第一章 操作系统及UNIX Shell

- 1.5 Unix/Linux Shell概述
- 1.6 Unix/Linux Shell 命令
- 1.7 Unix/Linux Shell 命令进阶
  - 1.7.1 假名（别名）机制
  - 1.7.2 命令行编辑和命令补齐（Bash/Cshell）
  - 1.7.3 工作目录栈（Bash/Cshell）
  - 1.7.4 命令史机制（C shell ）
- 1.8 Unix/Linux Shell编程

➢1.7.1 假名(别名)机制

假名(别名)是指在C shell中将已有的命令重新起名。

- 设置假名的命令格式

% alias [*假名* [*命令*]]	设置或者显示假名
% unalias *假名*	撤消假名

alias命令有以下三种使用形式：

% alias *假名 命令*	将*命令*起名为*假名*
% alias *假名*	显示*假名*所定义的值
% alias	显示所有假名的值

注：B shell/Bash也支持假名机制，格式：

$ alias *假名=命令*

$ unalias *假名*

➢假名应用示例

例如，显示ls的假名：

% alias ls

ls -CF 假设这是管理员预先定义的假名

取消ls原有的假名：

% unalias ls

% alias ls 原定义的假名消失

%

重新给ls起名：

% alias ls “ls -a” 使ls命令可以显示隐含目录/文件

% alias ls

ls -a

假名可以组合，例如：

% alias cd “cd ; pwd” 定义假名为组合命令

% alias cd

cd ; pwd

% cd

/mnt/eejm/me00 显示注册目录

➢1.7.2 命令行编辑和补齐（Bash / C shell）

➢命令行编辑

在bash中，可以用上下箭头键(↓和↑)来查看过去的命令，以便再次执行命令，或者修改命令行的内容后执行新的命令。例如：

$ find□/usr□-name□csh　　在/usr目录中查找csh

$　　没有找到，无显示。

此时可以使用向上箭头键↑，以显示上一条命令，即：

$↑　　将显示

$ find□/usr□-name□csh

然后再用左右箭头键(→ 和←)在命令行中移动，用回退键<BS>(Backspace)删除字符，键入新的字符，改为：

$ find□/bin□-name□csh<CR> usr被改为bin，执行

/bin/csh　　在/bin/csh下找到csh

可以在执行了若干条命令后再向上查看过去的命令，而不局限于查看刚刚执行过的命令，只是需要多按几次向上箭头键↑，如果过头了，则再按向下箭头键↓直到找到为止。

➢1.7.2 命令行编辑和补齐（Bash / C shell）

➢命令行自动完成

Bash提供命令行自动完成(command-line completion)功能，即在键入命令时，可以使用制表符<TAB>帮助用户自动完成命令或者自动完成文件名，或者称为命令或目录/文件名的自动补齐。

➢ 命令的补齐

当键入命令的前几个字符时，按<TAB>键，Bash会根据系统中确实存在的命令给予补齐。例如，命令switchdesk的功能是切换面板(panel)，但是字符数较多，可利用这一功能来补齐命令：

$ swi<TAB>　　<TAB>是一个键符，不显示

在字符swi后按<TAB>键，Bash会自动将命令补齐，立即显示：

$ switchdesk　　后续字符tchdesk是自动补齐的

➢1.7.2 命令行编辑和补齐（Bash / C shell）

- 目录/文件名的补齐

echo、ls、cd、cp和mv等命令的参数是目录或者文件，当键入目录/文件名的前几个字符时，按<TAB>键，Bash会根据系统中确实存在的目录/文件名给予补齐。

➢1.7.2 命令行编辑和补齐（Bash / C shell）

➢目录/文件名的补齐

【例1-21】目录/文件名补齐示例

```
$ ls
a.out     c          shell          假设c和shell是目录
$ ls shell
proj1     proj2      proj.c         projtest.c        shell下有四个文件
$ ls s<TAB>                         在ls s后按<TAB>, 将补齐为shell
$ ls shell<CR>                      按回车键<CR>后执行命令
proj1     proj2      proj.c         projtest.c
$ vi s<TAB>                         在vi s后按<TAB>, 将补齐为shell
$ vi shell/p<TAB>                   继续键入/p，再按<TAB>
$ vi shell/proj                     可匹配到proj
$ vi shell/projt<TAB>               继续键入t，再按<TAB>
$ vi shell/projtest.c<CR>
```

