Linux文本处理

```
作者：牟建波（1353429820@qq.com）
时间：2025-03-05
描述：日常学习笔记
```

1.文本编辑器VIM

VIM三种模式：命令模式、插入模式、底行模式

- 命令模式：进入就是，其他模式按 `ESC` 即可跳回
- 插入模式：按 `i`
- 末行模式： `shift+:`

命令模式常用命令：

- `dd`：删除
- `yy`：复制
- `p`：粘贴
- `u`：撤销操作
- `/要查找的内容`：查找内容

底行模式常用命令：

- `w`：写入
- `q`：退出
- `wq`：保存退出
- `q!`：不保存退出
- `s/要查找的内容/替换的内容/修饰符`：查找并替换
  - 修饰符：i(忽略大小写)、g(全局替换)、gc(替换前询问)

2.文本常见处理工具

2.1 文本内容查看命令

```
# 查看文本文件内容：cat、tac、nl、rev

# cat [选项] 文件名
常见选项：
-A：显示所有控制符，$标识行结束符
-n：显示行号
使用：cat test.txt                                  # 查看test.txt的内容

# tac [选项] 文件名
说明：tac逆向显示文本内容，就是cat的反向打印
使用：tac test.txt                                  # 逆向查看test.txt内容

# nl 文件名
说明：nl显示行号查看，相当于cat -b
使用：nl test.txt                                   # 查看test.txt的内容并显示行号

# rev 文件名
说明：rev将每一行内容逆向显示
使用：rev test.txt                                  # 查看test.txt的内容，每一行逆向显示

# 查看非文本文件内容：hexdump
说明：hexdump查看二进制文件
使用：hexdump test.txt                              # 查看test.txt的二进制形式
```

```
[root@Rocky jimbo]# hexdump test.txt
0000000 6173 6164 6473 7361 630a 7664 7364 7a0a
0000010 7763 670a 670a 670a 7a0a 7763 640a 730a
0000020 6461 7361 7a64 2077 2020 7361 6164 6473
0000030 2061 6120 6473 7361 6164 2073 6120 6473
0000040 7361 2064 0a0a 7361 6164 6473 0a0a 610a
0000050 6473 2064 7361 6164 6473 7361 2064 7361
0000060 6164 6473 0a20
0000066
```

2.2 分页查看文件内容

```
# 分页查看文件内容：more、less
说明：more只能向下翻页，翻完自动退出。less可上下翻页，翻完不自动退出
使用：more test.txt、less test.txt     # 分页查看文件内容
```

2.3 显示文本前面或后面的行内容

```
# 显示文本前面或后面的行内容：head、tail
说明：head默认显示文件前10行，tail默认显示文件后10行
选项：
    -n #：指定获取#行
    -c #：指定获取#字节
使用：
    head test.txt            # 显示test.txt前10行
    tail test.txt            # 显示test.txt后10行
    head -n 20 test.txt      # 显示test.txt前20行
    tail -n 20 test.txt      # 显示test.txt后20行
```

2.4 按列抽取文本cut（被awk替代）

```
# cut一般不用，被awk替代了

# cut [选项] 文件名
常用选项：
-d：指定字段分隔符（默认是制表符\t）
-f：指定要提取的列
-c：按字符位置提取

# 示例文档test.txt
Name,Age,City
Alice,30,New York
Bob,25,London

# 提取第2列
cut -d ',' -f2 test.txt     # 输出：Age
```

```
                                    30
                                    25

# 提取第1和第3列
cut -d ',' -f1,3 test.txt     # 输出：Name,City
                              #       Alice,New
                              #       Bob,London
```

2.5 合并多个文件paste

```
# paste [选项] 文件名
常用选项：
-d分隔符：指定分隔符（默认是制表符\t）
-s：所有行合并到一行显示

# 示例文档test.txt、test2.txt
            a       1
            b       2
            c       3
            d
            e

# 合并test.txt和test2.txt
paste test.txt test2.txt     # 显示
                               a    1
                               b    2
                               c    3
                               d
                               e

# 指定:分隔符合并
paste -d: test.txt test2.txt         # 显示
                                       a:1
                                       b:2
                                       c:3
                                       d:
                                       e:
```

