The Missing Semester

一、Shell

1. What

- 一种交互接口：文字接口
- 核心功能：执行程序，输入并获取某种半结构化的输出

2. How（以bash为例）

2.1 使用Shell

- shell 基于**空格**分割命令并进行解析，然后执行第一个单词代表的程序，并将后续的单词作为程序可以访问的参数
- 在 shell 中执行命令时，实际上是在执行一段 shell 可以解释执行的简短代码

2.2 在Shell中导航（以Linux为例）

- shell 中的路径是一组被分割的目录，使用 `/` 分割。路径 `/` 代表的是系统的根目录，所有的文件夹都包括在这个路径之下。如果某个路径以 `/` 开头，那么它是一个绝对路径，其他的都是相对路径

2.3 在程序间创建连接

- `< file` `> file` 将程序的输入输出流分别重定向到文件

```
lane@Lane:~$ echo hello > hello.txt
lane@Lane:~$ cat hello.txt
hello
lane@Lane:~$ cat < hello.txt
hello
lane@Lane:~$ cat < hello.txt > hello2.txt
lane@Lane:~$ cat hello2.txt
hello
```

- `>>` 向一个文件追加内容

```
lane@Lane:~$ echo lane >> hello.txt
lane@Lane:~$ cat hello.txt
hello
lane
```

- 管道符号 `|` 将一个程序的输出和另外一个程序的输入连接起来

```
lane@Lane:~$ ls | grep "txt"
file1.txt
file2.txt
hello.txt
hello2.txt
```

  - 上述命令将列出当前目录下的文件和文件夹，并将结果通过管道传递给后面的grep命令。grep命令用于在输入中搜索包含"txt"的行，即筛选出所有包含"txt"的文件名

2.4 Shell脚本

- Shebang（释伴）
  - 是一个位于脚本文件的第一行的特殊注释，用于指定该脚本应该由哪个解释器来执行
  - 它的格式通常是以井号（#）和叹号（!）开头，后面跟着解释器的路径。

以Python为例

```
#!/usr/bin/python
print("Hello, World!")
```

- 注释
  - `#`
- 变量
  - 变量赋值 `foo=bar` 中间不要空格
  - 引用变量 `$foo`

```
foo=bar
echo "$foo"
```

```
# 打印 bar(变量值会被替换)
echo '$foo'
# 打印 $foo(原义字符串)
```

- 输入输出
  - 输出：使用 `echo` 命令打印文本
  - 输入：使用 `read` 命令读取用户输入
  - 示例：

```
echo "What is your name?"
read name
echo "Hello, $name!"
```

- 条件判断
  - 字符串比较
    - `=`：判断两个字符串是否相等
    - `!=`：判断两个字符串是否不相等
  - 数值比较
    - `-eq`：判断两个数值是否相等
    - `-ne`：判断两个数值是否不相等
    - `-gt`：判断第一个数值是否大于第二个数值
    - `-lt`：判断第一个数值是否小于第二个数值
    - `-ge`：判断第一个数值是否大于等于第二个数值
    - `-le`：判断第一个数值是否小于等于第二个数值
  - 文件比较
    - `-e`：判断文件是否存在
    - `-f`：判断文件是否为普通文件
    - `-d`：判断文件是否为目录
    - `-s`：判断文件是否非空
    - `-r`：判断文件是否可读

        - `-w`：判断文件是否可写
        - `-x`：判断文件是否可执行
    - 逻辑运算符
        - `&&`：逻辑与，表示两个条件都为真时整个条件为真。
        - `||`：逻辑或，表示两个条件至少有一个为真时整个条件为真。
        - `!`：逻辑非，表示取反操作，将真变为假，假变为真。
- 条件语句：
    - `if` 语句
    - 示例：

```
if [ $age -gt 18 ]; then
    echo "You are an adult."
else
    echo "You are a minor."
fi
```

- 循环结构：
    - `for` 循环
    - `while` 循环
    - 示例：

```
for item in apple banana orange; do
    echo "I like $item."
done

counter=1
while [ $counter -le 10 ]; do
    echo "Count: $counter"
    counter=$((counter+1))
done
```

