용어정리

1. Redirection

: 파이프(|)와 같이 명령어의 결과로 다른 명령어를 실행하거나 기록할 때 쓰는 명령어.

ex) > (덮어쓰기) . >> (추가) . < (명령어에 내용입히기)

2. Pipe

: 명령어1 | 명령어2 일때 명령어 1을 실행한 결과값에 명령어 2 실행

ex) (cat aa.txt | grep abc : aaa.txt에서 abc를 포함하는 내용만 출력하라)

정리하기

1-1) 쉘과 쉘스크립트란?

① 쉘 (Shell) : 커널과 사용자간 인터페이스로 중간매개 역할을 한다.
입력받는 명령어를 해석하고 관련 유틸이나 커널을 호출 - 실행결과 출력을 한다.
쉘에도 여러 종류가 있으나 리눅스에서는 bash(배쉬)을 많이 쓴다. 명령어는 sh ≒ bash 지만 bash 명령어가 기능이 더 많다

② 쉘스크립트 : 명령어를 여러개 묶어서 실행하거나. 쉘명령어를 나열한 것을 "~.sh" 파일에 기록한 것을 쉘 스크립트, 쉘 프로그램 이라고 한다.
컴파일이 필요없어 실행이 빠르고 시스템운영이나 유지보수에 유리하나 복잡한 연산처리나 다른 OS에 이식하기 어렵다는 제한 점이 있다.

1-2) 쉘 스크립트 실행방법

- 가장간단하게는 ;로 명령어 사이에 구분을 두어 묶어서 실행 할 수 있다.
- 쉘스크립트는 sh나 bash 명령어의 인자로서 ~.sh 파일을 넣어 실행할 수 있다.
  ex. sh my.sh / bash my.sh
- 혹은 chmod a+x (u+x) filename 으로 실행권한을 준 뒤, 상대 · 혹은 절대경로를 입력하여 실행이 가능하다.
  ex. ./my.sh / home/kopo08/my.sh
- export PATH = $PATH: /home/kopo08 과 같이 예약어를 통해 기본 경로에 my.sh의 위치를 입력하고 실행도 가능하다.

2-1) 리다이렉션

명령의 출력값을 화면에 출력하는 대신 다른 장치나 파일등에 출력하여 저장할 때 사용

> 덮어쓰기

>> 내용추가

< 명령어에 대한 내용 입력 → 거의 안씀

2-2) 파이프

"명령어1 | 명령어2" 의 꼴로 있을 때 명령어 1의 결과를 받아 명령어 2를 실행. 특히 more.sort.grep에 많이 쓰인다.

2-3) echo

출력 명령어.
echo $변수 = "$변수" : 변수의 값 출력 = 문자열도 같음
echo '$변수' : $변수라는 문자열 그 자체를 출력
echo -n : echo는 한 줄 쓰는게 기본적으로 개행이 들어가는데, -n은 개행을 없애는 옵션이다

2-4) cut

텍스트나 명령어의 결과를 크롭하여 추출할 때 쓰는 명령어

ⓥ -b (바이트) - c (글자 (char)) ※ 한글은 1글자당 2byte

ex) -bN : N번째 자리 글자 출력
-bN.M.O : N.M.O번째 자리 글자 각각 출력
-bN- : N번째부터 끝까지 출력
-bN-M : N번째부터 M번째까지 출력
-b-N : 처음부터 N번째까지

ⓥ -d (delimeter: 구분자) -fN ex. ps | cut -d ' ' -fN 구분자가 ' ' (공백)이 되어 PS 실행후 공백 기준으로 N번째 문자열 출력

ㅁ = 공백

자바의 split이랑 범위 계산이 다르니 유의할 것

3. 이해하기

- Dos의 command.com과 같이 기본 제공 OS명령어를 수행하는 곳.
- 윈도우에서 탐색기의 왼쪽 마우스 메뉴의 압축 기능과 같은 기본 명령임.
- 최초 유닉스에서는 ksh (콘쉘), 이후 발전된 csh(씨쉘) 등이 사용되었으나, 리눅스에서는 bash(배쉘)이 많이 사용됨
- bash는 명령어의 문법은 거의 대부분의 sh와 호환되어 쓰임