2.6 分析文本的工具

```
# 收集文本统计数据：wc [选项] 文件名
常用选项：默认（行号、单词数、字节数）
-l：只计数行数
-w：只计数单词总数
-c：只计数字节总数
-m：只计数数字符总数
-L：显示文本中最长行

wc test.txt       # 显示test.txt的行号、单词数、字节数

# 文件排序：sort [选项] 文件名
常用选项：
-r：逆向排序
-h：以人类可读排序
-u：去重
-t 字符：用字符作为字段界定符

df -h | tr -s '' % | cut -d% -f5 | sort -nr | head -n1    # 查看分区利用率最高值

# 文件去重：uniq [选项] 文件名
常见选项：
-c：显示每行重复出现的次数
-d：仅显示重复过的行
-u：仅显示不曾重复过的行

sort test.txt | uniq # 先排序后去重

# 文件比较：diff、patch、vimdiff、cmp
说明：
    diff：用来比较两个文件之间的区别
    patch：复制在其它文件中进行的改变
    vimdiff：相当于vim -d
    cmp：查看二进制文件的不同
```

3.正则表达式

正则表达式分为两种：基本正则表达式、扩展正则表达式

- 基本正则表达式：`BRE`（Basic Regular Expressions）
- 扩展正则表达式：`ERE`（Extended Regular Expressions）
- **BRE和ERE的主要区别：ERE简化了写法，把\去掉了**

3.1 基本正则表达式

一、基本元字符

元字符	作用	示例
`.`	匹配任意**单个字符**（除换行符）	`a.c` → `abc`，`a3c`
`^`	匹配行首	`^start` → 以 "start" 开头的行
`$`	匹配行尾	`end$` → 以 "end" 结尾的行
`*`	匹配前一个字符**0次或多次**	`ab*c` → `ac`，`abbc`
`+`	匹配前一个字符**1次或多次**	`ab+c` → `abc`，`abbc`（BRE 中需转义为 `\+`）
`?`	匹配前一个字符**0次或1次**	`ab?c` → `ac`，`abc`（BRE 中需转义为 `\?`）
`[]`	匹配括号内的**任意一个字符**	`[aeiou]` → 匹配任意元音字母
`[^]`	匹配**不在**括号内的任意字符	`[^0-9]` → 匹配非数字字符
`\`	转义字符（用于匹配特殊字符本身）	`\.` → 匹配点号 `.`

二、字符集合与范围

模式	说明	示例
`[a-z]`	匹配任意小写字母	`gr[ae]y` → `gray` 或 `grey`
`[A-Z]`	匹配任意大写字母	`[A-Z]` → 匹配任意大写字母
`[0-9]`	匹配任意数字	`[0-9]` → 匹配数字字符
`[a-zA-Z0-9]`	匹配任意字母或数字	`[a-zA-Z0-9_]` → 匹配单词字符

三、预定义字符类

简写	等效表达式	说明
`\d`	`[0-9]`	数字字符
`\D`	`[^0-9]`	非数字字符
`\w`	`[a-zA-Z0-9_]`	单词字符（字母、数字、下划线）
`\W`	`[^a-zA-Z0-9_]`	非单词字符
`\s`	`[ \t\n\r\f\v]`	空白字符（空格、制表符等）
`\S`	`[^ \t\n\r\f\v]`	非空白字符

四、量词（需转义）

量词	说明	示例
`\{n\}`	匹配前一个字符**恰好 n 次**	`a\{3\}` → `aaa`
`\{n,\}`	匹配前一个字符**至少 n 次**	`a\{2,\}` → `aa`，`aaaaa`
`\{n,m\}`	匹配前一个字符**n 到 m 次**	`a\{2,4\}` → `aa`，`aaaa`

五、分组与捕获

模式	说明	示例
`\( \)`	分组（捕获匹配内容）	`\(ab\)\+` → `ab`，`ababab`
`\1`，`\2`	引用分组（后向引用）	`\(a\).\1` → `aXa`（X 为任意字符）


```
# 匹配邮箱地址
^[a-zA-Z0-9._%+-]\+@[a-zA-Z0-9.-]\+\.[a-zA-Z]\{2,\}$

# 匹配URL
^https\?://\([a-zA-Z0-9.-]\+\.[a-zA-Z]\{2,\}\)\(/.*\)\?$

# 匹配日期YYYY-MM-DD
^\d\{4\}-\d\{2\}-\d\{2\}$

# 匹配IP地址
^\d\{1,3\}\.\d\{1,3\}\.\d\{1.3\}\.\d\{1,3\}$
```

