文本编辑器测试命令文档

本文档提供了完整的测试命令序列，用于测试文本编辑器的所有功能。

测试前准备

1. 确保程序已编译并可以运行
2. 准备测试文件（可选，程序会自动创建）
3. 清理之前的 `.editor_workspace` 文件（如果需要测试首次启动）

一、基础功能测试

1.1 工作区命令测试

测试场景1：文件加载和初始化

```
# 1. 启动编辑器（首次启动）
# 程序启动后，应该显示提示信息或命令提示符

# 2. **初始化新文件**
init test1.txt
# 预期：创建新文件 test1.txt，并设置为活动文件

# 3. 查看编辑器列表
editor-list
# 预期输出：
# * test1.**txt**

# 4. 初始化带日志的文件
init test2.txt with-log
# 预期：创建新文件 test2.txt，首行自动添加 "# log"

# 5. 查看编辑器列表
editor-list
# 预期输出：
# * test2.txt
#   test1.txt

# 6. 加载已存在的文件（如果存在）
load test1.txt
# 预期：切换到 test1.txt 作为活动文件

# 7. 再次查看编辑器列表
editor-list
# 预期输出：
#   test2.txt
# * test1.txt
```

测试场景2：文件编辑和保存

```
# 1. 确保 test1.txt 是活动文件
edit test1.txt

# 2. 追加文本
append "Hello World"
# 预期：文件末尾添加一行 "Hello World"

# 3. 追加更多文本
append "This is line 2"
append "This is line 3"

# 4. 显示文件内容
show
# 预期输出：
# 1: Hello World
# 2: This is line 2
# 3: This is line 3

# 5. 查看编辑器列表（应该显示已修改）
editor-list
# 预期输出：
# * test1.txt [已修改]
#   test2.txt

# 6. 保存文件
save
# 预期：保存 test1.txt，修改标记消失

# 7. 再次查看编辑器列表
editor-list
# 预期输出：
# * test1.txt
#   test2.txt

# 8. 保存指定文件
save test2.txt
# 预期：保存 test2.txt（如果已修改）

# 9. 保存所有文件
save all
# 预期：保存所有已修改的文件
```

测试场景3：插入、删除和替换操作

```
# 1. 切换到 test1.txt
edit test1.txt

# 2. 显示当前内容
```

```
show
# 预期：显示之前添加的3行内容

# 3. 在第1行第6个字符位置插入文本
insert 1:6 " beautiful"
# 预期：第1行变为 "Hello beautiful World"

# 4. 显示修改后的内容
show
# 预期输出：
# 1: Hello beautiful World
# 2: This is line 2
# 3: This is line 3

# 5. 删除第2行的前5个字符
delete 2:1 5
# 预期：第2行变为 "is line 2"

# 6. 替换第3行的内容
replace 3:1 13 "Final line"
# 预期：第3行变为 "Final line"

# 7. 显示最终内容
show
# 预期输出：
# 1: Hello beautiful World
# 2: is line 2
# 3: Final line

# 8. 显示指定行范围
show 1:2
# 预期输出：
# 1: Hello beautiful World
# 2: is line 2
```

测试场景4：撤销和重做

```
# 1. 确保 test1.txt 是活动文件
edit test1.txt

# 2. 追加一行
append "Line 4"

# 3. 再追加一行
append "Line 5"

# 4. 显示内容
show
# 预期：显示5行内容

# 5. 撤销一次操作
```

```
undo
# 预期：删除 "Line 5"

# 6. 显示内容
show
# 预期：显示4行内容

# 7. 再撤销一次
undo
# 预期：删除 "Line 4"

# 8. 显示内容
show
# 预期：显示3行内容

# 9. 重做一次操作
redo
# 预期：恢复 "Line 4"

# 10. 显示内容
show
# 预期：显示4行内容

# 11. 再重做一次
redo
# 预期：恢复 "Line 5"

# 12. 显示内容
show
# 预期：显示5行内容
```

测试场景5：多文件编辑

```
# 1. 查看当前编辑器列表
editor-list
# 预期：显示所有打开的文件

# 2. 切换到 test2.txt
edit test2.txt

# 3. 在 test2.txt 中追加内容
append "File 2 content"

# 4. 切换到 test1.txt
edit test1.txt

# 5. 在 test1.txt 中追加内容
append "File 1 additional content"

# 6. 查看编辑器列表
editor-list
```

```
# 预期输出：
#   test2.txt [已修改]
# * test1.txt [已修改]

# 7. 保存当前文件（test1.txt）
save

# 8. 切换到 test2.txt 并保存
edit test2.txt
save

# 9. 查看编辑器列表（应该没有修改标记）
editor-list
# 预期输出：
#   test2.txt
# * test1.txt
```