可匹配到projtest.c，再按回车，将执行命令vi src/projtest.c。

➢1.7.2 命令行编辑和补齐（Bash / C shell）

C shell也提供目录/文件名的自动补齐功能。与Bash不同的是：

1) 必须先执行命令set filec，以启动文件名补齐功能。

2) 在执行以目录或者文件为参数的echo、ls等命令时，请求文件名补齐的是<ESC>键。

3) 如果按<ESC>键不能完成自动补齐功能，说明可识别的文件名模糊难辨，则可以按^D键，请求列出所有可能匹配的目录/文件名。

➢1.7.2 命令行编辑和补齐（Bash / C shell）

【例1-33】目录/文件名补齐示例

```
% set filec                                  启动文件名补齐功能
% ls
a.out     c          shell                  假设c和shell是目录
% vi s<ESC>                                  在vi s后按<ESC>, 将补齐为shell
% vi shell/^D                                按斜杠/及^D，
                                             请求列出shell下的所有文件
a.out     proj2     proj.c    projtest.c    tmp/     4个文件1个目录
% vi shell/p<ESC>                            继续键入p，再按<ESC>
% vi shell/proj                              可匹配到proj
% vi shell/projt<ESC>                        继续键入t，再按<ESC>
% vi shell/projtest.c<CR>                    可匹配到projtest.c，再按回车，
                                             将执行命令vi shell/projtest.c
```

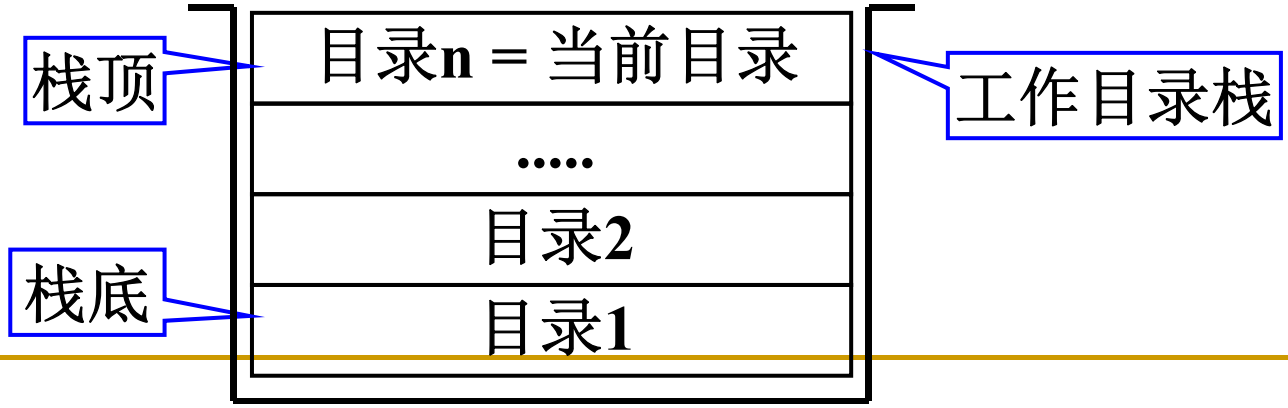
➢1.7.3 工作目录栈（Bash/C shell）

➢什么是工作目录栈

cd命令的功能是改变当前的工作目录，但是如果需要在数个目录中频繁地来回切换，仅用**cd**命令就显得不很方便。**Bash**和**C shell**设置了一个工作目录栈，允许引用栈中的目录，以便方便地切换目录。

