc/passwd 中不含 root 字符串的所有账户
- 搜索/etc/passwd 中包含 root 字符串的所有账户

## 解决方法

- grep root /etc/\*
- grep -v root /etc/passwd
- cat /etc/passwd | grep root



## 实际问题

- 在/etc 目录下面的文件中搜索 root 字符串
- 搜索/etc/passwd 中不含 root 字符串的所有账户
- 搜索/etc/passwd 中包含 root 字符串的所有账户

## 解决方法

- grep root /etc/\*
- grep -v root /etc/passwd
- cat /etc/passwd | grep root



表 4-5 grep 部分举例

命令	显示
grep '[A-Z]' list	list 中包含一个大写字母的行
grep '[0-9]' data	data 中包含数字的行
grep '[A-Z]...[0-9]' list	list 中包含以大写字母开始、数字结尾的 5 个字符组合的行
grep '\.pic\$' filelist	filelist 中以.pic 结尾的行



## 具体问题

- 不区分大小写，查找包括 “the” 的行，并显示行号
- 查找不包括 “the” 的行，统计行数
- 查找空白行
- 查找包括 2-5 个字母 o 的行

## grep 命令

- grep -in "the" input.txt
- grep -cv "the" input.txt
- grep -n '^\$' input.txt
- grep -n "o\{2,5\}" input.txt

## 具体问题

- 不区分大小写，查找包括 “the” 的行，并显示行号
- 查找不包括 “the” 的行，统计行数
- 查找空白行
- 查找包括 2-5 个字母 o 的行

## grep 命令

- grep -in "the" input.txt
- grep -cv "the" input.txt
- grep -n "^\\$" input.txt
- grep -n "o\{2,5\}" input.txt

## 具体问题

- 不区分大小写，查找包括 “the” 的行，并显示行号
- 查找不包括 “the” 的行，统计行数
- 查找空白行
- 查找包括 2-5 个字母 o 的行

## grep 命令

- grep -in "the" input.txt
- grep -cv "the" input.txt
- grep -n "o\\$" input.txt
- grep -n "o\{2,5\}" input.txt

## 具体问题

- 不区分大小写，查找包括 “the” 的行，并显示行号
- 查找不包括 “the” 的行，统计行数
- 查找空白行
- 查找包括 2-5 个字母 o 的行

## grep 命令

- grep -in "the" input.txt
- grep -cv "the" input.txt
- grep -n "^\\$" input.txt
- grep -n "o\{2,5\}" input.txt

## 具体问题

- 不区分大小写，查找包括 “the” 的行，并显示行号
- 查找不包括 “the” 的行，统计行数
- 查找空白行
- 查找包括 2-5 个字母 o 的行

## grep 命令

- grep -in "the" input.txt
- grep -cv "the" input.txt
- grep -n "^\\$" input.txt
- grep -n "o\{2,5\}" input.txt

## 具体问题

- 不区分大小写，查找包括 “the” 的行，并显示行号
- 查找不包括 “the” 的行，统计行数
- 查找空白行
- 查找包括 2-5 个字母 o 的行

## grep 命令

- grep -in "the" input.txt
- grep -cv "the" input.txt
- grep -n "^\\$" input.txt
- grep -n "o\{2,5\}" input.txt

## 具体问题

- 不区分大小写，查找包括 “the” 的行，并显示行号
- 查找不包括 “the” 的行，统计行数
- 查找空白行
- 查找包括 2-5 个字母 o 的行

## grep 命令

- grep -in "the" input.txt
- grep -cv "the" input.txt
- grep -n "^\\$" input.txt
- grep -n "o\{2,5\}" input.txt

## 具体问题

- 不区分大小写，查找包括 “the” 的行，并显示行号
- 查找不包括 “the” 的行，统计行数
- 查找空白行
- 查找包括 2-5 个字母 o 的行

## grep 命令

- grep -in "the" input.txt
- grep -cv "the" input.txt
- grep -n "^\\$" input.txt
- grep -n "o\{2,5\}" input.txt

## 具体问题

- 不区分大小写，查找包括 “the” 的行，并显示行号
- 查找不包括 “the” 的行，统计行数
- 查找空白行
- 查找包括 2-5 个字母 o 的行

## grep 命令

- grep -in "the" input.txt
- grep -cv "the" input.txt
- grep -n "^\\$" input.txt
- grep -n "o\{2,5\}" input.txt

1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题



# 文本处理命令



# 文本处理命令 | sort

## sort

**sort** is used to rearrange the lines of a text file either in ascending or descending order, according to a sort key. You can also sort by particular fields of a file. The default sort key is the order of the ASCII characters (i.e., essentially alphabetically).

When used with the **-u** option, sort checks for unique values after sorting the records (lines). It is equivalent to running **uniq** on the output of **sort**.

Syntax	Usage
<code>sort &lt;filename&gt;</code>	Sort the lines in the specified file
<code>cat file1 file2   sort</code>	Append the two files, then sort the lines and display the output on the terminal
<code>sort -r &lt;filename&gt;</code>	Sort the lines in reverse order

# 文本处理命令 | sort

## sort

**sort** is used to rearrange the lines of a text file either in ascending or descending order, according to a sort key. You can also sort by particular fields of a file. The default sort key is the order of the ASCII characters (i.e., essentially alphabetically).

When used with the **-u** option, sort checks for unique values after sorting the records (lines). It is equivalent to running **uniq** on the output of **sort**.

Syntax	Usage
<code>sort &lt;filename&gt;</code>	Sort the lines in the specified file
<code>cat file1 file2   sort</code>	Append the two files, then sort the lines and display the output on the terminal
<code>sort -r &lt;filename&gt;</code>	Sort the lines in reverse order

## uniq

**uniq** is used to remove duplicate lines in a text file and is useful for simplifying text display. **uniq** requires that the duplicate entries to be removed are consecutive. Therefore one often runs **sort** first and then pipes the output into **uniq**; if **sort** is passed the **-u** option it can do all this in one step.

To remove duplicate entries from some files, use the following command:

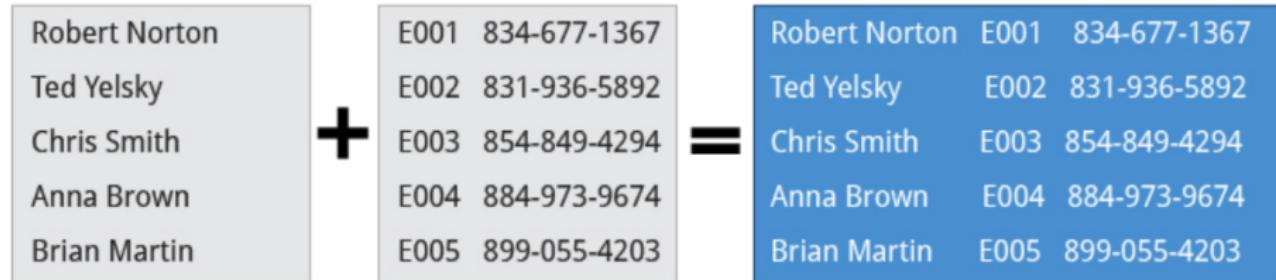
```
sort file1 file2 | uniq > file3; OR sort -u file1 file2 > file3
```

To count the number of duplicate entries, use the following command:

```
uniq -c filename
```



# 文本处理命令 | paste



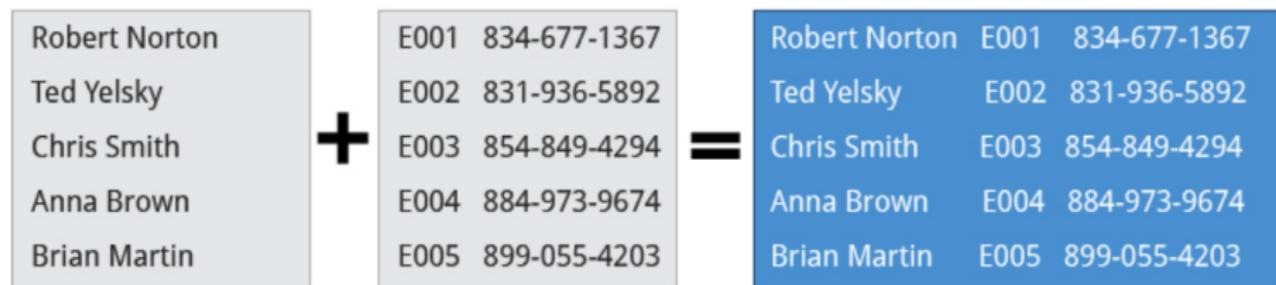
## Commands

- `paste file1 file2`
- `paste -d, file1 file2`

## Options

- `-d`: delimiters, which specify a list of delimiters to be used instead of tabs for separating consecutive values on a single line. Each delimiter is used in turn; when the list has been exhausted, `paste` begins again at the first delimiter.
- `-s`: which causes `paste` to append the data in series rather than in parallel; that is, in a horizontal rather than vertical fashion.

