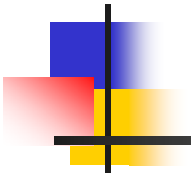
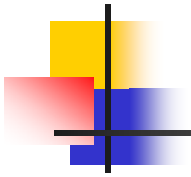


Chapter 4. 유닉스 쉘(Shell)

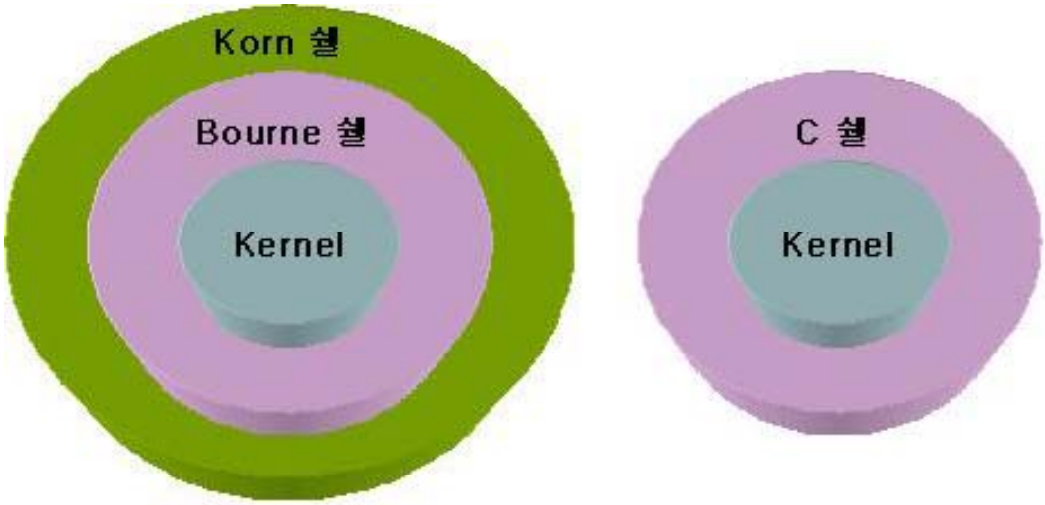


쉘이란?

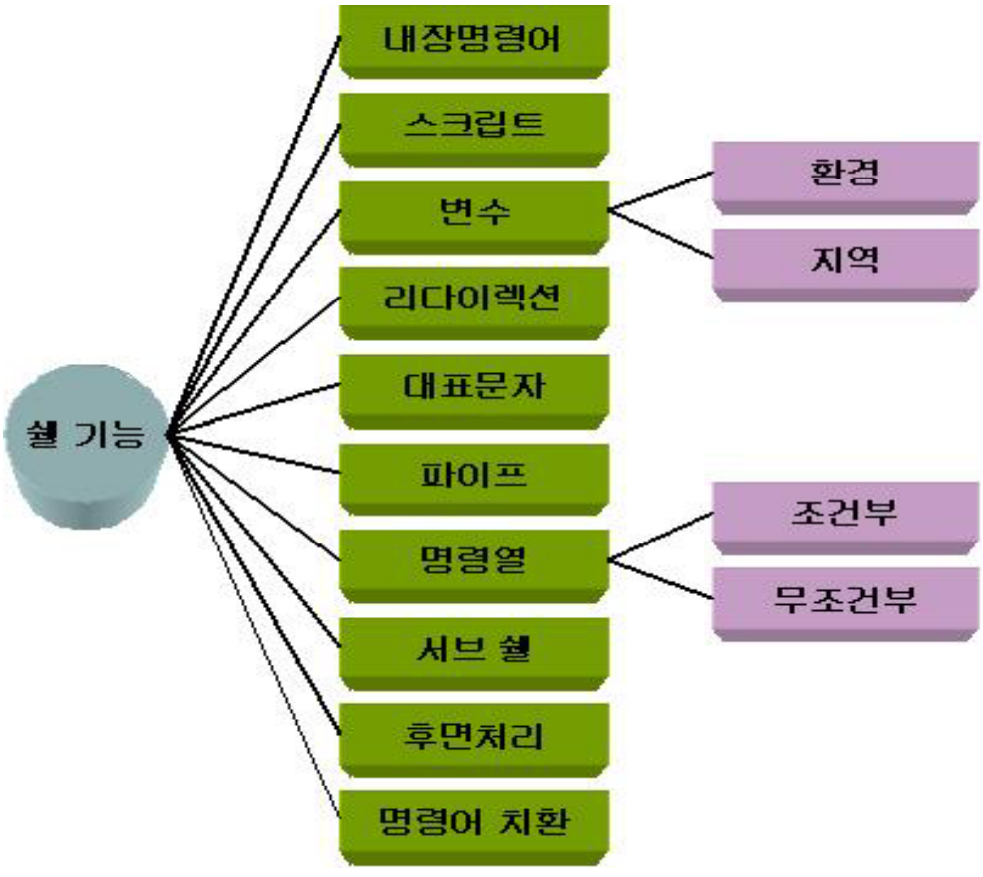
- 쉘(shell)
  - 사용자와 Unix 운영체제 시스템 사이에 위치하여 명령어 해석기(command interpreter)로서 동작하는 프로그램
  - 사용자들의 일반적인 작업을 보다 편리하게 하기 위한 기능을 제공
  - 쉘은 DOS에서의 command.com과 유사
  - 명령어 해석기 역할뿐만 아니라, 프로그래밍 언어이기도 함
  - '스크립트(script)'라는 프로그램을 작성할 수 있음
  - 각 쉘은 고유의 프로그래밍 언어와 규칙을 가지고 있음(6,7장 참고)
  - 사용자에 맞는 쉘을 선택할 수 있음