➢工作目录栈

工作目录栈的作用是从栈底向栈顶依次存入目录，从栈顶向栈底依次取出目录。而且总是将栈顶目录作为当前目录。

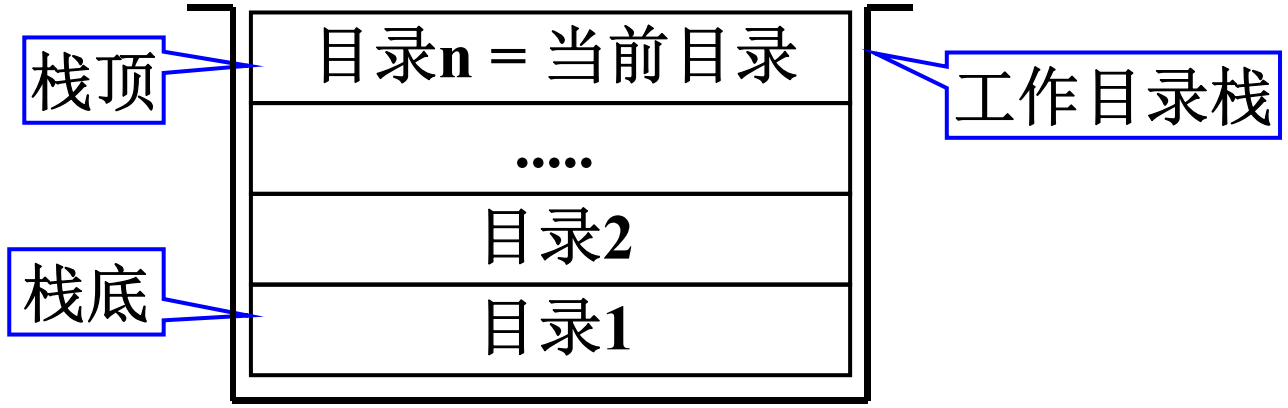


➢1.7.3 工作目录栈（Bash/C shell）

➢工作目录栈

工作目录栈命令：

pushd [*目录*]	将当前目录或者*目录*放入工作目录栈
popd	从工作目录栈中取出栈顶目录
dirs	查看目录栈中内容
cd [*目录*]	执行cd命令的同时会将栈顶目录改为新的*目录*



为了简便使用，设定假名为：

$ alias pd=pushd 用pd代替pushd

$ alias po=popd 用po代替popd

【例1-22】工作目录栈应用示例

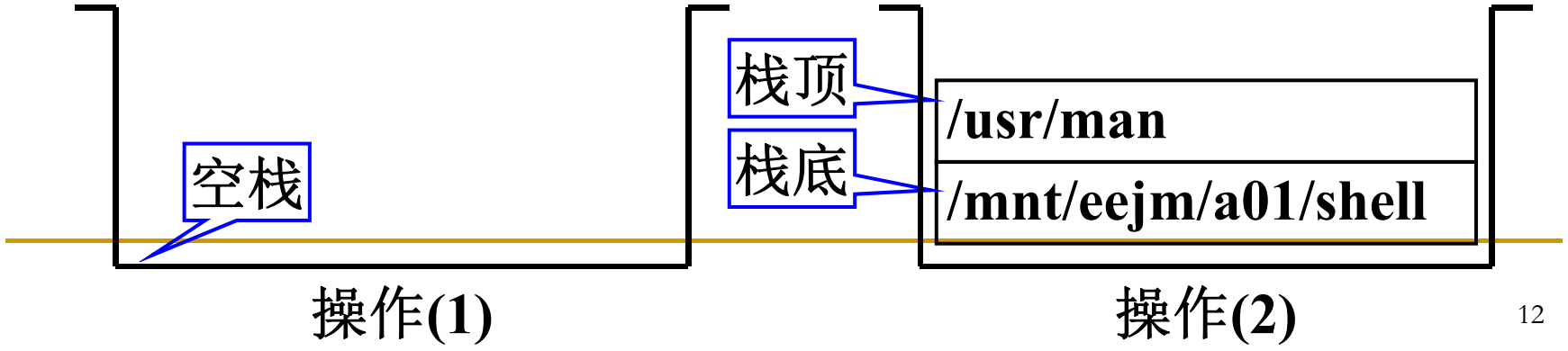
假定注册目录和当前工作目录分别为
/mnt/eejm/a01以及/mnt/eejm/a01/shell，即有：

```
$ pwd                          显示当前目录
/mnt/eejm/a01/shell
```

操作示例如下：

```
(1) $ pd                       由于初始栈为空栈，将显示出错信息
(2) $ pd /usr/man
    /usr/man  /mnt/eejm/a01/shell
    $ pwd                      显示当前目录
    /usr/man                     当前目录为栈顶目录
```