# 文本处理命令 | paste



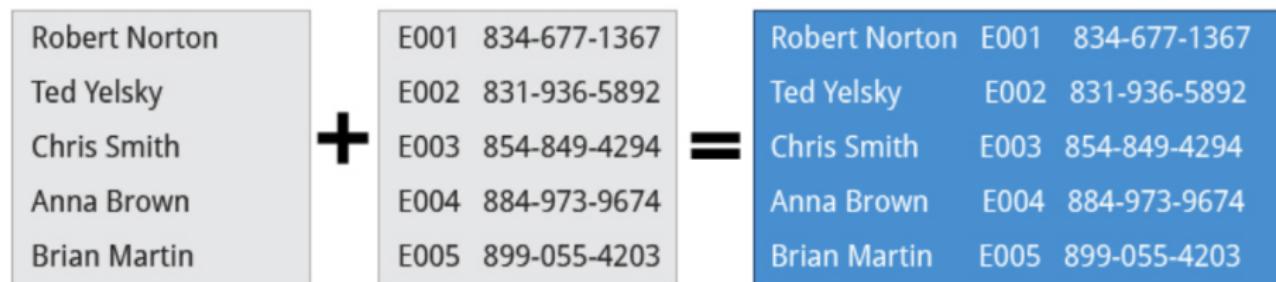
## Commands

- `paste file1 file2`
- `paste -d, file1 file2`

## Options

- `-d`: delimiters, which specify a list of delimiters to be used instead of tabs for separating consecutive values on a single line. Each delimiter is used in turn; when the list has been exhausted, paste begins again at the first delimiter.
- `-s`: which causes `paste` to append the data in series rather than in parallel; that is, in a horizontal rather than vertical fashion.

# 文本处理命令 | paste



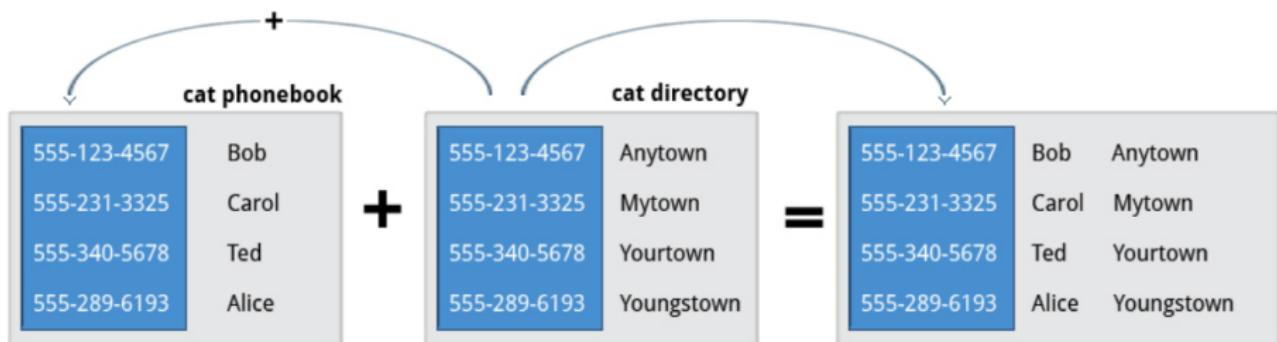
## Commands

- `paste file1 file2`
- `paste -d, file1 file2`

## Options

- `-d`: delimiters, which specify a list of delimiters to be used instead of tabs for separating consecutive values on a single line. Each delimiter is used in turn; when the list has been exhausted, paste begins again at the first delimiter.
- `-s`: which causes **paste** to append the data in series rather than in parallel; that is, in a horizontal rather than vertical fashion.

# 文本处理命令 | join

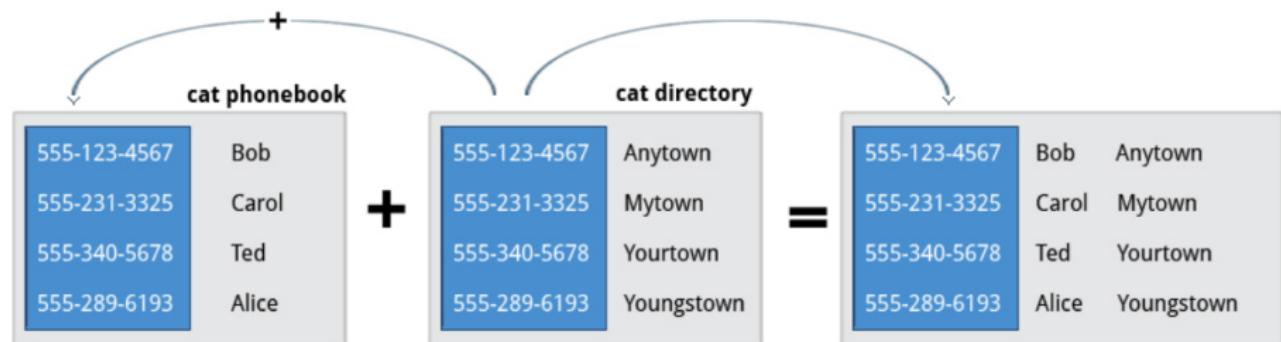


## Command

- join file1 file2



# 文本处理命令 | join

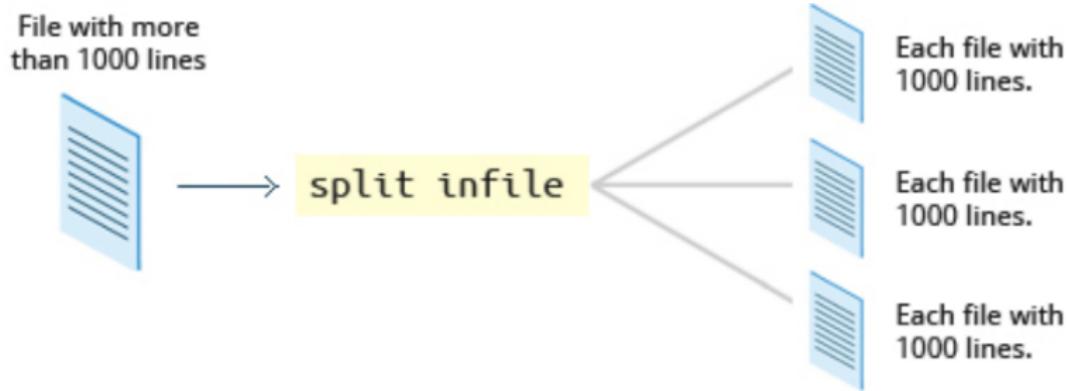


## Command

- join file1 file2



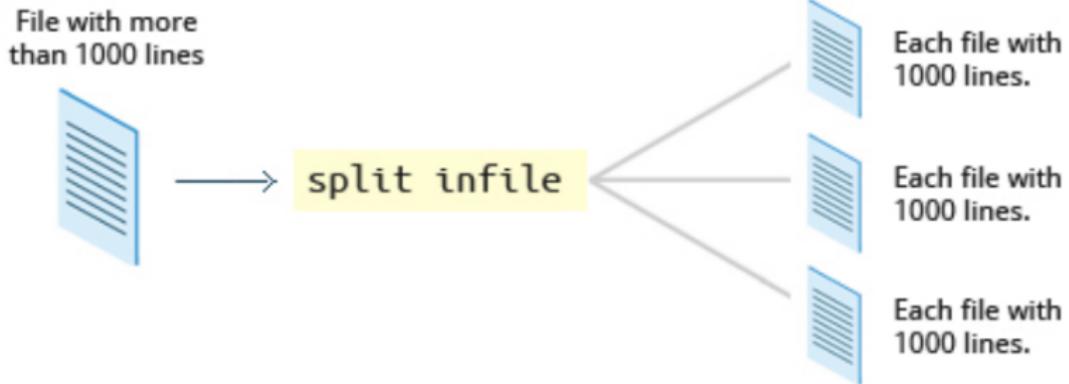
# 文本处理命令 | split



## Introduction

**split** is used to break up (or split) a file into equal-sized segments for easier viewing and manipulation, and is generally used only on relatively large files. By default **split** breaks up a file into 1,000-line segments. The original file remains unchanged, and a set of new files with the same name plus an added prefix is created. By default, the **x** prefix is added. To split a file into segments, use the command `split infile`. To split a file into segments using a different prefix, use the command `split infile <Prefix>`.

# 文本处理命令 | split



## Introduction

**split** is used to break up (or split) a file into equal-sized segments for easier viewing and manipulation, and is generally used only on relatively large files. By default **split** breaks up a file into 1,000-line segments. The original file remains unchanged, and a set of new files with the same name plus an added prefix is created. By default, the **x** prefix is added. To split a file into segments, use the command `split infile`. To split a file into segments using a different prefix, use the command `split infile <Prefix>`.

# 文本处理命令 | tr

The **tr** utility is used to **translate** specified characters into other characters or to delete them. The general syntax is as follows:

```
tr [options] set1 [set2]
```

The items in the square brackets are optional. **tr** requires at least one argument and accepts a maximum of two. The first, designated `set1` in the example, lists the characters in the text to be replaced or removed. The second, `set2`, lists the characters that are to be substituted for the characters listed in the first argument. Sometimes these sets need to be surrounded by apostrophes (or single-quotes (')) in order to have the shell ignore that they mean something special to the shell. It is usually safe (and may be required) to use the single-quotes around each of the sets.