3.2 扩展正则表达式(推荐使用)

1. 基本元字符

元字符是正则表达式中的特殊字符，用于匹配特定模式，而不仅仅是字面字符。

元字符	描述	示例
`.`	匹配任意单个字符（不包括换行符）	`c.t` 可匹配 `cat`、`cut`、`c9t`
`^`	匹配行的开头	`^Hello` 只匹配 `Hello` 在行首的情况
`$`	匹配行的结尾	`end$` 只匹配 `end` 在行尾的情况
`|`	逻辑 "或"	`dog|cat` 可匹配 `dog` 或 `cat`

2. 字符集合与范围

字符集合（`[]`）用于匹配其中的任意单个字符，而字符范围用于简化集合的书写。

表示法	描述	示例
`[abc]`	匹配 `a`、`b` 或 `c`	`c[ao]t` 可匹配 `cat` 或 `cot`
`[^abc]`	**不匹配** `a`、`b` 或 `c`	`[^0-9]` 匹配非数字字符
`[a-z]`	匹配 `a` 到 `z` 之间的任何字母	`[a-zA-Z]` 匹配所有字母
`[0-9]`	匹配 `0` 到 `9` 之间的任何数字	`[0-9]+` 匹配至少一个数字
`[a-zA-Z0-9_]`	匹配所有字母、数字和下划线（类似于 `\w`）	`_test` 可匹配

3. 预定义字符类

扩展正则表达式提供了一些常见的字符类，用于简化匹配。

字符类	等价表达式	描述
`\d`	`[0-9]`	匹配任意数字
`\D`	`[^0-9]`	匹配非数字字符
`\w`	`[a-zA-Z0-9_]`	匹配字母、数字、下划线
`\W`	`[^a-zA-Z0-9_]`	匹配非字母、数字、下划线
`\s`	`[ \t\r\n\f]`	匹配空白字符（空格、制表符、换行等）
`\S`	`[^ \t\r\n\f]`	匹配非空白字符

4. 量词

ERE 允许直接使用 `{}`、`+` 和 `?`，而不需要像 BRE 那样使用 `\{ \}`。

量词	描述	示例
`*`	匹配前一个字符 **0 次或多次**	`do*g` 可匹配 `dg`、`dog`、`dooog`
`+`	匹配前一个字符 **1 次或多次**	`do+g` 可匹配 `dog`、`dooog` 但不匹配 `dg`
`?`	匹配前一个字符 **0 次或 1 次**	`colou?r` 可匹配 `color` 或 `colour`
`{n}`	匹配前一个字符 **恰好 n 次**	`o{3}` 只匹配 `ooo`
`{n,}`	匹配前一个字符 **至少 n 次**	`o{2,}` 匹配 `oo`、`ooo`、`oooo`
`{n,m}`	匹配前一个字符 **至少 n 次，至多 m 次**	`o{2,4}` 匹配 `oo`、`ooo`、`oooo`

5. 分组与捕获

扩展正则表达式支持 `()` 进行分组，简化匹配逻辑，并支持 `\|` 进行多选匹配。

符号	描述	示例
`()`	进行分组	`(hello|hi), world` 匹配 `hello, world` 或 `hi, world`
`(?:...)`	非捕获组，不存储匹配内容	`(?:Mr|Mrs)\. Smith` 只匹配 `Mr. Smith` 或 `Mrs. Smith`

6. 转义字符

有些元字符如果想匹配它们的字面值，就需要用 `\` 进行转义。

元字符	转义后	匹配
`.`	`\.`	仅匹配 `.` 这个字符，而不是任意字符
`*`	`\*`	仅匹配 `*` 符号，而不是 0 次或多次
`+`	`\+`	仅匹配 `+` 号
`?`	`\?`	仅匹配 `?` 号
`{}`	`\{ \}`	仅匹配 `{}` 符号，而不表示量词
`()`	`\(\)`	仅匹配 `()` 符号，而不用于分组

```
# 匹配邮箱地址
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$

# 匹配URL
^https?://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}(/.*)?$

# 匹配日期YYYY-MM-DD
^(19|20)\d{2}-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])$

# 匹配IP地址
^(25[0-5]|2[0-4][0-9]|1?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|1?[0-9][0-9]?)\.
(25[0-5]|2[0-4][0-9]|1?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|1?[0-9][0-9]?)$
```