- 函数：
    - 定义函数：`function_name() { commands }`
    - 调用函数：`function_name`

- 示例：

```
greet() {
    echo "Hello!"
}

greet
```

- 特殊参数
  - `$0` - 脚本名
  - `$1` 到 `$9` - 脚本的参数。 `$1` 是第一个参数，依此类推。
  - `$@` - 所有参数
  - `$#` - 参数个数
  - `$?` - 前一个命令的返回值
  - `$$` - 当前脚本的进程识别码
  - `!!` - 完整的上一条命令，包括参数。
    - 常见应用：当你因为权限不足执行命令失败时，可以使用 `sudo !!` 再尝试一次。
  - `$_` - 上一条命令的最后一个参数。如果你正在使用的是交互式 shell，你可以通过按下 `Esc` 之后键入 . 来获取这个值。

创建并进入文件夹

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

- 退出码
  - 0表示正常执行，非0表示有错误发生
  - 退出码可以搭配 `&&` （与操作符）和 `||` （或操作符）使用，用来进行条件判断，决定是否执行其他程序
  - `||` 第一个表达式为假（0），则执行第二个表达式
  - `&&` 第一个表达式为真（!0），则执行第二个表达式

```
```

```bash
#程序 true 的退出码永远是0，false 的退出码永远是1

false || echo "Oops, fail"
# Oops, fail

true || echo "Will not be printed"
#

true && echo "Things went well"
# Things went well

false && echo "Will not be printed"
#

#同一行的多个命令可以用 ; 分隔
false ; echo "This will always run"
# This will always run
```

- 示例

示例

```bash
#!/bin/bash

echo "Starting program at $(date)" # date会被替换成日期和时间

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

for file in "$@"; do
    grep foobar "$file" > /dev/null 2> /dev/null
    # 如果模式没有找到，则grep退出状态为 1
    # 我们将标准输出流和标准错误流重定向到Null，因为我们并不关心这些信息
    if [[ $? -ne 0 ]]; then
        echo "File $file does not have any foobar, adding one"
        echo "# foobar" >> "$file"
    fi
done
```

2.5 快捷键

按键	作用
`Ctrl+c`	强行终止当前程序
`Ctrl+d`	键盘输入结束或退出终端
`Ctrl+s`	暂停当前程序，暂停后按下任意键恢复运行

`Ctrl+z`	将当前程序放到后台运行，恢复到前台为命令 `fg`
`Ctrl+r`	对历史记录输入子串来进行回溯搜索，反复按下可切换搜索结果
`Ctrl+a`	将光标移至输入行头，相当于 `Home` 键
`Ctrl+e`	将光标移至输入行末，相当于 `End` 键
`Ctrl+k`	删除从光标所在位置到行末
`Alt+Backspace`	向前删除一个单词
`Shift+PgUp`	将终端显示向上滚动
`Shift+PgDn`	将终端显示向下滚动

2.6 通配符

▾ 示例

```
shiyanlou:~/ $ touch a{1,2,3}.txt  //多个文件的快速创建
shiyanlou:~/ $ ls *.txt            //通配符的使用
a1.txt  a2.txt  a3.txt
```

字符	含义
`*`	匹配 0 或多个字符
`?`	匹配任意一个字符
`[list]`	匹配 list 中的任意单一字符
`[^list]`	匹配 除 list 中的任意单一字符以外的字符
`[c1-c2]`	匹配 c1-c2 中的任意单一字符 如：[0-9][a-z]
`{string1,string2,...}`	匹配 string1 或 string2 (或更多)其一字符串
`{c1..c2}`	匹配 c1-c2 中全部字符 如{1..10}