首次使用pd，栈中将存入两个目录：栈底为执行pd命令之前的当前目录/mnt/eejm/a01/shell，栈顶为现行当前目录/usr/man。



【例1-22】工作目录栈应用示例

(3) $ pd man1　　再将man1放入栈中，则栈顶为/usr/man/man1

/usr/man/man1　/usr/man　/mnt/eejm/a01/shell

$ pwd

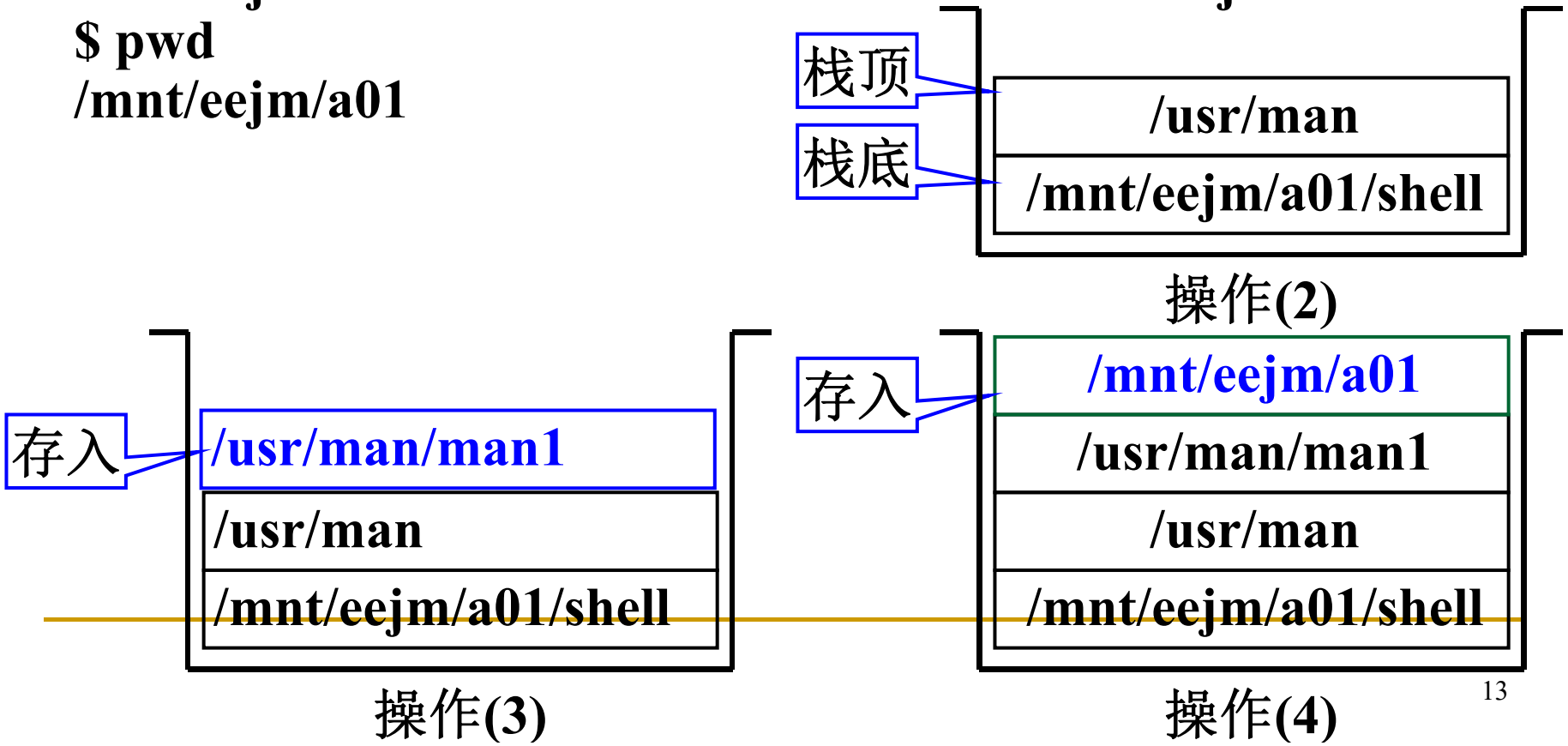
/usr/man/man1