쉘의 종류

- 현재 사용되고 있는 쉘 프로그램의 종류
  - Bourne shell, Korn shell, C shell
- 이들 쉘은 유닉스 시스템의 사용을 더 편리하게 사용할 수 있게 해주는 공통적인 핵심 연산 부분(커널, Kernel)은 함께 공유



쉘의 종류



쉘의 종류

- 최초의 쉘은 Bourne 쉘(*sh*)
  - 모든 유닉스 시스템은 Bourne 쉘 혹은 그와 호환되는 쉘을 포함
  - 이 쉘은 입/출력을 제어하기 위한 많은 좋은 특징들을 가지고 있으나 사용자와의 대화적인 목적을 위해서는 적합하지 않음

- 이러한 목적을 위해서 나오게 된 쉘이 바로 C 쉘(*csh*)
  - 거의 대부분의 유닉스 시스템에서 이용 가능
  - 이 쉘은 C 언어 형태의 문법을 사용하는데 입/출력에는 오히려 어색하지만 개선된 작업 제어(job control)를 제공

쉘의 종류

표 4-1. 유닉스 쉘의 종류

쉘 명	프로그램 이름
Bash	bash
Bourne shell	sh
C shell	csh
Korn shell	ksh
Tcsh	tcsh
Zsh	zsh

쉘의 선택

- 쉘 선택은 기호, 능력과 호환성에 관한 문제
  - 예를 들어, C 쉘은 대화형 작업에 있어서 Bourne 쉘보다 좋지만 스크립트 프로그래밍(script programming) 관점에서는 다소 나쁘다. Korn 쉘은 Bourne 쉘과 상위 수준으로 호환되게 설계되었으며, Bourne 쉘과 C 쉘의 가장 좋은 특성들을 결합하고 있다.
- 시스템에 로그인하면 미리 정해진 디폴트 쉘이 동작
  - 이것은 시스템 관리자가 사용자의 계정(account)을 부여할 때 미리 정해준 쉘
  - 쉘이 시작되면서 관련된 시동파일(startup files)을 수행. 이 파일에는 환경변수 설정이나 명령의 탐색 경로 등을 설정하는 명령들을 포함 가능
- Bourne 쉘에 대한 디폴트 프롬프트는 $(root 사용자의 경우는 #)
- C 쉘의 디폴트 프롬프트는 %

쉘 변경하기

- 시스템 관리자가 사용자의 계정(account)을 부여할 때 각 사용자의 디폴트 쉘 지정 가능(로그인 쉘)
  - 사용자가 자신의 쉘이 어떤 쉘인가를 알 수 있는 방법은 처음 로그인 했을 때 나타나는 프롬프트를 보면 알 수 있다. 프롬프트가 $이면 Bourne 쉘이나 Korn 쉘일 것이다. 프롬프트가 %이면 C 쉘이다.
- 쉘 변경
  - Bourne 쉘을 사용하고 싶으면 프롬프트 상에서 'sh'를 입력
  - C 쉘을 사용하고 싶으면 프롬프트 상에서 'csh'를 입력
  - 새로운 쉘을 빠져나오기 위해서는 'exit' 명령어를 이용
  - 로그아웃을 하기 위해서는 반드시 로그인 쉘을 통해서만 가능

쉘 변경하기

```
SunOS 5.8

login: redfox
Password:
Last login: Wed Jan  4 16:46:34 from 202.31.xxx.xxx
Sun Microsystems Inc.   SunOS 5.8       Generic Patch   October 2001
cecom% sh  ← Bourne 쉘 수행
$ csh   ← C 쉘 수행
cecom% ps
  PID TTY      TIME CMD
22801 pts/2    0:00 csh
22799 pts/2    0:00 sh
22793 pts/2    0:00 csh
cecom% exit   ← 현재 쉘인 C 쉘 종료
$ exit   ← 현재 쉘인 Bourne 쉘 종료
cecom% logout   ← 로그인 쉘에서 유닉스 시스템 종료

호스트에 대한 연결을 잃었습니다.
```