⭐4.文件处理三剑客：grep、sed、awk

在 Linux/Unix 中，`grep`、`sed` 和 `awk` 是最常用的文本处理工具。它们各自擅长不同的任务：

- `grep`：主要用于 **搜索**（查找匹配的文本行）。
- `sed`：主要用于 **编辑**（流式文本替换、删除、插入等）。
- `awk`：主要用于 **分析**（格式化文本、处理列数据、统计计算）。

⭐4.1 文本搜索：grep

```
# 格式：grep [选项] [模式] 文件名
常见选项：
    -i：忽略字符大小写
    -n：显示匹配的行号
    -v：显示不含字符的行
    -c：统计匹配的行数
    -r：递归目录，但不处理软链接
    -R：递归目录，处理软链接
    -E：使用扩展正则表达式

# grep基本用法
grep "hello" file.txt                  # 查找包含 "hello" 的行
grep -i "hello" file.txt               # 忽略大小写
grep -n "hello" file.txt               # 显示匹配行及行号
grep -v "hello" file.txt               # 反向匹配（不包含 "hello" 的行）
grep -c "hello" file.txt               # 统计匹配 "hello" 的行数
grep -r "pattern" /path/               # 递归查找目录

# grep正则表达式
grep "^start" file.txt                 # 匹配以 "start" 开头的行
grep "end$" file.txt                   # 匹配以 "end" 结尾的行
grep -E "foo|bar" file.txt             # 匹配 "foo" 或 "bar"（扩展正则）
grep -E "[0-9]{3}-[0-9]{4}" file.txt   # 匹配 3 位数字-4 位数字格式
```

⭐4.2 文本编辑：sed（难）

sed工作原理：读取一行，处理一行，输出一行，直到最后一行

- sed每读取一行就将其存储到临时缓冲区，再处理缓冲区里的内容，最后输出到屏幕
- sed每次处理一行，不会出现读取大文件时的卡顿问题，因而sed性能很高

- sed每次执行完会自动打印本行内容

```
# 格式：sed [选项] '脚本' 文件名
常见选项：
    -n：取消sed自动打印
    -e：多点编辑
    -i：原处编辑，一般使用-i.bak，先备份后编辑
脚本：
    查找替换：'s/pattern/string/修饰符'    g行内全局替换
    删除命令：'/pattern/d'                 删除包含pattern的行
    地址格式：'#1,#2d'                     删除#1到#2行

# sed基本用法
sed 's/old/new/' file.txt                          # 只替换第一处 old 为 new
sed 's/old/new/g' file.txt                         # 替换整行中所有 old 为 new
sed -i 's/old/new/g' file.txt                      # 直接修改文件（危险！建议先备份）
sed '/pattern/d' file.txt                          # 删除包含 "pattern" 的行
sed '2,5d' file.txt                                # 删除第 2 到 5 行

# sed插入/追加
sed '1i\Hello' file.txt                            # 在第一行前插入 "Hello"
sed '$a\The End' file.txt                          # 在最后一行后追加 "The End"

# sed处理正则
sed -E 's/[0-9]{3}-[0-9]{4}/###-####/g' file.txt   # 屏蔽电话号码
```

⭐4.3 文本分析：awk（难）

```
# awk基本语法
awk '{print $1, $3}' file.txt                # 只打印第 1 和第 3 列
awk -F ',' '{print $2}' file.csv             # 指定逗号为分隔符，打印第 2 列

# awk过滤
awk '$3 > 100 {print $1, $3}' file.txt       # 过滤第 3 列大于 100 的行
awk 'NR==5, NR==10' file.txt                 # 显示第 5 到 10 行

# awk统计
awk '{sum += $3} END {print sum}' file.txt   # 计算第 3 列的总和
```