2.7 常见命令

2.7.1 用户及文件权限管理

- 查看用户 `who`

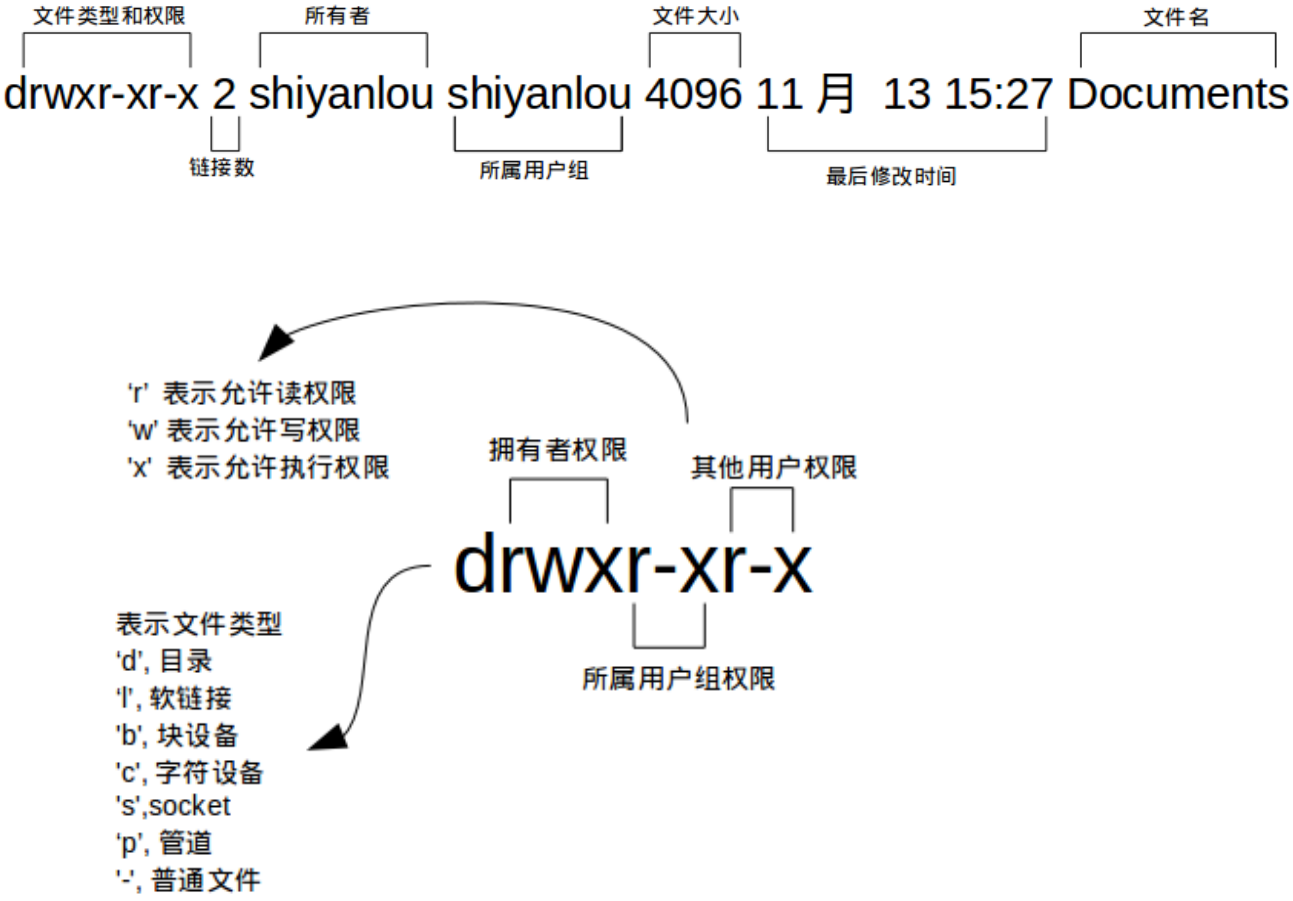
参数	说明
`-a`	打印能打印的全部
`-d`	打印死掉的进程
`-m`	同 `am i` ， `mom likes`
`-q`	打印当前登录用户数及用户名
`-u`	打印当前登录用户登录信息
`-r`	打印运行等级

- 新建用户
  - `sudo adduser lilei`
- 切换登录用户
  - `su -l lilei`
- 查询用户组
  - `groups shiyanlou`
  - 每次新建用户如果不指定用户组的话，默认会自动创建一个与用户名相同的用户组
- 将其它用户加入 sudo 用户组
  - 默认情况下新创建的用户是不具有 root 权限的，也不在 sudo 用户组，可以让其加入 sudo 用户组从而获取 root 权限
  - 首先切换回sudo账户

```
groups lilei
sudo usermod -G sudo lilei
groups lilei
```

```
shiyanlou:~/ $ groups lilei
lilei : lilei
shiyanlou:~/ $ sudo usermod -G sudo lilei
shiyanlou:~/ $ groups lilei
lilei : lilei sudo
shiyanlou:~/ $
```

- 删除用户
  - `sudo deluser lilei --remove-home`
- 删除用户组
  - 删除用户组可以使用 `groupdel` 命令，倘若该群组中仍包括某些用户，则必须先删除这些用户后，才能删除群组。
- 查看文件权限 `ls -l`



- 变更文件所有者 `sudo chown shiyanlou iphone11`
  - 将 `lilei` 的 `iphone11` 文件所有者改为 `shiyanlou`