그림 4-3. 새로운 쉘 수행 및 종료방법

until here

쉘 운영

- 사용자가 쉘을 호출하면 쉘은 다음과 같이 미리 정해진 순서대로 작업을 진행
  1. 몇 가지 초기화 정보를 포함하고 있는 특정한 시동 파일을 읽어 들여 수행. 이 파일은 일반적으로 사용자의 홈 디렉터리에 있다. 시동 파일은 각 쉘마다 약간씩 다르다.
  2. 프롬프트를 표시하고 사용자의 입력 대기
  3. 만일 사용자가 명령 줄에서 ^D(Control-D)를 입력하면, 쉘은 이를 입력의 끝으로 해석하여 쉘을 종료. 그렇지 않고 사용자가 어떤 명령을 입력하였다면 쉘은 그 명령을 수행하고 다시 2단계로 되돌아감
- 만일 터미널에서 사용자가 명령을 입력할 때 한 줄에 모두 입력할 수 없는 경우에는 명령 줄 끝에 역슬래시(₩)를 붙이고 다음 줄에 계속해서 입력

명령어 치환

- 명령 행이나 쉘 스크립트의 역 인용부호 '`'로 둘러싸인 명령은 실행이 되어 표준 출력됨

※주의!
Tab키 위에 있는, ESC키 밑에 있는 ‘이다.

```
cecom% echo 오늘은 `date` 입니다

오늘은 2006년 1월 10일 화요일 오후 07시 27분 16초 입니다
cecom% who
redfox     pts/3        1월 10 18:58   (202.31.201.134)
cecom% echo 현 시스템에 접속한 사람은 `who | wc -l` 명입니다

현 시스템에 접속한 사람은 1 명입니다
```

그림 4-4. 역 인용부호(“ ` ”) 사용 예

명령 열(sequences)

- 여러 개의 명령을 한번에 지정하고자 하는 사용자에게 유용한 기능으로 일련의 명령들을 세미콜론(;)으로 분리하여 입력

```
% date; pwd; ls
2000년 11월  6일 월요일 오후 12시 17분 59초
/home1/gildong/unix2
curriculum      files.long      logs.beauty     od.example
todays.logins
cut.exam        gzip.desc       logs.bottom     phone           users
file            listing         logs.photon     pmchoe.txt      users.a
file.new        logfile.tar.Z   logs.top        temp.file       users.b
%
```

- 또한 명령열의 각 명령은 개별적으로 입출력 리다이렉션이 가능

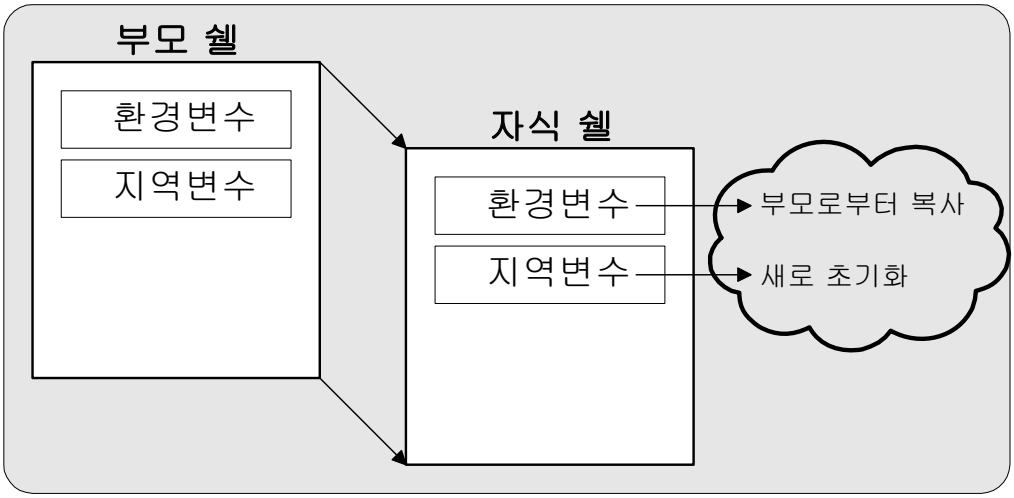
```
% date > date.txt; pwd > pwd.txt; ls
curriculum      file.new        logfile.tar.Z   logs.top        pwd.txt
cut.exam        files.long      logs.beauty     od.example      temp.file
date.txt        gzip.desc       logs.bottom     phone           todays.logins
file            listing         logs.photon     pmchoe.txt      users
%
```


서브 쉘(subshell)(1/2)