To translate all lower case characters to upper case, at the command prompt type `cat filename | tr a-z A-Z` and press the **Enter** key.



# 文本处理命令 | tr

Command	Usage
\$ tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNPQRSTUVWXYZ	Convert lower case to upper case
\$ tr '{}()' < inputfile > outputfile	Translate braces into parenthesis
\$ echo "This is for testing"   tr [:space:] '\t'	Translate white-space to tabs
\$ echo "This is for testing"   tr -s [:space:]	Squeeze repetition of characters using -s
\$ echo "the geek stuff"   tr -d 't'	Delete specified characters using -d option
\$ echo "my username is 432234"   tr -cd [:digit:]	Complement the sets using -c option
\$ tr -cd [:print:] < file.txt	Remove all non-printable character from a file
\$ tr -s '\n' '' < file.txt	Join all the lines in a file into a single line



# 文本处理命令 | tee

**tee** takes the output from any command, and while sending it to standard output, it also saves it to a file. In other words, it "tees" the output stream from the command: one stream is displayed on the standard output and the other is saved to a file.

For example, to list the contents of a directory on the screen and save the output to a file, at the command prompt type `ls -l | tee newfile` and press the **Enter** key.



# 文本处理命令 | wc

**wc** (word count) counts the number of lines, words, and characters in a file or list of files.

For example, to print the number of lines contained in a file, at the command prompt type `wc -l filename` and press the **Enter** key.

Option	Description
<code>-l</code>	display the number of lines.
<code>-c</code>	display the number of bytes.
<code>-w</code>	display the number of words.

By default all three of these options are active.



# 文本处理命令 | cut

**cut** is used for manipulating column-based files and is designed to extract specific columns. The default column separator is the **tab** character. A different delimiter can be given as a command option.

For example, to display the third column delimited by a blank space, at the command prompt type `ls -l | cut -d" " -f3` and press the **Enter** key.



# 文本处理命令 | less

Directly opening a large file in an editor will cause issues, due to high memory utilization, as an editor will usually try to read the whole file into memory first. However, one can use **less** to view the contents of a large file, scrolling up and down page by page without the system having to place the entire file in memory before starting. This is much faster than using a text editor.

By default, manual (i.e., the **man** command) pages are sent through the **less** command.

```
$ less filename
```

```
$ cat filename | less
```



Searches content  
piece by piece  
in large files.



# 文本处理命令 | head

**head** reads the first few lines of each named file (10 by default) and displays it on standard output. You can give a different number of lines in an option.

For example, If you want to print the first 5 lines from atmtrans.txt, use the following command: `head -n 5 atmtrans.txt`. (You can also just say `head -5 atmtrans.txt`.)



**tail** prints the last few lines of each named file and displays it on standard output. By default, it displays the last 10 lines. You can give a different number of lines as an option. **tail** is especially useful when you are troubleshooting any issue using log files as you probably want to see the most recent lines of output.

For example, to display the last 15 lines of atmtrans.txt, use the following command: `tail -n 15 atmtrans.txt`. (You can also just say `tail -15 atmtrans.txt`.)

To continually monitor new output in a growing log file: `tail -f atmtrans.txt`. This command will continuously display any new lines of output in atmtrans.txt as soon as they appear. Thus it enables you to monitor any current activity that is being reported and recorded.



# 文本处理命令 | nl

nl

number lines of files, 在文本中插入行号。

## 应用情形

- ① 希望在一些数据中永久插入行号，然后进行保存
- ② 希望在命令的输出中临时插入行号，便于理解

## 常用选项

- -v: start, 起始号
- -i: increment, 增量
- -b a: body numbering, all lines, 对所有行编号
- -n: number format, 后跟格式代码 (ln, rn, rz)

## nl

number lines of files, 在文本中插入行号。

## 应用情形

- ① 希望在一些数据中永久插入行号，然后进行保存
- ② 希望在命令的输出中临时插入行号，便于理解

## 常用选项

- -v: start, 起始号
- -i: increment, 增量
- -b a: body numbering, all lines, 对所有行编号
- -n: number format, 后跟格式代码 (ln, rn, rz)

# 文本处理命令 | nl

nl

number lines of files, 在文本中插入行号。

## 应用情形

- ① 希望在一些数据中永久插入行号，然后进行保存
- ② 希望在命令的输出中临时插入行号，便于理解

## 常用选项

- -v: start, 起始号
- -i: increment, 增量
- -b a: body numbering, all lines, 对所有行编号
- -n: number format, 后跟格式代码 (ln, rn, rz)

**strings** is used to extract all printable character strings found in the file or files given as arguments. It is useful in locating human readable content embedded in binary files: for text files one can just use **grep**.

For example, to search for the string `my_string` in a spreadsheet:

```
strings book1.xls | grep my_string.
```



# 文本处理命令 | z Command Family

When working with compressed files many standard commands can not be used directly. For many commonly-used file and text manipulation programs there is also a version especially designed to work directly with compressed files. These associated utilities have the letter **z** prefixed to their name. For example, we have utility programs such as **zcat**, **zless**, **zdiff**, and **zgrep**.

Command	Description
<code>\$ zcat compressed-file.txt.gz</code>	To view a compressed file
<code>\$ zless &lt;filename&gt;.gz</code> or <code>\$ zmore &lt;filename&gt;.gz</code>	To page through a compressed file
<code>\$ zgrep -i less test-file.txt.gz</code>	To search inside a compressed file
<code>\$ zdiff filename1.txt.gz filename2.txt.gz</code>	To compare two compressed files



Note that if you run **zless** on an uncompressed file, it will still work and ignore the decompression stage. There are also equivalent utility programs for other compression methods besides **gzip**; i.e, we have **bzcat** and **bzless** associated with **bzip2**, and **xzcat** and **xzless** associated with **xz**.



# 文本处理命令 | 实例

```
cat ~/.bash_history | cut -d " " -f1 | sort | uniq -c | sort -nr | head
```

空格  
delimiter  
用空格分隔出的第一部分  
为了使uniq起作用，所有的重复行必须是相邻的  
count  
number+reverse

## 解决的问题

找出使用频率最高（排名前十位）的命令，并统计其使用次数



```
cat ~/.bash_history | cut -d " " -f1 | sort | uniq -c | sort -nr | head
```

空格  
delimiter  
用空格分隔出的第一部分  
为了使uniq起作用，所有的重复行必须是相邻的  
count  
number+reverse

## 解决的问题

找出使用频率最高（排名前十位）的命令，并统计其使用次数



# 文本处理命令 | 命令汇总 (1/2)

命令	说明	命令	说明
cat	组合文件	split	分割文件
tac	反转文本行的顺序	rev	反转字符串的顺序
head	从数据开头选择数据行	tail	从数据末尾选择数据行
less	显示文件	more	显示文件
colrm	删除数据列	cmp	比较任意两个文件
cut	抽取数据列/字段	paste	组合数据列
nl	创建行号	wc	统计行/单词/字符数量
fold	格式化行	fmt	格式化段落
sort	排序数据	uniq	查找重复行



# 文本处理命令 | 命令汇总 (2/2)

命令	说明	命令	说明
join	合并两个文件中的有序数据	strings	在二进制文件中搜索字符串
tr	转换字符	tsort	由偏序创建全序
comm	比较有序文本文件	diff	比较无序文本文件
expand	将制表符转换成空格	unexpand	将空格转换成制表符
tee	管道分流	pr	按页/列格式化文本
grep	选取包含特定模式的行	look	查找以特定模式开头的行/单词
sed	非交互式文本编辑	awk	格式化输出
shuf	随机选取行/文件		



1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题



## 简单定义

- sed : 处理纯文本流的文本编辑器
- AWK : 一种输出格式化语言

## 处理对象

对已有文本进行操作：

- 管理文件（如/etc/passwd）的内容
- 另一个命令（如 ls）的输出
- 冗长的文本文件（如 FASTA 格式的序列）的内容



## 简单定义

- sed : 处理纯文本流的文本编辑器
- AWK : 一种输出格式化语言

## 处理对象

对已有文本进行操作：

- 管理文件（如/etc/passwd）的内容
- 另一个命令（如 ls）的输出
- 冗长的文本文件（如 FASTA 格式的序列）的内容



1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed

- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结

- 思考题



## 简介

sed(Stream EDitor) 是一个精简的、非交互式的流编辑器，它根据用户预先设置的规则来操作指定的文本流。

## 工作原理

逐行读取文件内容存储在临时缓冲区中，称为“模式空间”（pattern space），接着用 sed 命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一行，这样不断重复，直到文件末尾。原文件内容并没有改变。

## 选项

- e Expression, 指定多个编辑命令
- f File, 指定 sed 命令脚本文件
- n 阻止输入行自动输出（只有经过 sed 处理过的行才显示出来，其他不显示）

## 简介

sed(Stream EDitor) 是一个精简的、非交互式的流编辑器，它根据用户预先设置的规则来操作指定的文本流。

## 工作原理

逐行读取文件内容存储在临时缓冲区中，称为“模式空间”(pattern space)，接着用 sed 命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一行，这样不断重复，直到文件末尾。原文件内容并没有改变。

## 选项

- e Expression, 指定多个编辑命令
- f File, 指定 sed 命令脚本文件
- n 阻止输入行自动输出（只有经过 sed 处理过的行才显示出来，其他不显示）

## 简介

sed(Stream EDitor) 是一个精简的、非交互式的流编辑器，它根据用户预先设置的规则来操作指定的文本流。

## 工作原理

逐行读取文件内容存储在临时缓冲区中，称为“模式空间”(pattern space)，接着用 sed 命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一行，这样不断重复，直到文件末尾。原文件内容并没有改变。

## 选项

- e Expression, 指定多个编辑命令
- f File, 指定 sed 命令脚本文件
- n 阻止输入行自动输出（只有经过 sed 处理过的行才显示出来，其他不显示）

# sed | 使用方法

Command	Usage
<code>sed -e command &lt;filename&gt;</code>	Specify editing commands at the command line, operate on file and put the output on standard out (e.g., the terminal)
<code>sed -f scriptfile &lt;filename&gt;</code>	Specify a scriptfile containing sed commands, operate on file and put output on standard out.