② 쉘 스크립트, 쉘 프로그램

- 하나의 쉘 명령어를 여러 개 묶어서 실행가능
- 쉘 명령어를 나열하여 text file로 기록하여 이를 실행 함
- 이러한 묶음 명령어를 쉘 스크립트 또는 쉘 프로그램 이라고 함

③ 쉘 스크립트의 장점

- 타 프로그래밍 언어에 비해 실행속도가 빠름
- 컴파일 과정이 필요 없이 빠르게 실행가능
- 시스템 운영이나 유지보수때 사용하기 용이함

Tip 일반적인 C,Java등의 언어는 프로그램 언어를 번역하여 실행파일을 만드는 과정을 거쳐서 실행

④ 쉘 스크립트의 제한 사항

- 다중 산술 작업이나 복잡한 정보 처리작업에 사용하는 방식은 아님
- 유닉스, 리눅스 이외 다른 운영체계에 이식은 어려움

Tip 리눅스 쉘 프로그래밍을 잘 하기 위하여 무엇보다도 많이 사용하는 쉘 명령어를 잘 다룰 수 있어야 한다

3. 이해하기

```
kopoctc@ubuntu:~$ vi my.sh
cd /home
ls -l
pwd
~
:wq
```

<그림 Ⅲ-2> my.sh파일을 작성 후 저장

기본적으로 만들어지면
권한이 없어서 실행불가
⇓
sh, bash → 쉘을 이용하여 실행하거나
권한을 부여하여 실행

- 해당 파일명으로 명령어 실행
- 하지만 쉘 스크립트 파일을 실행하기 위하여 쉘 파일의 권한과 패스를 고려하여야 하기 때문에 실행 안됨

```
kopoctc@ubuntu:~$ pwd
/home/kopoctc
kopoctc@ubuntu:~$ vi my.sh
kopoctc@ubuntu:~$ my.sh
my.sh: command not found
kopoctc@ubuntu:~$
```

<그림 Ⅲ-3> 권한과 패스의 고려없이 my.sh파일을 실행

③권한

- sh shell_script_file 형식으로 실행 ex) sh my.sh (shell script 확장자)

3. 이해하기

```
kopoctc@ubuntu:~$ vi my.sh
cd /home
ls -l
pwd
~
:wq
```

<그림 III-2> my.sh파일을 작성 후 저장

기본으로 만들고나면 권한이 없어서 실행불가
sh / bash → 쉬운을 이용하여 실행하거나 권한을 부여하여 실행

- 해당 파일명으로 명령어 실행
- 하지만 쉘 스크립트 파일을 실행하기 위하여 쉘 파일의 권한과 패스를 고려하여야 하기 때문에 실행 안됨

```
kopoctc@ubuntu:~$ pwd
/home/kopoctc
kopoctc@ubuntu:~$ vi my.sh
kopoctc@ubuntu:~$ my.sh
my.sh: command not found
kopoctc@ubuntu:~$
```

<그림 III-3> 권한과 패스의 고려없이 my.sh파일을 실행

③권한

- sh shell_script_file 형식으로 실행 ex) sh my.sh (shell script 실행파일)

3. 이해하기

```
kopoctc@ubuntu:~$ sh my.sh
/home/kopoctc
total 72
-rw-r--r-- 1 root    root    51810 Dec 19 18:21 aa
-rw-r--r-- 1 root    root      476 Dec 20 20:48 a_out.log
-rwxr-xr-x 1 root    root       61 Dec 20 19:32 a.sh
-rw-r--r-- 1 root    root      290 Jan 17 17:57 boot.log
drwxr-xr-x 3 kopoctc kopoctc  4096 Jan 17 18:07 kopoctc
-rw-r--r-- 1 root    root       16 Jan 12 22:21 mytest.log
/home
kopoctc@ubuntu:~$
```