- 사용자가 로그인할 때 유닉스는 로그인 쉘을 제공
- 기본적인 단순한 명령들은 로그인 쉘에 의해 수행
- 그러나 현재의 쉘(부모 쉘)이 어떤 작업을 수행하기 위해서는 다음과 같은 여러 환경에서 자식 쉘(서브 쉘이라고도 함)을 생성하여 수행
  - (ls; pwd; date)의 형식과 같이 한 그룹의 명령이 수행될 때는 부모 쉘에서 이 명령을 수행할 자식 쉘을 생성. 만일 명령이 후면(background)에서 처리되지 않는다면 부모 쉘은 자식 쉘이 그 명령을 종료할 때까지 대기
  - 쉘 스크립트(6, 7장 참조)의 경우 스크립트 내의 명령들을 수행하기 위해 부모 쉘은 자식 쉘을 생성. 만일 명령이 후면에서 처리되지 않는다면 부모 쉘은 자식 쉘이 그 명령을 종료할 때까지 대기
  - 후면 작업이 수행될 때 부모 쉘은 후면에서 명령을 수행할 자식 쉘을 생성. 이 경우 부모 쉘은 자식 쉘의 종료를 기다리지 않고 자식 쉘과 병행하여 수행

서브 쉘(2/2)

```
% pwd                ... 로그인 쉘의 현재 작업 디렉토리 표시
/home1/gildong/unix2
% (cd /; pwd)       ... 서브 쉘 이동후 pwd 수행
/                   ... 서브 쉘로부터 출력
% pwd                ... 로그인 쉘은 이동되지 않음
/home1/gildong/unix2
%
```



후면처리(background processing)

- 명령 줄에서 명령의 끝에 '&' 메타문자를 붙이면 그 명령은 후면에서 수행
  - 어떤 명령이 후면에서 처리된다는 것은 그 작업이 서브 쉘에서 수행됨을 의미하며, 부모 쉘과 병행하여 수행
- 후면에서 수행되고 있는 프로세스가 키보드를 차지하지 않음
- 시간이 오래 걸리는 어떤 작업을 후면에서 처리되도록 해놓고 부모 쉘에서 다른 작업을 수행한다면 효율적으로 작업을 수행 가능

```
cecom% find / -name "a.c" -print &
cecom% pwd
/home1/gildong/unix2
cecom% date
2000년 11월  6일 월요일 오후 01시 19분 04초
```

후면처리(background processing)

```
cecom% find / -name "*.txt" -print   ← 전면에서 find 수행
find: 디렉토리 /lost+found을(를) 읽을 수 없습니다: 사용 권한이 거부됨
find: 디렉토리 /usr/lost+found을(를) 읽을 수 없습니다: 사용 권한이 거부됨
<많은 find 결과가 터미널에 나타남>
<이하 생략...>
cecom% find / -name "*.txt" -print &   ← 후면에서 find 수행
[1] 20179   ← 후면 프로세스의 id 번호
find: 디렉토리 /lost+found을(를) 읽을 수 없습니다: 사용 권한이 거부됨
find: 디렉토리 /usr/lost+found을(를) 읽을 수 없습니다: 사용 권한이 거부됨
cecom% date   ← 전면에서 다른 작업을 수행할 수 있음
2006년 1월 10일 화요일 오후 08시 24분 05초
/usr/local/lib/perl5/5.8.5/unicore/PropValueAliases.txt   ← 후면 find의 결과
/usr/local/lib/perl5/5.8.5/unicore/PropertyAliases.txt
cecom% ls
TTT      Test3    Unix3    date.txt  file3    file5    out.txt
Test2    Unix2    Unix_a   fff       file4    ls.txt   pwd.txt
cecom% date & pwd &
[1] 20182
[2] 20183
cecom% /home/redfox
2006년 1월 10일 화요일 오후 08시 29분 19초
```

그림 4-8. 후면처리 사용 예

특수문자

- 쉘은 사용자의 명령을 실행하기 전에 먼저 사용자가 무엇을 타이프 했는지를 분석
- 만일 특수문자가 발견되면 해당 특수문자를 다른 것으로 치환하는 특수연산을 수행함

특수 문자	의 미
공백문자	공백문자는 인자들을 구분 짓기 위해 사용, 여러 개의 공백문자는 무시
newline character	명령행의 끝을 표시하기 위해 사용
' " ₩	쉘이 특수문자를 해석하는 방식을 변경하는 특수한 인용문자들
&	명령어 다음 위치에 사용되며, 후면(background)에서 명령을 실행
< > >> << ` |	입출력 방향전환 문자들
* ? [ ] [^	파일명 치환 문자들
$	쉘 변수 의미 (예: $shell)
;	한 줄에서 여려 개의 명령을 구분하는데 사용

공백문자(white space)

- 쉘이 명령의 문자를 검사할 때 공백 문자에 의해 명령 자체와 인자(arguments)들을 서로 구분 짓는데, 중복된 공백 문자는 무시됨
- 명령이 실행될 때 먼저 인자의 목록이 제공되고 나서 명령의 기능이 수행됨
- echo 명령의 경우 인자들의 각각을 단일 공백 문자로 분리시켜 화면에 출력함

인용 문자(quote characters)