# sed | 使用方法 | 替换

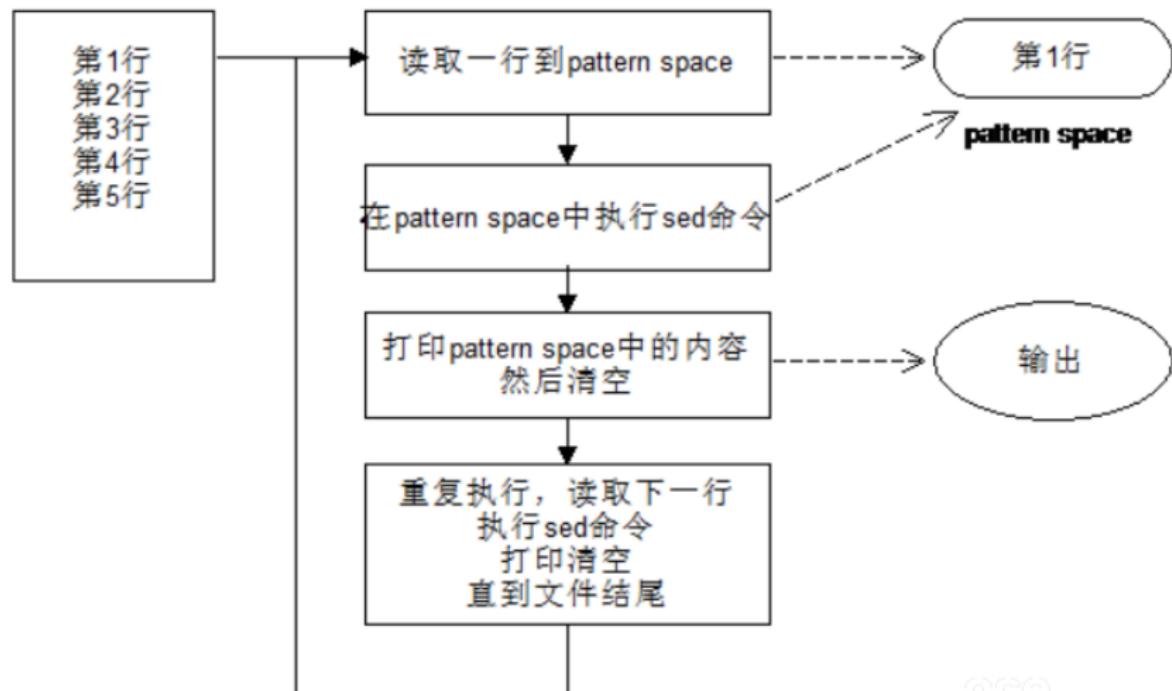
Command	Usage
<code>sed s/pattern/replace_string/ file</code>	Substitute first string occurrence in a line
<code>sed s/pattern/replace_string/g file</code>	Substitute all string occurrences in a line
<code>sed 1,3s/pattern/replace_string/g file</code>	Substitute all string occurrences in a range of lines
<code>sed -i s/pattern/replace_string/g file</code>	Save changes for string substitution in the same file

You must use the `-i` option with care, because the action is not reversible. It is always safer to use `sed` without the `-i` option and then replace the file yourself, as shown in the following example:

```
sed s/pattern/replace_string/g file > file2
```



# sed | 工作原理



## sed 命令

- `sed s/Paul/Pablo/g names1.txt > names3.txt`
- `sed 's/Paul/Pablo/g;s/Pat/Patricia/g' names1.txt names2.txt`
- `sed -e 's/Paul/Pablo/g' -e 's/Pat/Patricia/g' names1.txt names2.txt`
- `sed -f edits.sedscr names1.txt > names3.txt`

## 命令解释

- 把 Paul 替换为 Pablo ; g 表示进行全局查找和替换
- 把 Paul 和 Pat 分别替换为 Pablo 和 Patricia
- 效果同上，写法不同
- 根据 edits.sedscr 文件中的命令进行 sed 操作

## sed 命令

- `sed s/Paul/Pablo/g names1.txt > names3.txt`
- `sed 's/Paul/Pablo/g;s/Pat/Patricia/g' names1.txt names2.txt`
- `sed -e 's/Paul/Pablo/g' -e 's/Pat/Patricia/g' names1.txt names2.txt`
- `sed -f edits.sedscr names1.txt > names3.txt`

## 命令解释

- 把 Paul 替换为 Pablo ; g 表示进行全局查找和替换
- 把 Paul 和 Pat 分别替换为 Pablo 和 Patricia
- 效果同上，写法不同
- 根据 edits.sedscr 文件中的命令进行 sed 操作

## sed 命令

- `sed s/Paul/Pablo/g names1.txt > names3.txt`
- `sed 's/Paul/Pablo/g;s/Pat/Patricia/g' names1.txt names2.txt`
- `sed -e 's/Paul/Pablo/g' -e 's/Pat/Patricia/g' names1.txt names2.txt`
- `sed -f edits.sedscr names1.txt > names3.txt`

## 命令解释

- 把 Paul 替换为 Pablo ; g 表示进行全局查找和替换
- 把 Paul 和 Pat 分别替换为 Pablo 和 Patricia
- 效果同上，写法不同
- 根据 edits.sedscr 文件中的命令进行 sed 操作

## sed 命令

- `sed s/Paul/Pablo/g names1.txt > names3.txt`
- `sed 's/Paul/Pablo/g;s/Pat/Patricia/g' names1.txt names2.txt`
- `sed -e 's/Paul/Pablo/g' -e 's/Pat/Patricia/g' names1.txt names2.txt`
- `sed -f edits.sedscr names1.txt > names3.txt`

## 命令解释

- 把 Paul 替换为 Pablo ; g 表示进行全局查找和替换
- 把 Paul 和 Pat 分别替换为 Pablo 和 Patricia
- 效果同上，写法不同
- 根据 edits.sedscr 文件中的命令进行 sed 操作

## sed 命令

- `sed s/Paul/Pablo/g names1.txt > names3.txt`
- `sed 's/Paul/Pablo/g;s/Pat/Patricia/g' names1.txt names2.txt`
- `sed -e 's/Paul/Pablo/g' -e 's/Pat/Patricia/g' names1.txt names2.txt`
- `sed -f edits.sedscr names1.txt > names3.txt`

## 命令解释

- 把 Paul 替换为 Pablo ; g 表示进行全局查找和替换
- 把 Paul 和 Pat 分别替换为 Pablo 和 Patricia
- 效果同上，写法不同
- 根据 edits.sedscr 文件中的命令进行 sed 操作

## sed 命令

- `sed s/Paul/Pablo/g names1.txt > names3.txt`
- `sed 's/Paul/Pablo/g;s/Pat/Patricia/g' names1.txt names2.txt`
- `sed -e 's/Paul/Pablo/g' -e 's/Pat/Patricia/g' names1.txt names2.txt`
- `sed -f edits.sedscr names1.txt > names3.txt`

## 命令解释

- 把 Paul 替换为 Pablo ; g 表示进行全局查找和替换
- 把 Paul 和 Pat 分别替换为 Pablo 和 Patricia
- 效果同上，写法不同
- 根据 edits.sedscr 文件中的命令进行 sed 操作

## sed 命令

- `sed s/Paul/Pablo/g names1.txt > names3.txt`
- `sed 's/Paul/Pablo/g;s/Pat/Patricia/g' names1.txt names2.txt`
- `sed -e 's/Paul/Pablo/g' -e 's/Pat/Patricia/g' names1.txt names2.txt`
- `sed -f edits.sedscr names1.txt > names3.txt`

## 命令解释

- 把 Paul 替换为 Pablo ; g 表示进行全局查找和替换
- 把 Paul 和 Pat 分别替换为 Pablo 和 Patricia
- 效果同上，写法不同
- 根据 edits.sedscr 文件中的命令进行 sed 操作