<그림 III-4> sh my.sh 실행

- 또는 쉘 스크립트 파일이 실행권한을 부여 후 실행 ex) chmod 755 my.sh
  chmod u+x
  a+x

3. 이해하기

```
kopoctc@ubuntu:~$ ls -l my.sh
-rw-rw-r-- 1 kopoctc kopoctc 23 Jan 17 18:07 my.sh
kopoctc@ubuntu:~$ chmod 755 my.sh
kopoctc@ubuntu:~$ my.sh ㄱ경로지정이 안되어있음
my.sh: command not found
kopoctc@ubuntu:~$ ./my.sh ㄱ경로 지정 후 실행가능
/home/kopoctc
total 72
-rw-r--r-- 1 root    root    51810 Dec 19 18:21 aa
-rw-r--r-- 1 root    root      476 Dec 20 20:48 a_out.log
-rwxr-xr-x 1 root    root       61 Dec 20 19:32 a.sh
-rw-r--r-- 1 root    root      290 Jan 17 17:57 boot.log
drwxr-xr-x 3 kopoctc kopoctc  4096 Jan 17 18:07 kopoctc
-rw-r--r-- 1 root    root       16 Jan 12 22:21 mytest.log
/home
kopoctc@ubuntu:~$
```

echo $PATH : 환경변수 출력

<그림 III-5> my.sh파일에 권한과 패스를 부여후 실행

④ 패스

Tip

- 패스(PATH)란 명령어 실행이나 파일명 지정시 생략해도 되는 디렉토리 경로를 의미
- 만일 파일이나 명령어가 디렉토리 명칭이 생략되어진 풀 패스가 아닌 경우, PATH라는 값에 저장되어 있는 디렉토리 명을 순차적으로 찾아서 실행

- 쉘 스크립트 파일내 모든 명령은 절대패스로 명령어 기술 (full path)
- 쉘 스크립트 파일보다 일반명령어 실행이 우선되므로 쉘 스크립트 파일도 패스를 지정해야 함 : ex) ./mysh.sh
- 쉘 스크립트 파일에 환경변수를 인지하도록 하는 방법

ex) export PATH="/home/kopoctc/"

3. 이해하기

#!/bin/sh

- 그림은 쉘 스크립트 파일 내 패스가 인지 안 되는 사례

```
my.sh ㄱ경로 X


~
"my2.sh" 1L, 6C
```

<그림 III-6> 먼저 my.sh를 실행하는 my2.sh을 작성

```
kopoctc@ubuntu:~$ vi my2.sh
kopoctc@ubuntu:~$ sh my2.sh
my2.sh: 1: my2.sh: my.sh: not found
kopoctc@ubuntu:~$
```

<그림 III-7> my2.sh파일내 my.sh을 인식하지 못함

- 쉘 스크립트 파일내 실행명령을 풀 패스로 작성 후 실행

```
/home/kopoctc/my.sh
~ 경로지정시 실행가능 아 export PATH="/home/kopoctc"
"my2.sh" 1L, 20C written
```

<그림 III-8>먼저 my.sh를 풀 패스를 지정하여 my2.sh을 작성

export PATH = $PATH : /home/kopoctc
기존경로에 추가

3. 이해하기

```
kopoctc@ubuntu:~$ vi my2.sh
kopoctc@ubuntu:~$ sh my2.sh
/home/kopoctc
total 72
-rw-r--r-- 1 root    root    51810 Dec 19 18:21 aa
-rw-r--r-- 1 root    root      476 Dec 20 20:48 a_out.log
-rwxr-xr-x 1 root    root       61 Dec 20 19:32 a.sh
-rw-r--r-- 1 root    root      290 Jan 17 17:57 boot.log
drwxr-xr-x 3 kopoctc kopoctc  4096 Jan 17 18:17 kopoctc
-rw-r--r-- 1 root    root       16 Jan 12 22:21 mytest.log
/home
kopoctc@ubuntu:~$
```

<그림 Ⅲ-9> my2.sh실행

- 쉘 스크립트 파일내 패스를 지정하여 실행하는 방법

```
export PATH="/home/kopoctc"
#!/bin/sh
my.sh
~
"my2.sh" 3L, 44C
```