- 치환, 변수 치환, 명령 치환 기법을 금지하고자 할때 사용
- 첫 번째 인용문자(')를 만나면 쉘은 두 번째 인용문자(')까지의 내용에서 특수문자의 의미를 무시
- 특수문자인 여러 개의 공백 문자의 의미를 무시한 것
- echo 명령은 4개의 분리된 인자가 아니라 하나의 인자를 받아 들여 화면에 출력함

인용문자	명 칭	의 미
‘	단일 인용 문자 (작은 따옴표)	쉘은 단일 인용 문자로 둘러싸인 내용에서 모든 특수 문자를 무시한다
“	이중 인용 문자 (큰 따옴표)	쉘은 이중 인용 문자로 둘러싸인 내용 중 $ ‘ ₩ 문자를 제외한 모든 특수 문자를 무시한다
₩	역슬래시	쉘은 역슬래시 바로 다음에 오는 특수 문자를 무시한다

인용 문자(quote characters)

```
cecom% echo 'echo     my name is cho'
echo     my name is cho
cecom% echo *
TTT Test2 Test3 Unix2 Unix3 Unix_a date.txt fff file3 file4 file5 ls.txt out.txt
pwd.txt
cecom% echo '*'
*
cecom% echo ₩*
*
cecom% echo I'm Cho
맞지 않음 '
cecom% echo I₩'m Cho
I'm Cho
cecom% echo 'my name is $USER and the date is `date`'
my name is $USER and the date is `date`
cecom% echo "my name is $USER and the date is  ` date ` "
my name is redfox and the date is 2006년 1월 10일 화요일 오후 09시 25분 50초
cecom%
```

그림 4-12. 인용문자 사용 예

파일명 치환

- 파일명을 표현할 때 몇몇 특수 문자들인 *, ?, [], [^]를 사용할 수 있도록 해주는 기능
- 쉘은 이러한 특수 문자들을 인식하여 그들을 문자의 조합과 매치되는 파일명으로 치환하게 함
- 유닉스 와일드 카드

인용문자	의 미
*	0개 혹은 하나 이상의 문자와 매치
?	한 문자와 매치
[]	[와]사이의 문자 중 한 문자와 매치
[^]	^ 다음에 포함되지 않은 임의의 한 문자와 매치

파일명 치환

표 4-4. 파일명 치환 사용 예

사용 예	의 미
ls *	*은 현재 디렉토리의 모든 파일과 디렉토리의 이름으로 치환된다.
ls *.txt	.txt로 끝나는 모든 파일 이름으로 치환된다.
ls ?.txt	.txt 앞에 하나의 문자가 있는 파일 이름으로 치환된다.
ls [ap]*	a 또는 p로 시작하는 파일 이름으로 치환된다.
ls [A-Za-z]*	영문자로 시작하는 모든 파일 이름으로 치환된다.
ls Unix*/*.txt	Unix* 디렉토리에서 .txt로 끝나는 파일 이름으로 치환된다.
ls */*	서브 디렉토리의 모든 파일 명으로 치환된다.

파일명 치환

```
cecom% ls *
a.txt    date.txt  fff      file3    file4    file5    out.txt   pwd.txt
Test2:
file1
Unix3:
out.txt   <이하생략...>
cecom% ls *.txt
a.txt    date.txt  out.txt   pwd.txt
cecom% ls ?.txt
a.txt
cecom% ls [ap]*
a.txt    pwd.txt
cecom% ls [A-Za-z]*
a.txt    date.txt  fff      file3    file4    file5    out.txt   pwd.txt
Test2:
file1   <이하생략...>
cecom% ls Unix*/*.txt
Unix3/out.txt
cecom% ls */*
Test2/file1     Test3/file1     Unix2/file2     Unix3/out.txt
Unix_a/Unix_aa:
```

그림 4-13. 파일명 치환 사용 예

표준 입출력

- 유닉스가 운영체제에 미친 중요한 요소들
  - 공통된 작업을 수행하거나 원하는 정보를 획득하기 위한 많은 유틸리티를 제공해준다는 점
  - 또 다른 특징으로는 데이터가 유닉스 시스템에 저장되거나 전송되는 표준방식
    - 이 방식은 데이터를 파일, 터미널 스크린, 혹은 프로그램에 전송되도록 할 수 있고, 역으로 파일, 키보드, 혹은 프로그램으로부터 데이터를 받아올 수 있다.