测试场景6：文件关闭（含保存提示）

```
# 1. 查看编辑器列表
editor-list
# 预期：显示所有打开的文件（例如：* test1.txt / test2.txt）

# 2. 在 test1.txt 中制造未保存修改
edit test1.txt
append "unsaved change"

# 3. 关闭已修改的文件（带文件名）
close test1.txt
# 预期：终端提示
# 文件已修改，是否保存？(y/n)：
# - 若输入 y：先保存 test1.txt 再关闭，编辑器列表中不再有 test1.txt
# - 若输入 n：不保存直接关闭，文件内容保持上次保存状态
# - 若输入其它字符：输出 "操作已取消"，test1.txt 仍保持打开状态

# 4. 再次查看编辑器列表
editor-list
# 预期：根据上一步选择，test1.txt 可能已经被关闭；如果关闭了，其他文件中最近使用的一个会成为活动文件

# 5. 关闭当前活动文件（不指定文件名）
close
# 预期：关闭当前活动文件；如有未保存修改，同样弹出保存提示并按 y/n 处理

# 6. 多次执行 close，直到所有文件关闭
close
close

# 7. 查看编辑器列表
editor-list
```

```
# 预期输出：
# 没有打开的文件
```

测试场景7：目录树显示

```
# 1. 显示当前工作目录的目录树
dir-tree
# 预期：以树形结构显示当前目录的文件和文件夹

# 2. 显示指定目录的目录树
dir-tree .
# 预期：显示当前目录的目录树

# 3. 显示父目录的目录树（如果存在）
dir-tree ..
# 预期：显示父目录的目录树
```

二、日志功能测试

2.1 自动日志启用测试

```
# 1. 创建一个首行为 "# log" 的文件（手动创建或通过程序）
# 文件内容：
# # log
# This is a test file with logging enabled

# 2. 加载该文件
load log_test.txt
# 预期：自动启用日志记录

# 3. 执行一些编辑操作
append "Line 2"
insert 1:1 "Start: "
replace 1:1 6 "Begin:"

# 4. 查看日志
log-show
# 预期：显示所有命令执行记录，包括时间戳

# 5. 保存文件
save

# 6. 再次查看日志
log-show
# 预期：日志中包含 save 命令的记录
```

2.2 手动日志控制测试

```
# 1. 加载一个普通文件
load normal.txt

# 2. 启用日志
log-on
# 预期：为 normal.txt 启用日志记录

# 3. 执行一些操作
append "Test line 1"
append "Test line 2"
show

# 4. 查看日志
log-show
# 预期：显示刚才执行的命令记录

# 5. 关闭日志
log-off
# 预期：停止记录日志

# 6. 执行更多操作
append "Test line 3"
show

# 7. 查看日志（应该没有新的记录）
log-show
# 预期：只显示 log-off 之前的记录

# 8. 重新启用日志
log-on

# 9. 执行操作
append "Test line 4"

# 10. 查看日志（应该包含新记录）
log-show
# 预期：包含 log-on 和 append 的记录
```

2.3 多文件日志测试

```
# 1. 打开多个文件
load file1.txt
load file2.txt
load file3.txt

# 2. 为不同文件启用日志
edit file1.txt
log-on file1.txt
```

```
edit file2.txt
log-on file2.txt

# file3.txt 不启用日志

# 3. 在 file1.txt 中操作
edit file1.txt
append "File 1 content"

# 4. 在 file2.txt 中操作
edit file2.txt
append "File 2 content"

# 5. 在 file3.txt 中操作
edit file3.txt
append "File 3 content"

# 6. 查看各文件的日志
log-show file1.txt
# 预期：显示 file1.txt 的日志

log-show file2.txt
# 预期：显示 file2.txt 的日志

log-show file3.txt
# 预期：显示 "没有日志记录" 或空日志
```

三、边界条件和错误处理测试

3.1 空文件操作测试

```
# 1. 初始化空文件
init empty.txt

# 2. 显示内容
show
# 预期：显示空内容或提示文件为空

# 3. 在空文件中插入（必须在1:1位置）
insert 1:1 "First line"
# 预期：成功插入

# 4. 显示内容
show
# 预期输出：
# 1: First line

# 5. 尝试在无效位置插入（应该失败）
insert 2:1 "Second line"
```

```
# 预期：错误提示位置无效

# 6. 在正确位置插入
insert 1:12 "\nSecond line"
# 预期：成功插入，创建新行
```