(4) $ pd ~　　再将~放入栈中，则栈顶为/mnt/eejm/a01

/mnt/eejm/a01 /usr/man/man1　/usr/man　/mnt/eejm/a01/shell

$ pwd

/mnt/eejm/a01



【例1-22】工作目录栈应用示例

(5) $ pd　　没有参数，则把栈顶的两个目录互换

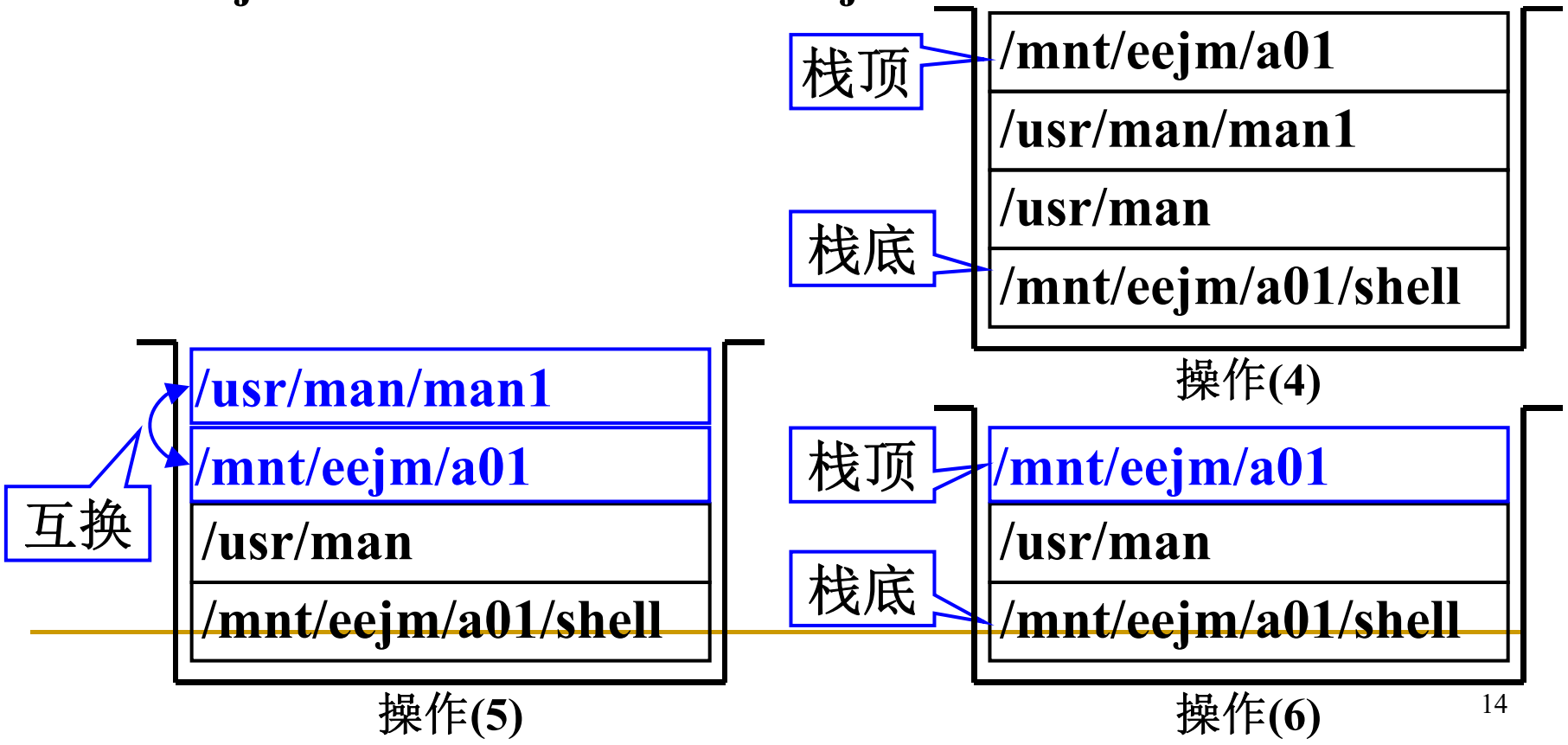
/usr/man/man1 /mnt/eejm/a01 /usr/man /mnt/eejm/a01/shell