## sed 命令

- `sed s/Paul/Pablo/g names1.txt > names3.txt`
- `sed 's/Paul/Pablo/g;s/Pat/Patricia/g' names1.txt names2.txt`
- `sed -e 's/Paul/Pablo/g' -e 's/Pat/Patricia/g' names1.txt names2.txt`
- `sed -f edits.sedscr names1.txt > names3.txt`

## 命令解释

- 把 Paul 替换为 Pablo ; g 表示进行全局查找和替换
- 把 Paul 和 Pat 分别替换为 Pablo 和 Patricia
- 效果同上，写法不同
- 根据 edits.sedscr 文件中的命令进行 sed 操作

## sed 命令

- `sed s/Paul/Pablo/g names1.txt > names3.txt`
- `sed 's/Paul/Pablo/g;s/Pat/Patricia/g' names1.txt names2.txt`
- `sed -e 's/Paul/Pablo/g' -e 's/Pat/Patricia/g' names1.txt names2.txt`
- `sed -f edits.sedscr names1.txt > names3.txt`

## 命令解释

- 把 Paul 替换为 Pablo ; g 表示进行全局查找和替换
- 把 Paul 和 Pat 分别替换为 Pablo 和 Patricia
- 效果同上，写法不同
- 根据 edits.sedscr 文件中的命令进行 sed 操作

表10-1 使用**sed**在文件中定位文本的方式

x	x为一行号，如1
x,y	表示行号范围从x到y，如2, 5表示从第2行到第5行
/pattern/	查询包含模式的行。例如/disk/或/[a-z]/
/pattern/pattern/	查询包含两个模式的行。例如/disk/disks/
pattern/,x	在给定行号上查询包含模式的行。如/ribbon/,3
x,/pattern/	通过行号和模式查询匹配行。3./vdu/
x,y!	查询不包含指定行号x和y的行。1,2!

<http://blog.csdn.net/huang2009303513>



命令	说明
n	第 n 行
\$	最后一行
m, n	从第 m 行到第 n 行
/pattern/	包含指定模式的行
/pattern/, n	从包含指定模式的行到第 n 行
n, /pattern/	从第 n 行到包含指定模式的行
/pattern1/, /pattern2/	从包含模式 1 的行到包含模式 2 的行
!	反向选择，不包含指定行，如 m, n!



p	打印匹配行
=	显示文件行号
a\	在定位行号后附加新文本信息
i\	在定位行号后插入新文本信息
d	删除定位行
c\	用新文本替换定位文本
s	使用替换模式替换相应模式
r	从另一个文件中读文本
w	写文本到一个文件
q	第一个模式匹配完成后推出或立即推出
	显示与八进制 ASCII 代码等价的控制字符
{}	在定位行执行的命令组
n	从另一个文件中读文本下一行，并附加在下一行
g	将模式2粘贴到 pattern n/
y	传送字符
\n	延续到下一输入行; 允许跨行的模式匹配语句



命令	助记	说明
p	Print	打印匹配行
=	—	显示匹配行的行号
d	Delete	删除匹配行
a\	Append	在指定行后面追加文本
i\	Insert	在指定行后面插入文本
c\	Correct	用新文本替换指定行
s	Substitute	替换命令
l	List	显示指定行中的所有字符
r	Read	读取文件
w	Write	写入文件
n	Next	读取指定行的下一行
q	Quit	退出 sed



## ◆ p : 打印匹配行

```
sed -n '3,5p' test.in // ('3,5!p')
```

```
sed -n '$p' test.in
```

```
sed -n '/north/p' test.in
```

## ◆ = : 显示匹配行的行号

```
sed -n '/north/=/' test.in
```

## ◆ d : 删除匹配的行

```
sed '/north/d' test.in
```



- ◆ **a\**：在指定行后面追加一行或多行文本，并显示添加的新内容，该命令主要用于 sed 脚本中。

```
sed '/north/a\AAA\  
>BBB\  
>CCC' test
```

- ◆ **i\**：在指定行前插入一行或多行，并显示添加的新内容，使用格式同 a\
- ◆ **c\**：用新文本替换指定的行，使用格式同 a\
- ◆ **l**：显示指定行中所有字符，包括控制字符(非打印字符)

```
sed -n '/north/l' test.in
```



## ◆ s : 替换命令，使用格式为：

```
[address] s/old/new/ [gpw]
```

- **address** : 如果省略，表示编辑所有的行。
- **g** : 全局替换
- **p** : 打印被修改后的行
- **w fname** : 将被替换后的行内容写到指定的文件中  
[http://blog.csdn.net/jnu\\_simba](http://blog.csdn.net/jnu_simba)

```
sed -n 's/north/NORTH/gp' test.in
```

```
sed -n 's/north/NORTH/w data' test.in
```

```
sed 's/[0-9][0-9]$/&.5/' datafile
```

& 符号用在替换字符串中时，代表 被替换的字符串



- ◆ r : 读文件，将另外一个文件中的内容附加到指定行后。

```
sed '$r data' test.in
```

- ◆ w : 写文件，将指定行写入到另外一个文件中。

```
sed -n '/public/w data2' test.in
```

- ◆ n : 将指定行的下面一行读入编辑缓冲区。

```
sed -n '/public/{n;s/north/NORTH/p}' test.in
```

对指定行同时使用多个 sed 编辑命令时，需用大括号 “ {} ” 括起来，命令之间用分号 “ ; ” 格开。注意与 -e 选项的区别

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

## 具体问题

- 输出文件除 1-3 行之外的行
- 输出匹配 “the” 的行，并且输出每行行号
- 将 “this” 全部替换成 “that”
- 添加文件扩展名
- 输出 ls -l 结果中的第 1-3 行

## sed 命令

- sed -n '1,3!p' 123.txt
- sed -n -e '/the/p' -e '/the/=' 123.txt
- sed 's>this>that' 123.txt
- echo "hello" | sed 's/\$/.txt/g'
- ls -l | sed -n '1,3p'

To convert 01/02/... to JAN/FEB/...

```
sed -e 's/01/JAN/' -e 's/02/FEB/' -e 's/03/MAR/' \
-e 's/04/APR/' -e 's/05/MAY/' -e 's/06/JUN/' \
-e 's/07/JUL/' -e 's/08/AUG/' -e 's/09/SEP/' \
-e 's/10/OCT/' -e 's/11/NOV/' -e 's/12/DEC/'
```



1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题



## 简介

AWK 是用于模式匹配的一种成熟的、结构化的、解释性的编程语言，用于处理数据和生成报告。

## 工作原理

AWK 逐行扫描输入（可以是文件或管道等），按给定的模式查找出匹配的行，然后对这些行执行 AWK 命令指定的操作。与 sed 一样，AWK 不会修改输入文件的内容。可以使用重定向将 AWK 的输出保存到文件中。

## 选项

- F Field, 指定输入记录字段的分隔符，默认使用环境变量 IFS 的值
- f File, 指定 AWK 命令脚本文件

## 简介

AWK 是用于模式匹配的一种成熟的、结构化的、解释性的编程语言，用于处理数据和生成报告。

## 工作原理

AWK 逐行扫描输入（可以是文件或管道等），按给定的模式查找出匹配的行，然后对这些行执行 AWK 命令指定的操作。与 sed 一样，AWK 不会修改输入文件的内容。可以使用重定向将 AWK 的输出保存到文件中。

## 选项

- F Field, 指定输入记录字段的分隔符，默认使用环境变量 IFS 的值
- f File, 指定 AWK 命令脚本文件

## 简介

AWK 是用于模式匹配的一种成熟的、结构化的、解释性的编程语言，用于处理数据和生成报告。

## 工作原理

AWK 逐行扫描输入（可以是文件或管道等），按给定的模式查找出匹配的行，然后对这些行执行 AWK 命令指定的操作。与 sed 一样，AWK 不会修改输入文件的内容。可以使用重定向将 AWK 的输出保存到文件中。

## 选项

- F Field, 指定输入记录字段的分隔符，默认使用环境变量 IFS 的值
- f File, 指定 AWK 命令脚本文件

## Introduction

awk is used to extract and then print specific contents of a file and is often used to construct reports. It was created at Bell Labs in the 1970s and derived its name from the last names of its authors: Alfred Aho, Peter Weinberger, and Brian Kernighan.

## Features

- It is a powerful utility and interpreted programming language.
- It is used to manipulate data files, retrieving, and processing text.
- It works well with **fields** (containing a single piece of data, essentially a column) and **records** (a collection of fields, essentially a line in a file).



## Introduction

awk is used to extract and then print specific contents of a file and is often used to construct reports. It was created at Bell Labs in the 1970s and derived its name from the last names of its authors: Alfred Aho, Peter Weinberger, and Brian Kernighan.

## Features

- It is a powerful utility and interpreted programming language.
- It is used to manipulate data files, retrieving, and processing text.
- It works well with **fields** (containing a single piece of data, essentially a column) and **records** (a collection of fields, essentially a line in a file).