3.2 位置越界测试

```
# 1. 加载或创建测试文件
load test.txt
# 假设文件有3行，每行10个字符

# 2. 尝试在无效行插入
insert 10:1 "text"
# 预期：错误提示行号超出范围

# 3. 尝试在无效列插入
insert 1:100 "text"
# 预期：错误提示列号超出范围

# 4. 尝试删除超出范围的内容
delete 1:100 10
# 预期：错误提示位置无效

# 5. 尝试替换超出范围的内容
replace 10:1 5 "text"
# 预期：错误提示位置无效
```

3.3 无活动文件测试

```
# 1. 关闭所有文件（如果已打开）
# close all files...

# 2. 尝试在没有活动文件时执行编辑命令
append "test"
# 预期：错误提示 "没有活动文件"

insert 1:1 "test"
# 预期：错误提示 "没有活动文件"

delete 1:1 5
# 预期：错误提示 "没有活动文件"

replace 1:1 5 "test"
# 预期：错误提示 "没有活动文件"

show
# 预期：错误提示 "没有活动文件"
```

```
undo
# 预期：错误提示 "没有活动文件"

redo
# 预期：错误提示 "没有活动文件"

save
# 预期：错误提示 "没有指定要保存的文件"
```

3.4 文件不存在测试

```
# 1. 尝试加载不存在的文件
load nonexistent.txt
# 预期：成功加载（创建空文件）或提示文件不存在

# 2. 尝试编辑不存在的文件
edit nonexistent.txt
# 预期：错误提示 "文件未打开"

# 3. 尝试关闭不存在的文件
close nonexistent.txt
# 预期：错误提示 "文件未打开"

# 4. 尝试为不存在的文件启用日志
log-on nonexistent.txt
# 预期：错误提示 "文件未打开"
```

3.5 撤销/重做边界测试

```
# 1. 加载文件
load test.txt

# 2. 在没有操作时尝试撤销
undo
# 预期：错误提示 "没有可撤销的操作"

# 3. 执行一些操作
append "Line 1"
append "Line 2"
append "Line 3"

# 4. 撤销多次
undo
undo
undo
# 预期：前两次成功，第三次可能失败或提示

# 5. 在没有撤销操作时尝试重做
redo
```

```
# 预期：错误提示 "没有可重做的操作"

# 6. 撤销一次后再重做
undo
redo
# 预期：成功重做
```

四、工作区状态持久化测试

4.1 状态保存和恢复测试

```
# 1. 启动编辑器（首次启动）
# 程序启动

# 2. 打开多个文件
init file1.txt
init file2.txt
load file3.txt

# 3. 在不同文件中编辑
edit file1.txt
append "File 1 content"
save

edit file2.txt
append "File 2 content"
# 不保存，保持修改状态

edit file3.txt
append "File 3 content"
save

# 4. 启用日志
log-on file1.txt
log-on file3.txt

# 5. 查看编辑器列表
editor-list
# 预期：显示所有文件及其状态

# 6. 退出程序
exit
# 预期：保存工作区状态到 .editor_workspace

# 7. 重新启动编辑器
# 程序启动

# 8. 查看编辑器列表
editor-list
# 预期：自动恢复之前打开的文件
```

```
# * file1.txt (或根据保存时的活动文件)
#   file2.txt [已修改]
#   file3.txt

# 9. 验证文件内容
edit file1.txt
show
# 预期：显示之前保存的内容

edit file2.txt
show
# 预期：显示文件系统中的内容（未保存的修改可能丢失）

edit file3.txt
show
# 预期：显示之前保存的内容

# 10. 验证日志状态
log-show file1.txt
# 预期：显示日志记录（如果已启用）

log-show file3.txt
# 预期：显示日志记录（如果已启用）
```

4.2 修改状态恢复测试

```
# 1. 启动编辑器
# 程序启动

# 2. 打开文件并修改
load test.txt
append "New line"
# 不保存

# 3. 查看编辑器列表
editor-list
# 预期：test.txt [已修改]

# 4. 退出程序
exit

# 5. 重新启动编辑器
# 程序启动

# 6. 查看编辑器列表
editor-list
# 预期：test.txt 应该被恢复，但修改状态可能根据实际实现而定
# 注意：未保存的修改在恢复时可能丢失（从文件系统加载）
```