$ pwd

/usr/man/man1

(6) $ po　　删去栈顶目录，将栈顶下一目录作为栈顶目录

/mnt/eejm/a01 /usr/man /mnt/eejm/a01/shell

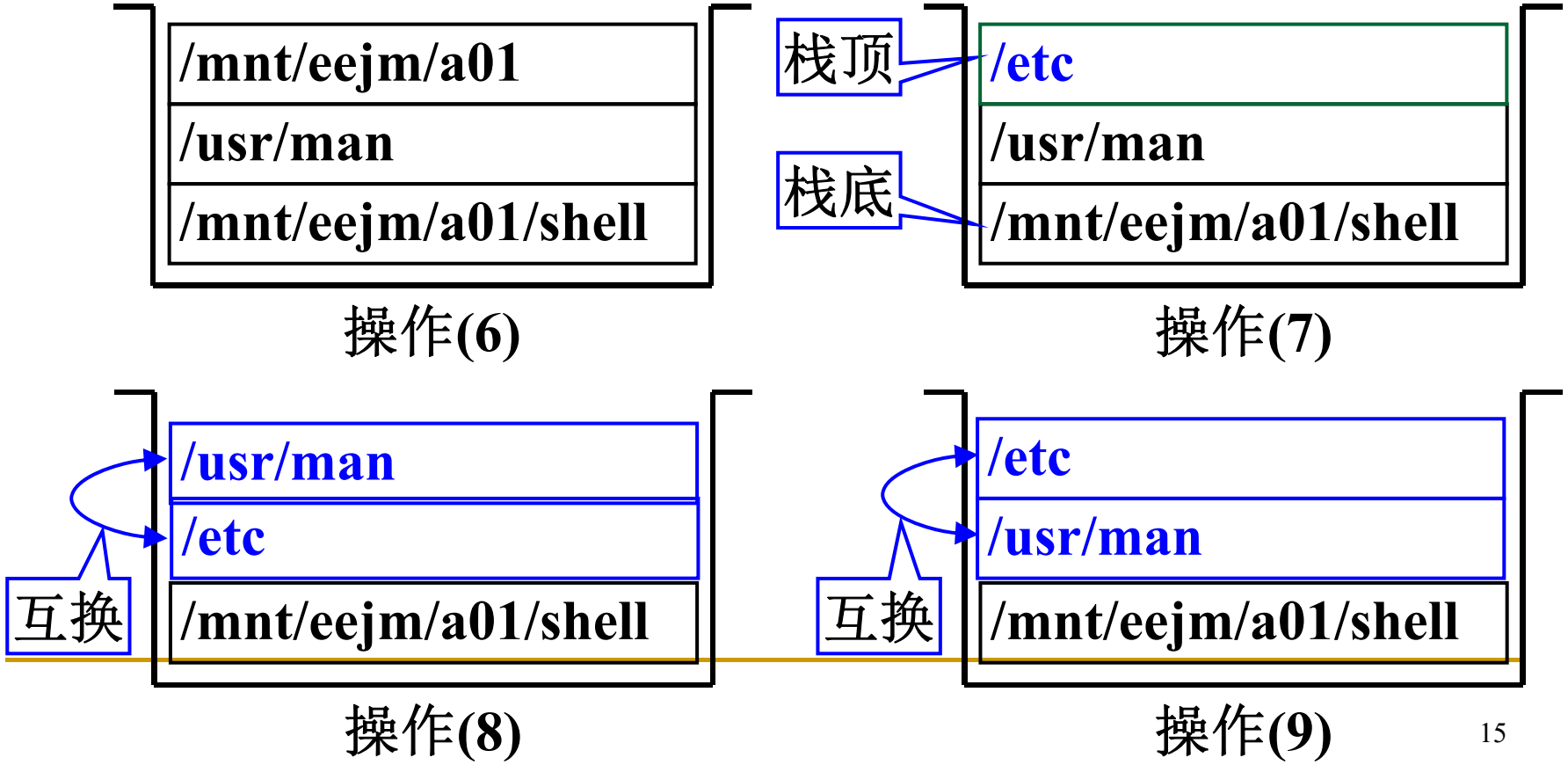


【例1-22】工作目录栈应用示例

(7) $ cd /etc　　栈顶目录/mnt/eejm/a01将改为/etc

(8) $ pd　　没有参数，则把栈顶的两个目录互换
/usr/man /etc /mnt/eejm/a01/shell

(9) $ pd　　没有参数，则把栈顶的两个目录互换
/etc /usr/man /mnt/eejm/a01/shell



➢1.7.4 命令史机制

- 命令史定义

C shell设置了一个命令缓冲区(存储空间)，记录先前执行过的命令，并且提供了显示和引用这些命令的功能，称为命令史机制。

【注】命令史是C shell和Bash都提供的功能。

与命令史相关的环境变量为history，即显示命令的数量。相关命令为history，用以显示history条先前执行过的命令。为了便于理解，在配置文件中增加语句 set prompt="[\!]>"，表示将待命符设置为"[命令号]>"，其中特殊变量"!"表示当前命令的序号(命令号)。