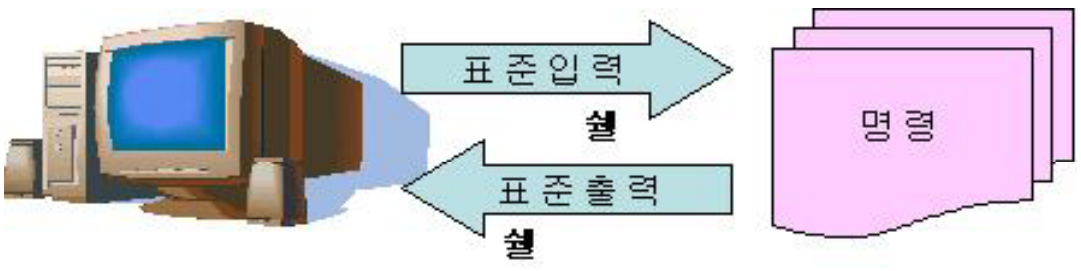
- 이와 같이 데이터를 관리하는 유닉스의 표준방식은 유닉스 유틸리티에 두 가지 중요한 특징을 제공
  - 입출력 리다이렉션(I/O 방향전환)
  - 파이프(pipe).

표준 입출력

- 일반적으로 세 가지 표준 파일

stdin(0)	프로그램에 표준 입력
stdout(1)	프로그램으로부터의 표준 출력
stderr(2)	프로그램으로부터 표준 오류 출력

- 입력은 정상적으로는 키보드 혹은 파일로부터 받아들임
- 출력(stdout과 stdin 모두)은 정상적으로 터미널로 보내지지만 이들 둘 다 하나 이상의 파일로 방향전환 될 수 있음



입출력 리다이렉션

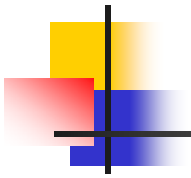
- 쉘의 리다이렉션 기능
  - 프로세스의 출력을 파일에 저장하게 해주는 출력 리다이렉션
  - 파일의 내용을 프로세스의 입력으로 사용할 수 있게 해주는 입력 리다이렉션

- 출력 리다이렉션
  - 프로세스의 출력을 파일에 저장하게 해줌으로써 나중에 그 파일을 표시하고, 프린트하고, 편집하고, 다음 프로세스의 입력으로 사용할 수 있게 해주는 편리한 기능
  - 출력 리다이렉션을 위해 >나 >> 메타문자를 사용
  - 쉘은 현재의 디렉터리에 "redfox.txt"라는 파일이 없으면 이를 새로 생성하고, 있으면 이전의 내용을 현재의 것으로 덮어씌움
  - 데이터 입력이 끝났을 때 ^D 키를 누름

입출력 리다이렉션

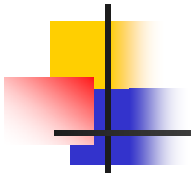
- 출력 리다이렉션
  - 명령의 출력을 터미널 스크린에 보내지 않고 파일로 보내 줌
  - 명령의 출력을 취하여 그 결과를 터미널에 보내지 않고 파일로 저장

```
cecom% cat > file1
this is file1
^D
cecom% cat > file2
this is file2
^D
cecom% cat file1 >> file2    ← 기존 file2에 덧붙임
cecom% cat file2
this is file2
this is file1
```



입출력 리다이렉션

- 출력 리다이렉션 사용 예
  - cecom% cat file1 file2 > file3
    - file1과 file2를 병합하여 그 출력을 file3으로 함
    - 만일 file3이 존재하지 않으면 새로 생성
    - 만일 파일이 존재하면 새로운 내용이 삽입되기 전에 기존의 파일 내용이 삽입되거나, 혹은 *csh*의 noclobber 옵션이 설정되어 있다면 그 명령은 취소
    - 기존의 파일들(file1과 file2)은 변경되지 않음
  - cecom% cat file1 >> file2
    - 위의 명령은 file1의 내용을 기존의 file2에 추가. 겹쳐 쓰는 것이 아님
    - 입력은 다음과 같은 형태로 파일로부터 방향전환



입출력 리다이렉션

- 입력 리다이렉션
  - 프로세스의 입력을 파일로부터 받아들일 수 있게 하는 방법으로 입력 내용을 미리 준비할 수 있게 해주기 때문에 편리
  - 입력 방향전환을 위해 <나 << 메타문자를 사용
  - 명령의 입력을 키보드로부터 받아들이지 않고 파일로부터 받아들이도록 해 줌
  - 파일을 명령의 입력으로 읽어 들임
  - 데이터 입력이 끝났을 때 ^D 키를 누름

입출력 리다이렉션

- 입력 리다이렉션 사용 예
- 1. cecom% command < fileName
  - fileName 파일 내용을 표준 입력으로 사용하여 command를 실행
  - fileName이 존재하지 않거나 읽기 허용을 갖지 않으면 에러 발생
- 2. cecom% command << word
  - word 로 시작하는 줄의 앞줄까지의 내용을 임시 파일로 복사하고, 그 임시 파일의 내용을 표준 입력으로 하여 command를 실행
- 이 외에도 여러 가지 입·출력 방향전환을 혼합하는 방법이 가능
  - cecom% cat < file3 > file4
  - cecom% cat << finished > input

입출력 리다이렉션

표 4-6. 입출력 리다이렉션

기 호	의 미
>	출력 리다이렉션
>!	출력 리다이렉션, csh의 noclobber 옵션을 중복 정의
>>	기존 파일에 출력을 추가
>>!	기존의 파일에 출력을 추가, csh의 noclobber 옵션을 중복 정의하고 파일이 존재하지 않으면 파일 생성
|	다른 명령으로 파이프 출력
<	입력 리다이렉션
<<word	word로 시작하는 줄의 앞줄까지 표준 입력으로 받아들임
>&	표준 출력과 표준 에러를 파일로 리다이렉션 한다.
>>&	표준 출력과 표준 에러를 파일에 추가한다.

