

lab0_Shell与实验环境补充

本教程为lab0助教个人整理，覆盖可能不是很全，如有错误之处敬请指出。

前三小节是bash编程内容，后面几小节是工具的使用。

echo

echo之后支持传入多个参数，将解析每个空格分隔的参数（可解析\$代表的变量，但不包括命令执行）并输出。

yourid@stu-x:~\$ 代表在命令行中输入，请不要输入这段内容。

```
yourid@stu-x:~$ echo echo Hello world
echo Hello world
```

可使用反引号，使得被括起来的内容转换为bash执行该指令的标准输出（stdout）。

```
yourid@stu-x:~$ echo `echo Hello world` `date`
Hello world Thu Feb 25 10:20:30 CST 2021
```

单引号、双引号和反引号

- 单引号代表忽略所有特殊字符（包括 \$，\ 和反引号）
- 双引号不忽略以上三个特殊字符
- 反引号则会被替换为**被括指令**的标准输出

单引号引起来的字符串会忽略所有特殊字符（包括 \$，\ 和反引号），不作变量解析、字符转义操作，只作为普通纯文本。

双引号不忽略以上三个特殊字符，会执行变量解析、字符转义操作。

考虑test.sh中如下两行：（同时观察 \$1 颜色的不同）

```
echo "hello $1"
echo 'hello $1'
```

尝试运行 `bash test.sh world!`

变量赋值、运算与使用

如果只包含数字，变量被识别为整数型（十进制），为了调用变量，可在之前加入\$符号。\$之后遇到第一个不合法变量字符将自动分割，否则将被视为整体，可使用双引号分割。

注意，以下命令均在命令行中直接执行

```
$ var=5
yourid@stu-x:~$ echo $var
5
yourid@stu-x:~$ echo $var 5
5 5
yourid@stu-x:~$ echo $var5
```

```
# (空白, 因为变量var5不存在)
yourid@stu-x:~$ echo "$var"5
55
yourid@stu-x:~$ var="$var"5
yourid@stu-x:~$ echo $var
55 # 此时var仍然被认为是整数变量, 或者不如说在进行赋值和比较的时候才检查其是否是整数变量, 可一直认为它是字符串。
yourid@stu-x:~$ let var=$var+1
yourid@stu-x:~$ echo $var
56 # 另一种赋值方式
```

注意, 直接采用赋值将被识别为字符串 (因为存在+号)。

```
yourid@stu-x:~$ var=$var+5
yourid@stu-x:~$ echo $var
5+5
```

值得一提的是, bash的空格切记不要滥用, 其作为分隔符在bash中有特殊意义。

```
yourid@stu-x:~$ var=1 # good :)
yourid@stu-x:~$ var = 1 # won't work :(
```

但在某些情况下需要使用空格分离字符, 以下命令在**bash脚本**中执行

```
a=1
while [ $a -ne 10 ] # 10后没有空格直接接上], 将导致10]变为字符串整体, bash检查后发现缺少], 不会工作
while [ $a -ne 10 ] # 正确的版本
do
...
done
```

tmux (好用, 尽快学会)

命令行输入 `tmux` 即可进入tmux界面, 看起来和正常的界面没什么不同。下面介绍**常用操作, 窗格和窗口操作一定要掌握** (精选过了)。

如下操作均在tmux中完成, 需要先进入tmux界面, 看到下方的窗口序号即进入tmux。

```
zhangzongye@stu-111:~/test$  
  
[1] 0:bash* "stu-111" 22:24 09-Mar-21
```

以下操作均需要先按ctrl+b，然后再按对应按键，如ctrl + b 然后按 %

窗格操作（类似Windows一个窗口中分屏）

分屏功能，之后写代码必备！

- % 水平分屏
- " 垂直分屏
- 方向键（上下左右）：切换到对应方向的分屏
- o 依次切换
- z(zoom) 最大化当前窗格
- x(close) 关闭当前窗格

窗口操作（类似Windows中多个窗口）

一个屏幕只能分两屏？（不然太小不好看）没关系，还可以开多窗口。

- c 创建新的窗口
- p 切换上一个(previous)窗口（上下窗口类似alt+tab）
- n 切换下一个(next)窗口
- 0 切换到第0个窗口，可在下方看到窗口数量与编号，星号代表选中。同理可切换到1~9号窗口（如果存在）

git冲突

在实验过程你可能会遇到过这样的情况：原来评测得了60分的代码（版本1），一通修改之后再提交（版本2），分数反而变得更低了；于是你再用 `reset --hard` 回到了60分的版本重新修改，可是再次提交时（版本3）产生了如下报错信息：

```
To git@xxxxxx:19xxxxxx-lab
! [rejected]        labx -> labx (non-fast-forward)
error: failed to push some refs to 'git@xxxxxx:19xxxxxx-lab'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes (e.g. 'git pull') before pushing again.  See the
'Note about fast-forwards' section of 'git push --help' for details.
```

这是因为本地分支和远程分支产生了分叉，根据需求不同解决方法如下：

同时保留版本2和版本3的改动（可能需要手动合并）

```
git pull 从远程仓库获取版本2的改动 （如果报错，解决方法参考指导书1.5.8 git 冲突与解决冲突）
git push
```

只保留版本3的改动

此时通过 `git reflog` 查看历史改动：

```
82b83da HEAD@{0}: commit: xxxxxxxxx
16e2bb3 HEAD@{1}: reset: moving to 16e2bb373643aabdd3fe111b41eee829dfbe3c89
0a823b9 HEAD@{2}: commit: xxxxxxxxx
```

HEAD@{1}记录了reset操作，回到了版本1；HEAD@{0}是reset之后的commit的版本3；HEAD@{2}是版本2的提交。此时通过：

```
git reset --soft 0a823b9 （把HEAD修改到版本2位置，和远程仓库保持一致，但是代码不改动）
git commit --allow-empty -m "blabla..."
git push labx:labx
```

即可正确触发评测

此外，如果你曾经把 `labx-result` 拉取到本地查看，reset之后如果使用git push可能会产生冲突。建议push时候加上确定的分支名称，例如labx:labx

关于课下测试的一些提示

课下测试需要完成前4步之后才能正确触发评测，会从dst中拉取文件测试

Makefile嵌套的可能写法：

1. 通过文件目录暴力调用内层（比较繁琐）
2. 进入子目录执行make
3. 直接调用子目录的make（`make --directory=code`）

注意：Makefile中需要引入头文件目录

扩展阅读

如果你希望学习更多Linux脚本编程知识，建议阅读如下书籍。

《Linux命令、编辑器与Shell编程》Mark G. Sobell

《UNIX编程环境》Brian W. Kernighan、Rob Pike