➢1.7.4 命令史机制

执行以下命令：

[5]> set history=6　　设置命令记录数为6(当前命令号为5)

[6]> history　　显示命令史栈中的内容

1 ls -l

2 cat main.c

3 vi main.c

4 cc -c main.c

5 set history=6

6 history

[7]>

命令史引用示例

假定设置命令history的假名为h，即已有定义为：

alias h history

则执行命令：

[7]>h 显示命令史栈

2 cat main.c

3 vi main.c

4 cc -c main.c

5 set history=6

6 history

7 h

[8]>

命令史引用示例

(1) 使用命令号引用栈中的命令

格式为： !*n*　　*n*表示需要引用命令的命令号

[8]> !1　　使用命令号引用并执行第1条命令ls -l

ls -l

.....*执行命令*ls -l

[9]> !-2　　用相对命令号引用前第2条命令(7)h

h

4 cc -c main.c

5 set history=6

6 history

7 h

8 ls -l

9 h

[10]>

命令史引用示例

(2) 重复前一条的格式为： !!

```
[10]> !!
h
5 set history=6
6 history
7 h
8 ls -l
9 h
10 h
[11]>
```

(3) 使用前导匹配字符

格式为：!*xxx*　　*xxx*为需要匹配的字符串

```
[11]> !c          匹配到最近以c为字首的命令cc
cc -c main.c
[12]> !ca         匹配到最近以ca为字首的命令cat
cat main.c
[13]>
```

命令史引用示例

(4) 显示某条命令，但不执行

格式为： **:p**

[13]> !ca:p 显示以ca为字首的命令，但不执行

cat main.c

[14]>

(5) 修改上一命令中的内容后执行

格式为： **^*匹配的字符串*^*新的字符串*[^]**

[14]> ^main^../tst/try^ 将上条命令的main改为../tst/try

cat ../tst/try.c 将cat main.c改为cat ../tst/try.c

[15]>

(6) 引用以前命令的某个命令行参数

格式为： **:*nn*** ***nn***为参数的序号

[15]> ls main.c try.c

main.c try.c

[16]> cc -c !!:2 引用上条命令的第2个参数try.c

cc -c try.c

[17]>

命令史引用示例

(7) 引用前一命令的全部参数

格式为： !*

```
[17]> !15:p                显示第15条命令
ls main.c try.c
[18]> cp !* ../tmp         引用上条命令的全部参数
cp main.c try.c ../tmp
[19]>
```

(8) 引用前一命令的最后一个参数

格式为： !$

```
[19]> ls !$                引用上条命令的最后一个参数../tmp
ls ../tmp
[20]>
```

第一章 操作系统及UNIX Shell

- 1.5 Unix/Linux Shell概述
- 1.6 Unix/Linux Shell 命令
  - 1.6.1 单行和多行命令
  - 1.6.2 输入输出定向
  - 1.6.3 文件名的通配符
  - 1.6.4 Shell变量
  - 1.6.5 特殊字符
- 1.7 Unix/Linux Shell 命令进阶
  - 1.7.1 假名（别名）机制
  - 1.7.2 命令行编辑和命令补齐（Bash/Cshell）
  - 1.7.3 工作目录栈（Bash/Cshell）
  - 1.7.4 命令史机制（C shell ）
- 1.8 Unix/Linux Shell编程

➢作　　业

- 习题　　E-1　　1-14
  E-6～E-7：　1-23.1（别名）
  1-23.2（工作目录栈）
  1-23.3（命令史）
  1-23.4（工作目录栈）
- 上机操作、复习课堂讲过的命令
- 上机习题：
  (Bash)E-2: 1-16.3;
- 自学上机指南二：vi快速入门
- C语言编程练习 E-18：3-4(每周至少完成3题)