- stdout과 stdin을 두 개의 독립된 파일로 방향전환하고자 한다면 사용자는 서브쉘에서 다음처럼 먼저 stdout을 방향전환해야 함
  - (command > out_file) >& err_file

입출력 리다이렉션

```
cecom% cat  < file3 > file4    ②
cecom% cat  file4
this is file1
this is file2
cecom% cat  << finished  >  input    ③
my name is cho!!!
finished
cecom% (find / -name "*.txt" -print > out_file) >& err_file &    ④
[1] 20275
cecom% pwd
/home/redfox
cecom% cat  out_file
/usr/local/share/vim/vim63/doc/usr_01.txt
/usr/local/share/vim/vim63/doc/usr_02.txt
<이하생략...>
cecom% cat  err_file
find: 디렉토리 /var/spool/lp/tmp을(를) 읽을 수 없습니다: 사용 권한이 거부됨
find: 디렉토리 /var/dt/sdtlogin을(를) 읽을 수 없습니다: 사용 권한이 거부됨
<이하생략...>
```

④ : find 검색 결과값은 out_file로, 에러값들은 err_file로 출력리다이렉션

연습

- 1. cat 명령을 사용하여 자기소개를 위한 profile.txt파일을 작성하라
- 2. cat명령을 사용하되 profile.txt파일을 입력으로 사용하고 profile2.txt파일로 저장하라.
- 3. wc명령을 사용하여 profile.txt파일에 포함된 라인수,워드수,문자수를 각각 구하되 그 결과를 wc.txt파일로 저장하라.

파이프(pipe)

- 쉘
  - 한 프로세스의 표준출력을 다른 프로세스의 표준입력으로 사용할 수 있도록 해줌
  - 이것은 다음과 같이 파이프(|) 메타문자를 사용하여 프로세스들을 서로 연결하면 된다.
- 명령1 [인수] | 명령2 [인수]
  - 명령1의 표준출력을 명령2의 표준입력으로 받아들임
  - 이와 같이 하여 여러 개의 명령들을 파이프로 연결
  - 이러한 방식으로 연결된 명령들의 순서를 파이프라인(pipeline)이라고 함
  - 파이프라인은 유닉스 개발의 기본 철학중의 하나로 하나의 커다란 문제를 작고, 재사용 가능한 여러 프로세스의 연결에 의해 해결할 수 있게 해주는 기능

파이프(pipe)

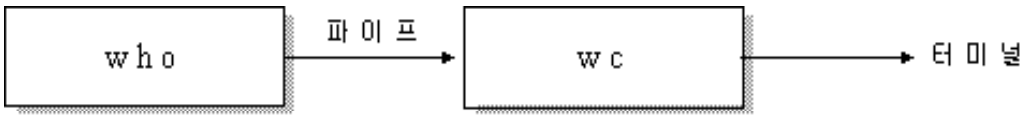


그림 4-17. 파이프라인

- 파이프는 한 명령의 표준 출력을 하나의 파일로 리다리렉션한 후 그 파일을 또 다른 명령의 표준 입력으로 사용하는 것과 같은 효과
- 별도의 명령어나 중간 파일을 사용하지 않음

파이프

```
[ce:/home1/gildong]% who | sort
2end        pts/2         11월 20 09:52
(192.9.200.122)
gildong     pts/4         11월 20 08:16
(brain.ssu.ac.kr)
iljo        pts/5         11월 20 09:53
(192.9.200.128)
s002023     pts/8         11월 20 10:16
(192.168.20.121)
s961002     pts/6         11월 20 10:10
(192.9.200.103)
s961009     pts/12        11월 14 13:13
(192.9.200.125)
s991052     pts/38        11월 14 15:07   (ce35)
[ce:/home1/gildong]%
```


파이프

```
cecom% ls -al > tempfile
cecom% wc -l < tempfile
    27
cecom% ls -al | wc -l
    27
cecom% who | wc -l
     1
cecom% ls | sort
TTT
Test2
<중략...>
a.txt
err_file
```