五、综合场景测试

5.1 完整编辑流程测试

```
# 1. 启动编辑器
# 程序启动

# 2. 初始化新文件
init article.txt with-log

# 3. 编写文章内容
append "标题：我的第一篇文章"
append ""
append "这是第一段内容。"
append "这是第二段内容。"
append ""
append "结尾段落。"

# 4. 显示全文
show
# 预期：显示所有6行内容

# 5. 修改标题
replace 1:4 2 "标题"
# 预期：第1行变为 "标题：我的第一篇文章"

# 6. 在第一段后插入新段落
insert 4:1 "这是插入的新段落。\n"

# 7. 显示修改后的内容
show
# 预期：显示修改后的内容

# 8. 撤销插入操作
undo

# 9. 显示内容
show
# 预期：恢复到插入前的状态

# 10. 保存文件
save

# 11. 查看日志
log-show
# 预期：显示所有操作记录

# 12. 关闭文件
close

# 13. 重新加载文件
load article.txt
```

```
# 14. 显示内容
show
# 预期：显示保存的内容
```

5.2 多文件协作编辑测试

```
# 1. 启动编辑器
# 程序启动

# 2. 创建多个文件
init main.txt
init config.txt
init readme.txt

# 3. 在不同文件中编辑
edit main.txt
append "Main file content"
save

edit config.txt
append "Config file content"
save

edit readme.txt
append "Readme file content"
save

# 4. 切换文件并继续编辑
edit main.txt
append "Additional main content"

edit config.txt
append "Additional config content"

# 5. 查看所有文件状态
editor-list
# 预期：显示所有文件，标记已修改的文件

# 6. 保存所有文件
save all

# 7. 关闭所有文件
close main.txt
close config.txt
close readme.txt

# 8. 验证所有文件都已关闭
editor-list
# 预期：没有打开的文件
```

5.3 日志完整流程测试

```
# 1. 启动编辑器
# 程序启动

# 2. 创建带日志的文件
init log_demo.txt with-log

# 3. 执行一系列操作
append "Step 1: Initial content"
insert 1:1 "Step 0: "
replace 1:1 5 "Step"
delete 1:6 1
append "Step 2: More content"
show
save

# 4. 查看完整日志
log-show
# 预期：显示所有操作的时间戳和命令

# 5. 关闭日志
log-off

# 6. 执行更多操作
append "Step 3: No logging"
show

# 7. 查看日志（应该没有新记录）
log-show
# 预期：只显示 log-off 之前的记录

# 8. 重新启用日志
log-on

# 9. 执行操作
append "Step 4: Logging resumed"
save

# 10. 查看日志
log-show
# 预期：包含 log-on 之后的所有记录
```

六、特殊字符和边界情况测试

6.1 特殊字符处理测试

```
# 1. 创建测试文件
init special.txt
```

```
# 2. 测试包含空格的文本
append "Text with spaces"
# 预期：成功追加

# 3. 测试包含引号的文本（如果支持转义）
append "Text with \"quotes\""
# 预期：根据实现处理

# 4. 测试空字符串
append ""
# 预期：追加空行

# 5. 测试换行符
insert 1:1 "Line 1\nLine 2"
# 预期：插入多行内容

# 6. 显示内容
show
# 预期：正确显示多行内容
```

6.2 大文件操作测试

```
# 1. 创建文件
init large.txt

# 2. 添加多行内容
append "Line 1"
append "Line 2"
# ... 添加更多行（可以编写脚本批量添加）

# 3. 显示指定范围
show 1:10
# 预期：显示前10行

# 4. 显示中间范围
show 50:60
# 预期：显示第50-60行

# 5. 显示末尾范围
show 90:100
# 预期：显示第90-100行
```

6.3 替换操作边界测试

```
# 1. 创建测试文件
init replace_test.txt
append "Hello world"
append "Test line"
```

```
# 2. 替换长度为0（相当于插入）
replace 1:6 0 " beautiful"
# 预期：在位置6插入 " beautiful"
# 结果：Hello beautiful world

# 3. 替换文本为空字符串（相当于删除）
replace 1:6 10 ""
# 预期：删除从位置6开始的10个字符

# 4. 替换跨行内容
replace 1:1 20 "New content"
# 预期：替换第一行和部分第二行
```

七、错误恢复测试

7.1 文件操作错误恢复

```
# 1. 尝试操作不存在的文件
load test.txt
# 如果文件不存在，应该创建空文件或提示

# 2. 尝试保存到只读位置（如果可能）
save /readonly/test.txt
# 预期：错误提示，但不崩溃

# 3. 尝试加载损坏的文件
load corrupted.txt
# 预期：错误处理，不崩溃
```

7.2 命令参数错误测试

```
# 1. 测试缺少参数的命令
load
# 预期：错误提示缺少文件路径

insert
# 预期：错误提示缺少参数

delete
# 预期：错误提示缺少参数

# 2. 测试参数格式错误
insert abc:def "text"
# 预期：错误提示参数格式错误

delete 1:abc 5
# 预期：错误提示参数格式错误
```

```
# 3. 测试无效的命令
invalid_command
# 预期：错误提示未知命令
```