# AWK | 使用方法

Command	Usage
<code>awk 'command' var=value file</code>	Specify a command directly at the command line
<code>awk -f scriptfile var=value file</code>	Specify a file that contains the script to be executed along with <code>f</code>
Command	Usage
<code>awk '{ print \$0 }' /etc/passwd</code>	Print entire file
<code>awk -F: '{ print \$1 }' /etc/passwd</code>	Print first field (column) of every line, separated by a space
<code>awk -F: '{ print \$1 \$6 }' /etc/passwd</code>	Print first and sixth field of every line



# AWK | 脚本结构

```
1 awk  
2 '  
3 BEGIN {actions}  
4   /pattern1/{actions}  
5   .....  
6   /patternN/{actions}  
7  
8 END {actions}  
9 '  
10  
11  
12  
13 InputFile
```



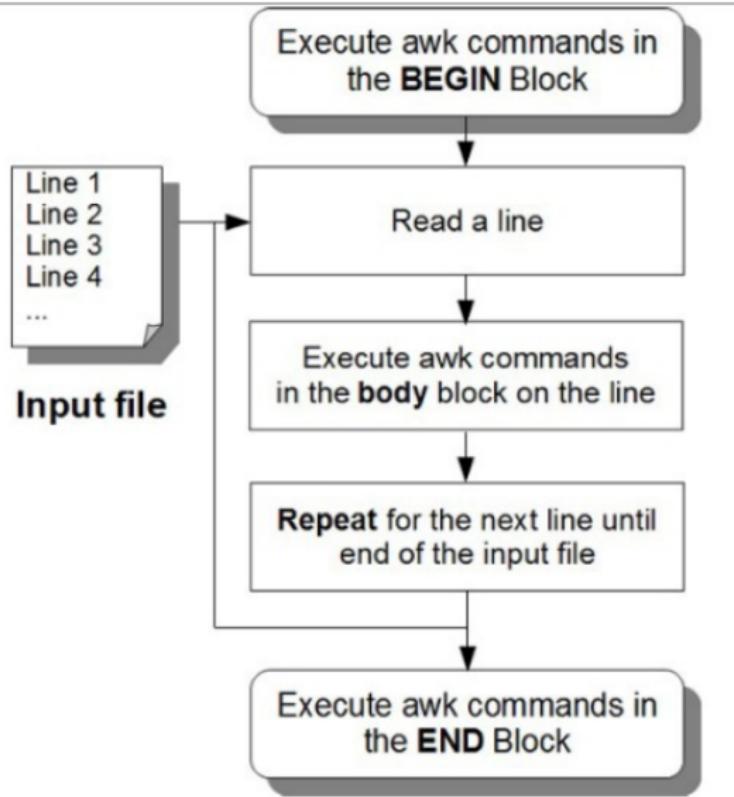
- BEGIN{actions} 和 END{actions} 是可选的
- /pattern/ 和 {actions} 可以省略，但不能同时省略
  - /pattern/ 省略时表示对所有的输入行执行指定的 actions
  - {actions} 省略时表示打印整行



# AWK | 脚本结构 | 实例

```
1 awk
2 '
3 BEGIN { FS=":" }
4 {printf "username: " $1 "\t\t\t user id: "
5   $3}
6
7 END {printf "All done processing /etc/
8 passwd"}
9 '
10
11 /etc/passwd
```





- ① 如果存在 **BEGIN** , **awk** 首先执行它指定的 **actions**
- ② **awk** 从输入中读取一行，称为一条输入记录
- ③ **awk** 将读入的记录分割成数个字段，并将第一个字段放入变量 **\$1** 中，第二个放入变量 **\$2** 中，以此类推；**\$0** 表示整条记录；字段分隔符可以通过选项 **-F** 指定，否则使用缺省的分隔符。
- ④ 把当前输入记录依次与每一个 **awk\_cmd** 中 **pattern** 比较：  
如果相匹配，就执行对应的 **actions**；  
如果不匹配，就跳过对应的 **actions**，直到完成所有的 **awk\_cmd**
- ⑤ 当一条输入记录处理完毕后，**awk** 读取输入的下一行，重复上面的处理过程，直到所有输入全部处理完毕。
- ⑥ **awk** 处理完所有的输入后，若存在 **END**，执行相应的 **actions**
- ⑦ 如果输入是文件列表，**awk** 将按顺序处理列表中的每个文件。



变量	描述
\$n	当前记录的第n个字段，字段间由FS分隔。
\$0	完整的输入记录。
ARGC	命令行参数的数目。
ARGIND	命令行中当前文件的位置(从0开始算)。
ARGV	包含命令行参数的数组。
CONVFMT	数字转换格式(默认值为%.6g)
ENVIRON	环境变量关联数组。
ERRNO	最后一个系统错误的描述。
FIELDWIDTHS	字段宽度列表(用空格键分隔)。
FILENAME	当前文件名。



FNR	同NR，但相对于当前文件。
FS	字段分隔符(默认是任何空格)。
IGNORECASE	如果为真，则进行忽略大小写的匹配。
NF	当前记录中的字段数。
NR	当前记录数。
OFMT	数字的输出格式(默认值是%.6g)。
OFS	输出字段分隔符(默认值是一个空格)。
ORS	输出记录分隔符(默认值是一个换行符)。
RLENGTH	由match函数所匹配的字符串的长度。
RS	记录分隔符(默认是一个换行符)。
RSTART	由match函数所匹配的字符串的第一个位置。
SUBSEP	数组下标分隔符(默认值是\034)。



## 命令构成

### ① 编辑命令（一个、一组命令或一个命令文件）

- ① 模式：只编辑与模式相匹配的记录行；没有提供模式时，匹配所有行
- ② 命令：具体执行的编辑命令；没有指定命令时，打印整个记录行

### ② 要编辑的数据（数据或数据文件）



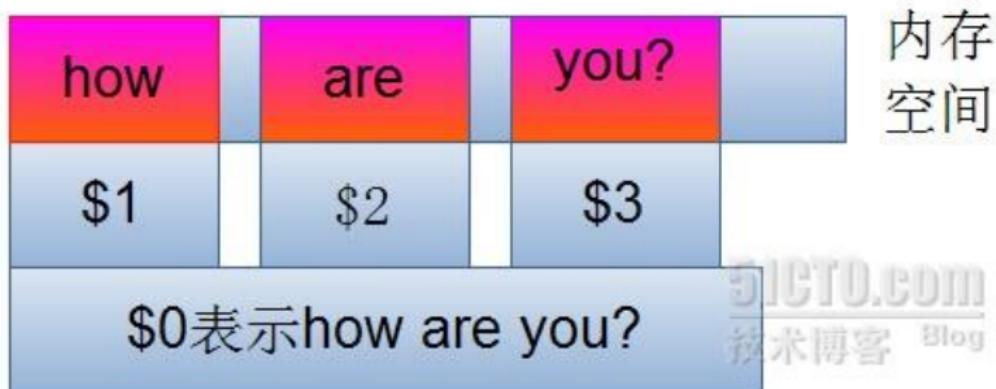
# AWK | 命令 | 记录与字段

	字段 1 (名字)	字段 2 (姓)	字段 3 (报酬率)	字段 4 (小时数)
记录 2	Susan	White	6.00	23
	Mark	Eagle	6.25	40
记录 4	Tuan	Nguyen	7.89	44
	Dan	Black	7.23	40
	Amanda	Trapp	6.95	40
	Brian	Devaux	7.95	0
	Chris	Walljasper	6.89	32
	Mary	Lamb	8.22	40
记录 10	Jackie	Kammaoto	7.59	40
	Nicky	Barber	6.35	40

有 10 个记录的文件，每个记录 4 个字段



假如文本有一行内容 how are you?



