오픈소스소프트웨어

Open-Source Software

ICT융합학부 조용우

전면 처리 및 후면 처리

전면 처리 VS 후면 처리

- 전면 처리
  - 입력된 명령어를 전면에서 실행하고 쉘은 명령어 실행이 끝날 때까지 기다린다.
  - $ 명령어
- 후면 처리
  - 명령어를 후면에서 실행하고 전면에서는 다른 작업을 실행하여 동시에 여러 작업을 수행할 수 있다.
  - $ 명령어 &

후면 작업 확인

$ jobs [%작업번호]

후면에서 실행되고 있는 작업들을 리스트 한다. 작업 번호를 명시하면 해당 작업만 리스트 한다.

```
$ jobs
[1] + Running ( sleep 100; echo done )
[2] - Running find . -name test.c –print
$ jobs %1
[1] + Running ( sleep 100; echo done )
```

후면 작업을 전면 작업으로 전환

$ fg [%작업번호]

작업번호에 해당하는 후면 작업을 전면 작업으로 전환 시킨다.

```
$ (sleep 100; echo DONE) &
[1] 10067
$ fg %1
( sleep 100; echo DONE )
```

입출력 재지정

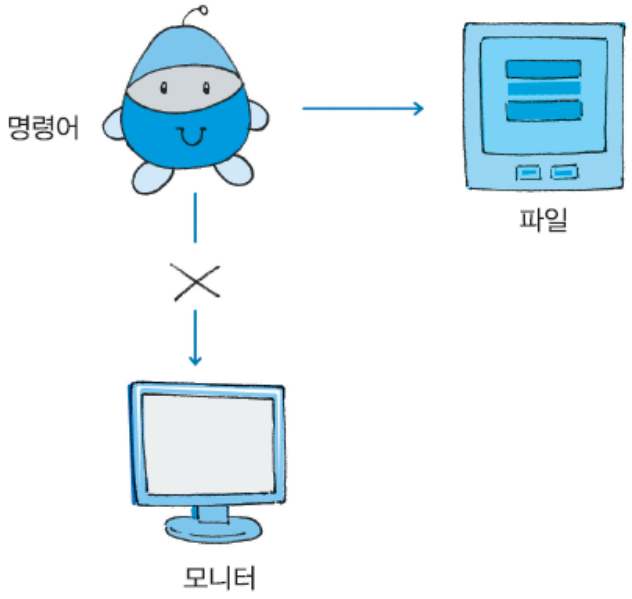
출력 재지정(output redirection)

$ 명령어 > 파일

명령어의 표준출력을 모니터 대신에 파일에 저장한다.

```
$ who > names.txt
$ cat names.txt
```

```
$ cat > list1.txt
  Hi !
  This is the first list.
```



출력 추가(redirecting to append)

$ 명령어 >> 파일

명령어의 표준출력을 모니터 대신에 파일에 추가한다.

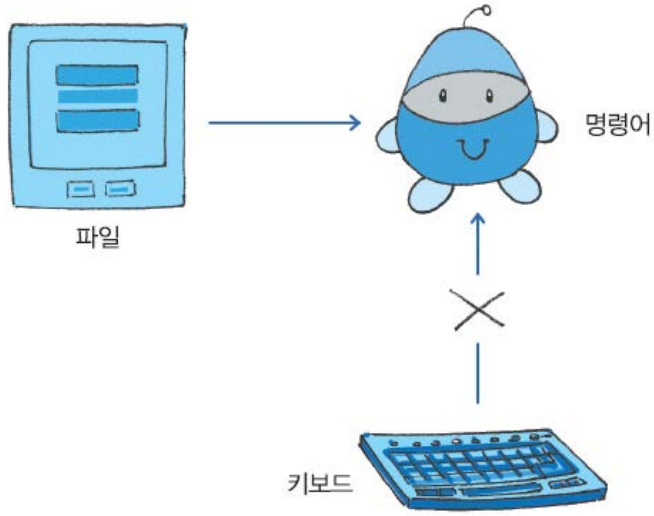
```
$ date >> list1.txt
$ cat list1.txt
Hi !
This is the first list.
Fri Mar 29 18:45:26 KST 2019
```

