第一次实验

汤琦

23020007110

https://github.com/wlmrh/System-development-tool-basics

2024 年 8 月 25 日

目录

1 Latex简单使用

1.1 形成基础结构

使Latex支持中文字体: `\usepackage[UTF8]{ctex}`(使用CTex宏包)
全局去除Latex默认的段首空格: `\setlength{\parindent}{0pt}`(即将段首空格设置为0pt)
添加空行：`\hspace*{\fill}\\`（即用空格填充一行再换行，以此来达到视觉上的空行效果）
遇到报错：`Underfull \hbox (badness 10000)`, 报错原因为Latex无法很好的处理`\verb`指令, 应加入`\sloppy`启用宽松排版, 使其自动处理长行。

1.2 制作目录

1.2.1 问题一

在创建目录时, 发现起始页对应的序号为1

解决方法：
在正文开始前，使用命令`\setcounter{page}{1}`,将开始页的标号设置为1。

1.3 公式输入

与markdown类似

$$e = mc^2 \tag{1}$$

$$\pi = \frac{c}{d} \tag{2}$$

$$\frac{d}{dx}e^x = e^x \tag{3}$$

$$\frac{d}{dx}\int_0^\infty f(s)ds = f(s) \tag{4}$$

$$f(x) = \sum_i 0^\infty \frac{f^{(i)}(0)}{i!}x^i \tag{5}$$

$$x = \sqrt{\frac{x_i}{z}y} \tag{6}$$

1.4 字体样式

使用诸如 `\textit{words in italics}` 等指令, 可以改变字体。
导入`\usepackage{color}`导入包后可以通过`\color{`颜色名称`}`来改变字体颜色, 该项默认作用于全局，可以使用`{}`来缩小该指令的作用域。

使用`\tiny, \scriptsize, \large`等指令可以调整字体大小, 如:
`{\color{cyan}\Large text}`指令的执行效果如下, 其颜色为cyan, 大小为large
text

使用`\colorbox{magenta}{\color{cyan}\Huge Hello}`的执行效果如下, 其背景为magenta, 字体颜色为cyan, 字体大小为huge
Hello

1.5 表格制作

先确定列数, 列之间是否有竖线，每一列的对齐方式: 左对齐(l), 右对齐(r), 向中对齐(c), 将这些参数写到begin指令的右边, 然后使用指令画出横线, 下面是oiwiki中表格部分对应的两道练习题。

Item	Quantity	Price($)
Nails	500	0.34
Bricks	240	11.50

	Year		
City	2006	2007	2008
London	45789	46551	51298
Berlin	34549	32543	29870
Paris	49835	51009	51970

2 Git简单使用

阅读了Pro Git中的部分内容,下面是对课程配套课后练习的作答。

2.1 Exercise2

克隆本课程网站的仓库
在连接中查询到本课程的仓库地址，在目标文件夹中右键点击在终端中打开, 输入指令

```
git clone https://github.com/missing-semester-cn/
missing-semester-cn.github.io.git
```

1. 将版本历史可视化并进行探索
   通过命令`git log`来显示历史日志，通过参数`--graph`来获取可视化视图

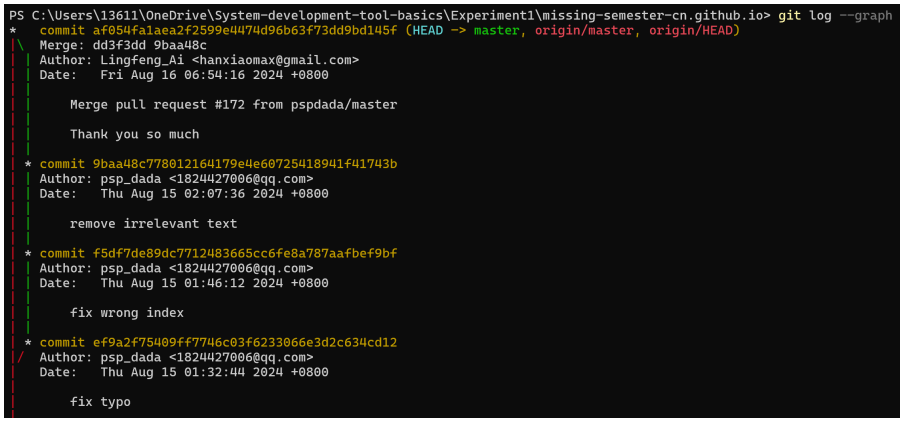


图 1: 版本历史可视化

2. 是谁最后修改了 `README.md` 文件？（提示：使用 `git log` 命令并添加合适的参数）
   使用指令`git log -1 (--) README.md`，其中-1表示只显示最后一次的commit, --告诉Git后面的是文件路径而不是分支名, 但在本题中不存在冲突, 可以省略, `README.md`则将搜索范围限制到对`README.md`的修改。