## □ 字段分隔符

`awk` 中的字段分隔符可以用 `-F` 选项指定，缺省是空格。

```
awk '{print $1}' test.in
```

```
awk -F: '{print $1}' test.in
```

```
awk -F'[:\t]' '{print $1}' test.in
```

## □ 重定向与管道

```
awk '{print $1, $2 > "output"}' test.in
```

```
awk 'BEGIN{"cat" | getline a; print a}'
```



② 使用布尔(比较)表达式，表达式的值为真时执行相应的操作(actions)

- 表达式中可以使用变量(如字段变量 \$1, \$2 等)和 /rexp/
- 表达式中的运算符有
  - 关系运算符: < > <= >= == !=
  - 匹配运算符:  
x ~ /rexp/ 如果 x 匹配 /rexp/，则返回真；  
x !~ /rexp/ 如果 x 不匹配 /rexp/，则返回真。

```
awk '$1 > 20 {print $0}' test.in
```

```
awk '$2 ~ /^6/ {print $0}' test.in
```



- 复合表达式: `&&`(逻辑与)、`||`(逻辑或)、`!`(逻辑非)

`expr1 && expr2` 两个表达式的值都为真时，返回真

`expr1 || expr2` 两个表达式中有一个的值为真时，返回真

`!expr` 表达式的值为假时，返回真

```
awk '$1<20 && $2~/^6/' {print $0}' test.in
```

```
http://blog.csdn.net/jnu_simba
```

```
awk '$1<20 || $2~/^6/' {print $0}' test.in
```

```
awk '!($2~/^6/){print $0}' test.in
```

```
awk '/^#/ && ${#$/ { print }' test.in
```

注：表达式中有比较运算时，一般用圆括号括起来

# AWK | 命令 | 转义序列

转义序列	说明
\t	制表符, Tab 键
\n	换行, 新行
\r	回车键
\'	单引号
\"	双引号
\\"	反斜线
\f	走纸换页, 新页
\v	垂直制表符
\a	警报, 警钟
\b	退格键



# AWK | 脚本结构 | 实例

```
1 awk
2 '
3 BEGIN {FS=":"}
4 {printf "username: " $1 "\t\t\t user id: "
5   $3}
6
7 END {printf "All done processing /etc/
8 passwd"}
9 '
10
11 /etc/passwd
```



## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS="" ;OFS="\t"}' {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: '\$length(\$2)==0 {print \$1}' /etc/shadow

## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS=" ";OFS="\t"} {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: '\$length(\$2)==0 {print \$1}' /etc/shadow

## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS="" ;OFS="\t"} {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: '\$length(\$2)==0 {print \$1}' /etc/shadow

## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS=" ";OFS="\t"} {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: '{length(\$2)==0 {print \$1}}' /etc/shadow

## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS=" ";OFS="\t"} {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: 'length(\$2)==0 {print \$1}' /etc/shadow

## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS=" ";OFS="\t"} {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: 'length(\$2)==0 {print \$1}' /etc/shadow

## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS=" ";OFS="\t"} {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: 'Length(\$2)==0 {print \$1}' /etc/shadow

## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS=" ";OFS="\t"} {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: 'length(\$2)==0 {print \$1}' /etc/shadow

## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS=" ";OFS="\t"} {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: 'length(\$2)==0 {print \$1}' /etc/shadow

## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS=" ";OFS="\t"} {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: 'length(\$2)==0 {print \$1}' /etc/shadow

## 具体问题

- 输出 ls -l 中非文件夹的完整信息
- 给文本的每一行添加行号
- 将文本中的空格换成制表符
- 检测系统中 UID 为 0 的用户
- 检测系统中密码为空的用户

## 命令

- ls -l | awk '{if(\$1 !~ /^d/) {print \$0}}'
- awk '{printf("%03d %s\n",NR,\$0)}' ori.txt > dst.txt
- awk 'BEGIN{FS=" ";OFS="\t"} {print \$1,\$2,\$3}' ori.txt > dst.txt
- awk -F: '\$3==0 {print \$1}' /etc/passwd
- awk -F: 'length(\$2)==0 {print \$1}' /etc/shadow

1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题



# Useful bash one-liners for bioinformatics

## 具体问题

- Count FASTA record
- Split a multi-FASTA file into individual FASTA files
- Convert a FASTQ file to FASTA
- Returns all lines on Chr 1 between 1MB and 2MB in file.txt

## 命令

- grep '>' file.fasta | wc -l
- awk '/^>/(s=t+d".fa") {print > s}' multi.fa
- sed -n '1~4s/^@/>/p;2~4p' file.fq > file.fa
- cat file.txt | awk '\$1=="1"' | awk '\$3>=1000000' | awk '\$3<=2000000'

# Useful bash one-liners for bioinformatics

## 具体问题

- Count FASTA record
- Split a multi-FASTA file into individual FASTA files
- Convert a FASTQ file to FASTA
- Returns all lines on Chr 1 between 1MB and 2MB in file.txt

## 命令

- grep '>' file.fasta | wc -l
- awk '/^>/{s=++d".fa"} {print > s}' multi.fa
- sed -n '1~4s/^@/>/p;2~4p' file.fq > file.fa
- cat file.txt | awk '\$1=="1"' | awk '\$3>=1000000' | awk '\$3<=2000000'

# Useful bash one-liners for bioinformatics

## 具体问题

- Count FASTA record
- Split a multi-FASTA file into individual FASTA files
- Convert a FASTQ file to FASTA
- Returns all lines on Chr 1 between 1MB and 2MB in file.txt

## 命令

- grep '>' file.fasta | wc -l
- awk '/^>/{s=++d".fa"} {print > s}' multi.fa
- sed -n '1~4s/^@/>/p;2~4p' file.fq > file.fa
- cat file.txt | awk '\$1=="1"' | awk '\$3>=1000000' | awk '\$3<=2000000'

# Useful bash one-liners for bioinformatics

## 具体问题

- Count FASTA record
- Split a multi-FASTA file into individual FASTA files
- Convert a FASTQ file to FASTA
- Returns all lines on Chr 1 between 1MB and 2MB in file.txt

## 命令

- grep '>' file.fasta | wc -l
- awk '/^>/{s=++d".fa"} {print > s}' multi.fa
- sed -n '1~4s/^@/>/p;2~4p' file.fq > file.fa
- cat file.txt | awk '\$1=="1"' | awk '\$3>=1000000' | awk '\$3<=2000000'

# Useful bash one-liners for bioinformatics

## 具体问题

- Count FASTA record
- Split a multi-FASTA file into individual FASTA files
- Convert a FASTQ file to FASTA
- Returns all lines on Chr 1 between 1MB and 2MB in file.txt

## 命令

- grep '>' file.fasta | wc -l
- awk '/^>/{s=++d".fa"} {print > s}' multi.fa
- sed -n '1~4s/^@/>/p;2~4p' file.fq > file.fa
- cat file.txt | awk '\$1=="1"' | awk '\$3>=1000000' | awk '\$3<=2000000'

# Useful bash one-liners for bioinformatics

## 具体问题

- Count FASTA record
- Split a multi-FASTA file into individual FASTA files
- Convert a FASTQ file to FASTA
- Returns all lines on Chr 1 between 1MB and 2MB in file.txt

## 命令

- grep '>' file.fasta | wc -l
- awk '/^>/{s=++d".fa"} {print > s}' multi.fa
- sed -n '1~4s/^@/>/p;2~4p' file.fq > file.fa
- cat file.txt | awk '\$1=="1"' | awk '\$3>=1000000' | awk '\$3<=2000000'

# Useful bash one-liners for bioinformatics

## 具体问题

- Count FASTA record
- Split a multi-FASTA file into individual FASTA files
- Convert a FASTQ file to FASTA
- Returns all lines on Chr 1 between 1MB and 2MB in file.txt

## 命令

- grep '>' file.fasta | wc -l
- awk '/^>/{s=++d".fa"} {print > s}' multi.fa
- sed -n '1~4s/^@/>/p;2~4p' file.fq > file.fa
- cat file.txt | awk '\$1=="1"' | awk '\$3>=1000000' | awk '\$3<=2000000'

# Useful bash one-liners for bioinformatics

## 具体问题

- Count FASTA record
- Split a multi-FASTA file into individual FASTA files
- Convert a FASTQ file to FASTA
- Returns all lines on Chr 1 between 1MB and 2MB in file.txt

## 命令

- grep '>' file.fasta | wc -l
- awk '/^>/{s=++d".fa"} {print > s}' multi.fa
- sed -n '1~4s/^@/>/p;2~4p' file.fq > file.fa
- cat file.txt | awk '\$1=="1"' |  
awk '\$3>=1000000' | awk '\$3<=2000000'

# Useful bash one-liners for bioinformatics

## 具体问题

- Count FASTA record
- Split a multi-FASTA file into individual FASTA files
- Convert a FASTQ file to FASTA
- Returns all lines on Chr 1 between 1MB and 2MB in file.txt

## 命令

- grep '>' file.fasta | wc -l
- awk '/^>/{s=++d".fa"} {print > s}' multi.fa
- sed -n '1~4s/^@/>/p;2~4p' file.fq > file.fa
- cat file.txt | awk '\$1=="1"' |  
awk '\$3>=1000000' | awk '\$3<=2000000'

# Useful bash one-liners for bioinformatics

## 具体问题

- Print all possible 3mer DNA sequence combinations
- Determine the number of genes annotated in a GFF3 file
- Determine all feature types annotated in a GFF3 file
- Basic FASTQ sequence statistics(number,percentage,...)

## 命令

- `echo {A,C,G,T}{A,C,G,T}{A,C,G,T}`
- `grep -c '\tgene\t' yourannots.gff3`
- `grep -v '^#' GFF3 | cut -s -f 3 | sort | uniq`
- `cat myfile.fq | awk '{ if((NR-2)%4==0) {read=$1; total++; count[read]++} END{for(read in count){if((max+count[read])>max){(max=count[read]); maxRead=read}}; if(count[read]==1){unique+=1}}}' | sort -n | head -100 | awk '{print $1"\t"$2"\t"$3"\t"$4"\t"$5"\t"$6"\t"$7"\t"$8"\t"$9"\t"$10}' > myoutput.txt`

# Useful bash one-liners for bioinformatics

## 具体问题

- Print all possible 3mer DNA sequence combinations
- Determine the number of genes annotated in a GFF3 file
- Determine all feature types annotated in a GFF3 file
- Basic FASTQ sequence statistics(number,percentage,...)