2.7.2 目录结构及文件操作

- `find` 查找文件

示例

```
# 查找所有名称为src的文件夹
find . -name src -type d
# 查找所有文件夹路径中包含test的python文件
find . -path '*/test/*.py' -type f
# 查找前一天修改的所有文件
find . -mtime -1
# 查找所有大小在500k至10M的tar.gz文件
find . -size +500k -size -10M -name '*.tar.gz'
```

- 除了列出所寻找的文件之外，find 还能对所有查找到的文件进行操作。这能极大地简化一些单调的任务。

示例

```
# 删除全部扩展名为.tmp 的文件
find . -name '*.tmp' -exec rm {} \;
# 查找全部的 PNG 文件并将其转换为 JPG
find . -name '*.png' -exec convert {} {}.jpg \;
```

- `cd -` 进入上一次所在目录
- `pwd` 显示绝对路径
- `touch a.txt` 在当前目录下创建名为a的txt文件
- `cp test father/son/grandson` 将test文件复制到该目录
  - `cp -r father family` 将father目录复制到family目录（目录复制要加参数`-r`）
- `rm test` 删除文件（同理删除目录要加参数`-r`）
- `cat` 查看文件内容
- `echo` 看输出
- `more` 阅读文件内容
- `file` 查看文件类型
- 打包和解压文件
  - `tar -zcvf` 打包压缩后的文件名 要打包的文件
  - `tar -zxvf a.tar` //解压 a.tar文件（Linux下压缩的文件）
  - `unzip a.zip` //解压 a.zip文件（Windows下压缩的文件）

二、Vim

以下内容是Linux自带的`vimtutor`总结

1. 第一讲

1. 光标在屏幕文本中的移动既可以用箭头键，也可以使用 hjkl 字母键。
   `h（左移）　　j（下行）　　k（上行）　　l（右移）`
2. 欲进入 Vim 编辑器(从命令行提示符)，请输入：`vim 文件名 <回车>`
3. 欲退出 Vim 编辑器，请输入
   a. `:q!　<回车> 放弃所有改动`
   b. `:wq　<回车> 保存改动`
4. 在正常模式下删除光标所在位置的字符，请按：`x`
5. 欲插入或添加文本，请输入：

```
i    输入欲插入文本    <ESC>              在光标前插入文本
A    输入欲添加文本    <ESC>              在一行后添加文本
```

6. 按下 <ESC> 键会带您回到正常模式或者撤消一个不想输入或部分完整的命令。

2. 第二讲

1. 欲从当前光标删除至下一个单词，请输入：`dw`
2. 欲从当前光标删除至当前行末尾，请输入：`d$`
3. 欲删除整行，请输入：`dd`
4. 欲重复一个动作，请在它前面加上一个数字：`2w`
5. 在正常模式下修改命令的格式是：
   `operator　[number]　motion`
   其中：
   `operator` - 操作符，代表要做的事情，比如 d 代表删除
   `[number]` - 可以附加的数字，代表动作重复的次数
   `motion`　- 动作，代表在所操作的文本上的移动，例如 `w` 代表单词(word)，`$` 代表行末
   等等。
6. 欲移动光标到行首，请按数字0键：`0`
7. 欲撤消以前的操作，请输入：`u`

> 欲撤消在一行中所做的改动，请输入： `U`
> 欲撤消以前的撤消命令，恢复以前的操作结果，请输入： `CTRL-R`

3. 第三讲

1. 要重新置入已经删除的文本内容，请按小写字母 p 键。该操作可以将已删除的文本内容置于光标之后。如果最后一次删除的是一个整行，那么该行将置于当前光标所在行的下一行。
2. 要替换光标所在位置的字符，请输入小写的 r 和要替换掉原位置字符的新字符即可。
3. 更改类命令允许您改变从当前光标所在位置直到动作指示的位置中间的文本。比如输入 `ce` 可以替换当前光标到单词的末尾的内容；输入 `c$` 可以替换当前光标到行末的内容。
4. 更改类命令的格式是：