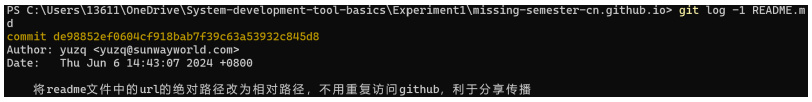


图 2: README.md最后一次修改

3. 最后一次修改 `_config.yml` 文件中 collections: 行时的提交信息是什么？(提示：使用 `git blame` 和 `git show`)
   先运行`git blame _config.yml`获取对 `_config.yml` 进行的所有修改, 通过管道运算符, 将其输出作为 grep 指令的输入, 筛选出对 collections 行的提交信息。

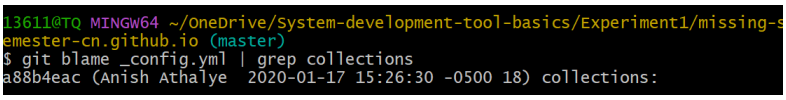


图 3: 最后一次行修改

2.2 Exercise4

从 GitHub 上克隆某个仓库，修改一些文件。当您使用 `git stash` 会发生什么？当您执行 `git log --all --oneline` 时会显示什么？通过 `git stash pop` 命令来撤销 `git stash` 操作，什么时候会用到这一技巧？
运行`git stash`命令后, 使用`git status`命令, 发现之前做的修改消失了。

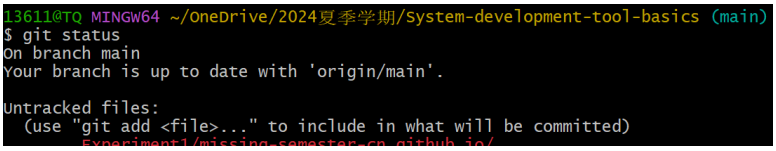
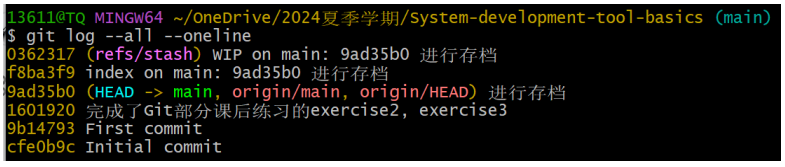
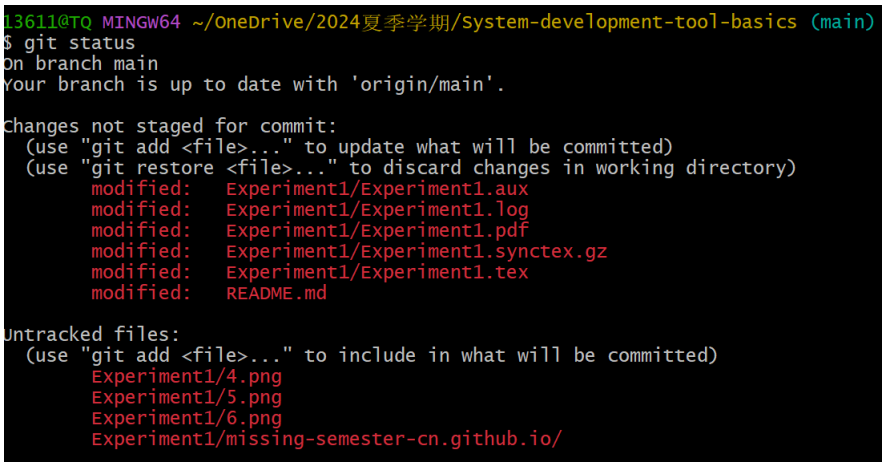


图 4: 运行后的结果

执行`git log --all --oneline`后, 运行结果如下：



看到之前做的修改被保存为 main 分支上的WIP(work in progress)，只是从文件夹中被移除。在运行指令`git stash pop`指令后发现之前修改的`README.md` 又回到了Changes not staged for commit 中



推测 `git stash`是为了在本地暂存已进行的修改而不进行提交, 以进行其他活动。使用场景可能是在进行功能开发时, 发现有其他bug需要维修这时可以在本地暂存已进行的开发, 而不用进行一次无用的提交来达到缓存的目的。

2.3 Exercise5

与其他的命令行工具一样，Git 也提供了一个名为 /.gitconfig 配置文件(或 dotfile)。请在 /.gitconfig 中创建一个别名，使您在运行 git graph 时，您可以得到 git log –all –graph –decorate –oneline 的输出结果；只需在`.gitconfig`文件中添加

```
[alias]
graph = log --all --graph --decorate --oneline
```

结果如下：