## 命令

- echo {A,C,G,T}{A,C,G,T}{A,C,G,T}
- grep -c '\tgene\t' yourannots.gff3
- grep -v '^#' GFF3 | cut -s -f 3 | sort | uniq
- cat myfile.fq | awk '{ if((NR-2)%4==0) {read=\$1; total++; count[read]++} END{for(read in count){if((maxcount[read]>max) - (maxcount[read]-maxRead-read))/10(count[read]-1) > unique+0.001){unique+=1}}}'

# Useful bash one-liners for bioinformatics

## 具体问题

- Print all possible 3mer DNA sequence combinations
- Determine the number of genes annotated in a GFF3 file
- Determine all feature types annotated in a GFF3 file
- Basic FASTQ sequence statistics(number,percentage,...)

## 命令

- echo {A,C,G,T}{A,C,G,T}{A,C,G,T}
- grep -c '\tgene\t' yourannots.gff3
- grep -v '^#' GFF3 | cut -s -f 3 | sort | uniq
- cat myfile.fq | awk '{read=(NR-2)\*4+0) (read>0) {total++;count[read]+=(NR-2)\*4+0) (read>0) {maxCount=read; count[read]=1} (unique+=1) (maxCount<read) {maxCount=read; count[read]=1} (unique+=1) } END{print "Total reads: " total, "Unique reads: " unique, "Max read length: " maxCount}'

# Useful bash one-liners for bioinformatics

## 具体问题

- Print all possible 3mer DNA sequence combinations
- Determine the number of genes annotated in a GFF3 file
- Determine all feature types annotated in a GFF3 file
- Basic FASTQ sequence statistics(number,percentage,...)

## 命令

- echo {A,C,G,T}{A,C,G,T}{A,C,G,T}
- grep -c '\tgene\t' yourannots.gff3
- grep -v '^#' GFF3 | cut -s -f 3 | sort | uniq
- cat myfile.fq | awk '{read=(NR-2)\*4+0) (read>0) {total++;count[read]+=(NR-2)\*4+0) (read>0) {maxCount[read]=read; count[read]=1} (unique+1) (maxCount[read]>max) {max=unique[read]; maxCount[read]=read} }' | sort | uniq

# Useful bash one-liners for bioinformatics

## 具体问题

- Print all possible 3mer DNA sequence combinations
- Determine the number of genes annotated in a GFF3 file
- Determine all feature types annotated in a GFF3 file
- Basic FASTQ sequence statistics(number,percentage,...)

## 命令

- echo {A,C,G,T}{A,C,G,T}{A,C,G,T}
- grep -c '\tgene\t' yourannots.gff3
- grep -v '^#' GFF3 | cut -s -f 3 | sort | uniq
- cat myfile.tq | awk '{read=(NR-2)\*4+0) (read>0) {total++;count[read]++} END{for(read in count){if((max1) < count[read]) {max= count[read]; maxRead=read} ;if(count[read]==1) {unique+=1}}}'

# Useful bash one-liners for bioinformatics

## 具体问题

- Print all possible 3mer DNA sequence combinations
- Determine the number of genes annotated in a GFF3 file
- Determine all feature types annotated in a GFF3 file
- Basic FASTQ sequence statistics(number,percentage,...)

## 命令

- echo {A,C,G,T}{A,C,G,T}{A,C,G,T}
- grep -c '\tgene\t' yourannots.gff3
- grep -v '^#' GFF3 | cut -s -f 3 | sort | uniq
- cat myfile.fq | awk '((NR-2)%4==0){read=\$1;total++;count[read]++;}END{for(read in count){if(!max){count[read]>max}{max=count[read];maxRead=read};if(count[read]==1){unique+=1}};print total,unique,unique\*100/total,maxRead,count[maxRead],count[maxRead]\*100/total}'

# Useful bash one-liners for bioinformatics

## 具体问题

- Print all possible 3mer DNA sequence combinations
- Determine the number of genes annotated in a GFF3 file
- Determine all feature types annotated in a GFF3 file
- Basic FASTQ sequence statistics(number,percentage,...)

## 命令

- echo {A,C,G,T}{A,C,G,T}{A,C,G,T}
- grep -c '\tgene\t' yourannots.gff3
- grep -v '^#' GFF3 | cut -s -f 3 | sort | uniq
- cat myfile.fq | awk '((NR-2)%4==0) {read=\$1;total++;count[read]++}END{for(read in count){if(!max){max=count[read];maxRead=read};if(count[read]==1){unique+=}};print total,unique,unique\*100/total,maxRead,count[maxRead],count[maxRead]\*100/total}'

# Useful bash one-liners for bioinformatics

## 具体问题

- Print all possible 3mer DNA sequence combinations
- Determine the number of genes annotated in a GFF3 file
- Determine all feature types annotated in a GFF3 file
- Basic FASTQ sequence statistics(number,percentage,...)

## 命令

- echo {A,C,G,T}{A,C,G,T}{A,C,G,T}
- grep -c '\tgene\t' yourannots.gff3
- grep -v '^#' GFF3 | cut -s -f 3 | sort | uniq
- cat myfile.fq | awk '((NR-2)%4==0) {read=\$1;total++;count[read]++}END{for(read in count){if(!max||count[read]>max){max=count[read];maxRead=read};if(count[read]==1){unique+=}};print total,unique,unique\*100/total,maxRead,count[maxRead],count[maxRead]\*100/total}'

# Useful bash one-liners for bioinformatics

## 具体问题

- Print all possible 3mer DNA sequence combinations
- Determine the number of genes annotated in a GFF3 file
- Determine all feature types annotated in a GFF3 file
- Basic FASTQ sequence statistics(number,percentage,...)

## 命令

- echo {A,C,G,T}{A,C,G,T}{A,C,G,T}
- grep -c '\tgene\t' yourannots.gff3
- grep -v '^#' GFF3 | cut -s -f 3 | sort | uniq
- cat myfile.fq | awk '((NR-2)%4==0){read=\$1;total++;count[read]++;}END{for(read in count){if(!max||count[read]>max){max=count[read];maxRead=read};if(count[read]==1){unique+=}};print total,unique,unique\*100/total,maxRead,count[maxRead],count[maxRead]\*100/total}'

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因？
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因？
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -r`

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因?
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因?
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -r`

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因?
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因?
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -r`

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因?
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因?
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -nr`

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因?
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因?
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -r`

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因?
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因?
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -r`

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因?
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因?
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -r`

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因?
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因 ?
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -nr`

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因?
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因 ?
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -nr`

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因?
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因 ?
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -nr`

# Useful bash one-liners for bioinformatics

## 具体问题

- 基因组中一共有多少基因?
- 获取基因组中所有基因的 ID
- 计算每条染色体上的基因数目
- 每条染色体的正负链上各有多少基因 ?
- 把染色体的正负链按照基因的数目进行排序

## 命令

- `wc -l gene.bed`
- `cut -f4 gene.bed > gene_id.txt`
- `cut -f1 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c`
- `cut -f1,6 gene.bed | sort | uniq -c | sort -nr`

- Introduction to Linux for bioinformatics
- A Bioinformatician's UNIX Toolbox
- Some unix/perl oneliners for Bioinformatics
- Scott's list of linux one-liners
- Bioinformatics one-liners
- Article series “Bash One-Liners Explained”
- Article series “Awk One-Liners Explained”
- Article series “Sed One-Liners Explained”
- Article series “Perl One-Liners Explained”
- Article series “CommandLineFu One-Liners Explained”



1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题



1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题



## 知识点

- 元字符（字符，字符集，量词，边界），正则表达式（解析，构建）
- find 和 grep：常用选项，常用测试
- 文本处理命令：cut, wc, sort, uniq, ...
- sed：工作原理，定位方式，编辑命令
- AWK：脚本结构，工作原理，常见变量，记录，字段

## 技能

- 解析并编写正则表达式
- find 和 grep 的基本用法
- 文本处理命令的日常使用
- 使用 sed 和 AWK 处理文本
- 正则表达式在 grep、sed 和 AWK 中的应用

1 引言

2 正则表达式和元字符

- 元字符
- 正则表达式

3 find 和 grep

- find
- grep

4 文本处理命令

5 sed 和 AWK

- sed
- AWK

6 生物信息学中的应用

7 回顾与总结

- 总结
- 思考题



- ① 列举五个常见的元字符并解释其含义。
- ② 根据要求编写正则表达式。
- ③ 根据要求使用 find 查找文件。
- ④ 根据要求使用 grep 查找字符串。
- ⑤ 根据要求组合使用文本处理命令。
- ⑥ 根据要求编写 sed 脚本编辑文件。
- ⑦ 根据要求编写 AWK 脚本输出特定字段。



# 下节预告

在 Windows 系统下，你是如何安装、维护软件的？  
回顾总结 XX 软件管家的主要作用，你经常进行的操作。



# Powered by



T<sub>E</sub>X L<sup>A</sup>T<sub>E</sub>X X<sub>E</sub>T<sub>E</sub>X Beamer