   > `c [number] motion`

4. 第四讲

1. `CTRL-G` 用于显示当前光标所在位置和文件状态信息。
   G 用于将光标跳转至文件最后一行。
   先敲入一个行号然后输入大写 G 则是将光标移动至该行号代表的行。
   gg 用于将光标跳转至文件第一行。
2. 输入 `/` 然后紧随一个字符串是在当前所编辑的文档中正向查找该字符串。
   输入 `?` 然后紧随一个字符串则是在当前所编辑的文档中反向查找该字符串。
   完成一次查找之后按 `n` 键是重复上一次的命令，可在同一方向上查
   找下一个匹配字符串所在；或者按大写 `N` 向相反方向查找下一匹配字符串所在。
   `CTRL-O` 带您跳转回较旧的位置， `CTRL-I` 则带您到较新的位置。
3. 如果光标当前位置是括号(、)、[、]、{、}，按 % 会将光标移动到配对的括号上。
4. 在一行内替换头一个字符串 old 为新的字符串 new，请输入 `:s/old/new`
   在一行内替换所有的字符串 old 为新的字符串 new，请输入 `:s/old/new/g`
   在两行内替换所有的字符串 old 为新的字符串 new，请输入 `:#,#s/old/new/g`
   在文件内替换所有的字符串 old 为新的字符串 new，请输入 `:%s/old/new/g`
   进行全文替换时询问用户确认每个替换需添加 c 标志 `:%s/old/new/gc`

5. 第五讲

1. `:!command` 用于执行一个外部命令 command。

   a. `:!ls` - 用于显示当前目录的内容。

b. `:!rm FILENAME` - 用于删除名为 FILENAME 的文件。

2. `:w FILENAME` 可将当前 VIM 中正在编辑的文件保存到名为 FILENAME 的文件中。
3. `v motion :w FILENAME` 可将当前编辑文件中可视模式下选中的内容保存到文件FILENAME中。
4. `:r FILENAME` 可提取磁盘文件 FILENAME 并将其插入到当前文件的光标位置后面。
5. `:r !dir` 可以读取 dir 命令的输出并将其放置到当前文件的光标位置后面。

6. 第六讲

1. 输入小写的 o 可以在光标下方打开新的一行并进入插入模式。
   输入大写的 O 可以在光标上方打开新的一行。
2. 输入小写的 a 可以在光标所在位置之后插入文本。
   输入大写的 A 可以在光标所在行的行末之后插入文本。
3. e 命令可以使光标移动到单词末尾。
4. 操作符 y 复制文本，p 粘贴先前复制的文本。
5. 输入大写的 R 将进入替换模式，直至按 <ESC> 键回到正常模式。
6. 输入 `:set xxx` 可以设置 xxx 选项。一些有用的选项如下：
   `'ic' 'ignorecase'` 查找时忽略字母大小写
   `'is' 'incsearch'` 查找短语时显示部分匹配
   `'hls' 'hlsearch'` 高亮显示所有的匹配短语
   选项名可以用完整版本，也可以用缩略版本。
7. 在选项前加上 no 可以关闭选项： `:set noic`

7. 第七讲

1. 输入 `:help` 或者按 <F1> 键或 <Help> 键可以打开帮助窗口。
2. 输入 `:help cmd` 可以找到关于 cmd 命令的帮助。
3. 输入 `CTRL-W` 可以使您在窗口之间跳转。
4. 输入 `:q` 以关闭帮助窗口
5. 您可以创建一个 `vimrc` 启动脚本文件用来保存您偏好的设置。
6. 当输入 : 命令时，按 `CTRL-D` 可以查看可能的补全结果。
   按 `<TAB>` 可以使用一个补全。

三、Git

1. 基础

- `git help <command>`：获取 git 命令的帮助信息
- `git init`：创建一个新的 git 仓库，其数据会存放在一个名为 `.git` 的目录下
- `git status`：显示当前的仓库状态
- `git add <filename>`：添加文件到暂存区
- `git commit`：创建一个新的提交
- `git log`：显示历史日志
- `git log --all --graph --decorate`：可视化历史记录（有向无环图）
- `git diff <filename>`：显示与暂存区文件的差异
- `git diff <revision> <filename>`：显示某个文件两个版本之间的差异
- `git checkout <revision>`：更新 HEAD 和目前的分支

2. 分支和合并

- `git branch`：显示分支
- `git branch <name>`：创建分支
- `git checkout -b <name>`：创建分支并切换到该分支
  - 相当于 `git branch <name>; git checkout <name>`
- `git merge <revision>`：合并到当前分支

3. 远端操作

- `git remote`：列出远端
- `git remote add <name> <url>`：添加一个远端
- `git fetch`：从远端获取对象/索引
- `git pull`：相当于 `git fetch; git merge`
- `git clone`：从远端下载仓库

4. 撤销

- `git commit --amend`：编辑提交的内容或信息
- `git reset HEAD <file>`：恢复暂存的文件
- `git checkout -- <file>`：丢弃修改