<그림 Ⅲ-10> 먼저 my.sh2파일의 기본실행 디렉토리 /home/kopoctc를 지정

```
kopoctc@ubuntu:~$ vi my2.sh
kopoctc@ubuntu:~$ sh my2.sh
/home/kopoctc
/home/kopoctc/my.sh: 3: /home/kopoctc/my.sh: ls: not found
/home
kopoctc@ubuntu:~$
```

<그림 Ⅲ-11> 실행은 되나 ls명령이 실행 안 됨

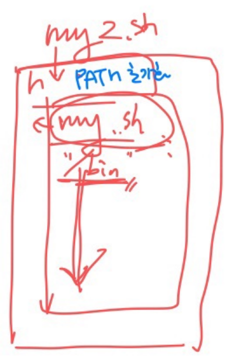
3. 이해하기

```
kopoctc@ubuntu:~$ find / -name ls
...
find: `/proc/1394/ns': Permission denied
find: `/boot/lost+found': Permission denied
/bin/ls
find: `/run/watershed': Permission denied
```

<그림 Ⅲ-12> ls 명령이 어디에 있는지 찾음 /bin/ls

```
export PATH="/bin"
#!/bin/sh
pwd
cd /home
ls -l
pwd
~
"my.sh" 6L, 52C written
```

<그림 Ⅲ-13> 먼저 my.sh파일의 기본실행 디렉토리 /bin을 지정



```
kopoctc@ubuntu:~$ vi my.sh
kopoctc@ubuntu:~$ sh my2.sh
/home/kopoctc
total 72
-rw-r--r-- 1 root    root    51810 Dec 19 18:21 aa
-rw-r--r-- 1 root    root      476 Dec 20 20:48 a_out.log
-rwxr-xr-x 1 root    root       61 Dec 20 19:32 a.sh
-rw-r--r-- 1 root    root      290 Jan 17 17:57 boot.log
drwxr-xr-x 3 kopoctc kopoctc  4096 Jan 17 18:29 kopoctc
-rw-r--r-- 1 root    root       16 Jan 12 22:21 mytest.log
/home
kopoctc@ubuntu:~$
```

<그림 Ⅲ-14> my2.sh실행 성공

3. 이해하기

4) cut

텍스트 파일이나 명령어의 결과 중 필요한 부분을 간단히 추출할 때 cut명령어를 사용한다.

① 옵션 -b, -c 사용사례

- -b 옵션은 바이트(byte), -c는 글자(character)를 의미
- 한글은 2바이트가 1글자로 취급되는 것을 주의

형태	설명	사용 예
N N	자릿수만을 의미	cut -b3
N,M,O	N 자리와 M자리 O자리를 각각 의미	cut -b5,6,7,8,9
N-	N 자리부터 마지막까지를 의미	cut -b10-
N-M	N 자리부터 M 자리까지를 의미	cut -b5-10
-M	처음부터 M 자리까지를 의미	cut -b-10

<표 III-1> cut 옵션 사용 예

```
kopoctc@ubuntu:~$ cat testfile
1234567890abcdefghijk
kopoctc@ubuntu:~$ cat testfile|cut -b8
3
kopoctc@ubuntu:~$ cat testfile|cut -b5,6,7,8,9
56789
kopoctc@ubuntu:~$ cat testfile|cut -b10-
0abcdefghijk
kopoctc@ubuntu:~$ cat testfile|cut -b5-10
567890
kopoctc@ubuntu:~$ cat testfile|cut -b-10
1234567890
kopoctc@ubuntu:~$
```


3. 이해하기

```
kopoctc@ubuntu:~$ ps
  PID TTY          TIME CMD
 1466 pts/0    00:00:00 bash
 1627 pts/0    00:00:00 ps
kopoctc@ubuntu:~$ ps |cut -d ' ' -f3
PID
pts/0
pts/0
pts/0
kopoctc@ubuntu:~$ ps |cut -d ' ' -f2

1466
1638
1639
kopoctc@ubuntu:~$ ps |cut -d ' ' -f4-8
TTY
   00:00:00 bash
   00:00:00 ps
   00:00:00 cut
kopoctc@ubuntu:~$ ps |cut -d ' ' -f8

bash
ps
cut
kopoctc@ubuntu:~$
```

<그림 III-23> cut 명령어 예1