입력 재지정(input redirection)

$ 명령어 < 파일

명령어의 표준입력을 키보드 대신에 파일에서 받는다.

```
$ wc < list1.txt
3 13 58 list1.txt
```



문서 내 입력(here document)

$ 명령어 << 단어

…

단어

명령어의 표준입력을 키보드 대신에 단어와 단어 사이의 입력 내용으로 받는다.

```
$ wc << END
hello !
word count
END
2    4      20
```

오류 재지정(redirecting the error)

$ 명령어 2> 파일

명령어의 표준 오류를 모니터 대신에 파일에 저장한다.

- 명령어의 실행결과
  - 표준출력(standard output): 정상적인 실행의 출력
  - 표준오류(standard error): 오류 메시지 출력

```
$ ls -l /bin/usr 2> err.txt
$ cat err.txt
ls: cannot access /bin/usr: No such file or directory
```

파이프(pipe)

$ 명령어1 | 명령어2

명령어1의 표준 출력이 파이프를 통해 명령어 2의 표준 입력이 된다.

```
$ who | sort
hci pts/5 3월 29일 13:23 (203.252.201.55)
ywcho pts/3 3월 29일 13:28 (221.139.179.42)
mgkim pts/4 3월 29일 13:35 (203.252.201.51)
```

입출력 재지정 관련 명령어 요약

명령어 사용법	의미
명령어 > 파일	명령어의 표준 출력을 모니터 대신에 파일에 저장한다
명령어 >> 파일	명령어의 표준 출력을 모니터 대신에 파일에 추가한다
명령어 < 파일	명령어의 표준입력을 키보드 대신에 파일에서 받는다
명령어 << 단어	표준입력을 키보드 대신에 단어와 단어 사이의 입력내용으로 받는다
명령어 2> 파일	명령어의 표준오류를 모니터 대신에 파일에 저장한다.
명령어1 | 명령어2	명령어1의 표준출력이 파이프를 통해 명령어2의 표준입력이 된다

컴파일 기본

gcc 컴파일러

$ gcc [-옵션] 파일

C 프로그램을 컴파일 한다. 옵션을 사용하지 않으면 실행파일 a.out을 생성한다 .

```
//hello.c
#include <stdio.h>
Int main() {
    printf("Hello World!\n");
}
```

```
$ gcc hello.c
$ ./a.out
Hello World!
$ gcc –o hello hello.c
$ hello
Hello World!
```

자동 빌드 도구

- make 시스템의 중요성
- 다중 모듈 프로그램을 구성하는 일부 파일이 변경된 경우
  - 변경된 파일만 컴파일하고, 파일들의 의존 관계에 따라서 필요한 파일만 다시 컴파일하여 실행 파일을 만들면 좋다.
- make 시스템
  - 대규모 프로그램의 경우에는 헤더, 소스 파일, 목적 파일, 실행 파일의 모든 관계를 기억하고 체계적으로 관리하는 것이 필요
  - make 시스템을 이용하여 효과적으로 작업

Makefile

$ make [-f makefile]

Make 시스템은 makefile을 이용하여 보통 실행 파일을 빌드한다.

옵션을 사용하여 별도의 메이크파일을 지정할 수 있다.

- 메이크파일
  - 실행 파일을 만들기 위해 필요한 파일들
  - 그들 사이의 의존 관계
  - 만드는 방법을 기술
- make 시스템
  - 메이크파일을 이용하여 파일의 상호 의존 관계를 파악하고 실행 파일을 쉽게 다시 만듦

메이크 파일의 구성

- 메이크파일의 구성 형식

목표(target): 의존리스트(dependencies)

명령리스트(commands)

```
main: main.o test.o
    gcc -o main main.o  test.o
main.o: main.c
    gcc -c main.c
test.o: test.c
    gcc -c test.c
```

메이크 파일의 구성

- make 실행

```
$ make 혹은 $ make main
gcc -c main.c
gcc -c test.c
gcc -o main main.o test.o
```

- test.c 파일이 변경된 후

```
$ make
gcc -c test.c
gcc -o main main.o test.o
```

여러 개의 명령어 사용하기

명령어 열(command sequence)

$ 명령어1; …; 명령어n

나열된 명령어들을 순차적으로 실행한다

```
$ date; pwd; ls
Fri Mar 29 18:08:25 KST 2019
/home/ywcho/linux/test
list1.txt list2.txt list3.txt
```

명령어 그룹(command group)

$ (명령어1; ...; 명령어n)

나열된 명령어들을 하나의 그룹으로 묶어 순차적으로 실행한다

```
$ (date; pwd; ls) > out1.txt
$ cat out1.txt
Fri Mar 29 18:08:25 KST 2019
/home/ywcho/linux/test
list1.txt list2.txt list3.txt
```

조건 명령어 열(conditional command sequence)

$ 명령어1 && 명령어2

명령어 1이 성공적으로 실행되면 명령어 2가 실행되고, 그렇지 않으면 명령어 2가 실행되지 않는다.

```
$ gcc test.c && a.out
```

조건 명령어 열(conditional command sequence)

$ 명령어1 || 명령어2

명령어 1이 실패하면 명령어 2가 실행되고, 그렇지 않으면 명령어 2가 실행되지 않는다.

- `$ gcc test.c || echo compile error`

여러 개 명령어 사용 요약)

명령어 사용법	의미
명령어1; …; 명령어n	나열된 명령어들을 순차적으로 실행한다
(명령어1; …; 명령어n)	나열된 명령어들을 하나의 그룹으로 묶어 순차적으로 실행한다.
명령어1 && 명령어2	명령어1이 성공적으로 실행되면 명령어2가 실행되고, 그렇지 않으면 명령어2가 실행되지 않는다
명령어1 || 명령어2	명령어1이 실패하면 명령어2가 실행되고, 그렇지 않으면 명령어2가 실행되지 않는다

파일 이름 대치와 명령어 대치

파일 이름 대치

- 대표문자를 이용한 파일 이름 대치
  - 대표문자를 이용하여 한 번에 여러 파일들을 나타냄
  - 명령어 실행 전에 대표문자가 나타내는 파일이름들로 먼저 대치하고 실행

대표문자	의미
*	빈 스트링을 포함하여 임의의 스트링을 나타냄
?	임의의 한 문자를 나타냄
[..]	대괄호 사이의 문자 중 하나를 나타내며 부분범위 사용 가능함.

```
$ gcc *.c
$ gcc a.c b.c test.c
$ ls *.txt
$ ls [ac]*
```

명령어 대치(command substitution)

- 명령어를 실행할 때 다른 명령어의 실행 결과를 이용
  - `명령어` 부분은 그 명령어의 실행 결과로 대치된 후에 실행

```
$ echo 현재 시간은 `date`
현재 시간은 Fri Mar 29 18:08:25 KST 2019

$ echo 현재 디렉터리 내의 파일의 개수 : `ls | wc -w`
현재 디렉터리 내의 파일의 개수 : 32
```

따옴표 사용

- 따옴표를 이용하여 대치 기능을 제한

```
$ echo 3 * 4 = 12
3 cat.csh count.csh grade.csh invite.csh menu.csh test.sh = 12
$ echo "3 * 4 = 12"
3 * 4 = 12
$ echo '3 * 4 = 12'
3 * 4 = 12

$ name=나가수
$ echo '내 이름은 $name 현재 시간은 `date`'
내 이름은 $name 현재 시간은 `date`
$ echo "내 이름은 $name 현재 시간은 `date`"
내 이름은 나가수 현재 시간은 2019. 03. 29. (금) 18:27:48 KST
```