그림 4-19. 파이프 사용 예

파이프

- tee 명령어
  - 파이프의 결과를 파일에 복사하게 하고, 계속해서 파이프라인을 따라 흐르게 함
  - 하나 이상의 파일 이름을 지정하여 여러 개의 복사본을 만들 수 있음
  - 명령어 구조 : tee -ia 파일명
    - -i : 입터럽트 무시
    - -a : 입력을 파일에 덮어씌우지 않고 대신에 그 파일의 끝에 덧붙임

파이프

```
cecom% who | tee who.capture | sort
redfox     pts/2        1월 11 17:00   (202.31.xxx.xxx)
cecom% cat who.capture
redfox     pts/2        1월 11 17:00   (202.31.xxx.xxx)
cecom% ps -ef | grep redfox | tee t1 t2 | sort    ①
 redfox 20194 20192  0 17:00:08 pts/2    0:00 -csh
 redfox 20312 20194  0 20:42:57 pts/2    0:00 grep redfox
 redfox 20313 20194  0 20:42:57 pts/2    0:00 tee t1 t2
 redfox 20314 20194  0 20:42:57 pts/2    0:00 sort
cecom% cat t1
 redfox 20312 20194  0 20:42:57 pts/2    0:00 grep redfox
 redfox 20314 20194  0 20:42:57 pts/2    0:00 sort
<중략…>
cecom% cat t2
 redfox 20312 20194  0 20:42:57 pts/2    0:00 grep redfox
 redfox 20314 20194  0 20:42:57 pts/2    0:00 sort
<중략 …>
```

변수

- 쉘 변수: 지역변수와 환경변수
- 사용자는 환경변수와 지역변수 모두 사용 가능
  - 한 쉘이 다른 쉘(자식 쉘)을 호출할 때 자식 쉘은 그 부모 쉘의 환경변수 복사본은 얻을 수 있지만 지역변수의 경우는 그 복사본을 얻을 수 없다는 것이 차이점
- 환경변수
  - 환경변수는 부모 쉘과 그 자식 쉘 사이에 유용한 정보를 전달하는 데 사용. 모든 쉘은 각 쉘과 관련된 시동파일에 의해 초기화된 환경변수 집합을 가짐.
  - 환경변수는 사용자의 로그인 쉘에 의해 설정되며, 쉘은 부모(parent) 쉘의 환경을 그대로 상속
- 지역변수
  - 오로지 자신의 쉘에 의해서만 사용되며 다른 프로세스에 의해 사용 불가. 즉 자식(child) 프로세스는 지역변수를 부모 프로세스에 전달 불가
  - 모든 쉘은 특별한 의미를 가지는 미리 정의된 지역변수 집합도 가짐.

변수(미리 정의된 환경변수)

DISPLAY	사용할 그래픽 디스플레이, 예: brain:0.0
EDITOR	디폴트 편집기에 대한 패스, 예: /usr/bin/vi
GROUP	사용자의 로그인 그룹, 예: staff
HOME	사용자의 홈 디렉터리 패스, 예: /home/gildong
HOST	시스템의 호스크명, 예: brain
IFS	내부 필드 구분자
LOGNAME	로그인 명
PATH	명령어들을 탐색할 패스, 예: /usr/bin:usr/ucb: /usr/local/bin
PS1	기본 프롬프트 문자열(Bourne 쉘의 경우만), 예: $
PS2	2번째 프롬프트 문자열(Bourne 쉘의 경우만), 예: >
SHELL	사용중인 로그인 쉘, 예: /usr/bin/csh
TERM	터미널 형태, 예: xterm
USER	사용자명, 예: gildong

변수

- 대부분 환경변수들은 사용자가 로그인할 때 자동으로 설정
- 환경변수의 설정 내용을 변경하려면 시동파일(.profile 혹은 .login)에서 항목을 수정하거나 쉘 내에서 언제든지 변경 가능
- PATH나 DISPLAY 변수 등이 사용자의 환경을 맞추기 위해 자주 수정하게 되는 것들
- PATH 변수는 사용자가 지정한 명령을 자동으로 찾을 수 있도록 디렉터리를 지정
  - 이것에 대한 예제는 다음 장의 시동 파일에서 찾아보면 된다.

변수

- 환경변수 설정, 확인, 해제 방법(5장 참고)
  - C shell의 경우 다음과 같은 명령으로 환경변수를 설정
  - *env* 명령으로 설정된 변수 목록 보기 가능
  - *unsetenv*(C 쉘)나 *unset*(Bourne 쉘)으로 환경변수의 설정을 해제 가능

```
%setenv NAME value
%echo $NAME
value
%unsetenv NAME
%echo $NAME
NAME: 정의되지 않은 변수
%
```

  - Bourne 쉘의 경우는 다음과 같다.

```
$NAME=value;export NAME
```

  - 설정된 변수의 현재 값은 '$변수명' 혹은 '${변수명}'으로 확인할 수 있음.

변수

```
cecom% env
HOME=/home/redfox
PATH=/usr/bin:
<중략…>
SHELL=/bin/csh
MAIL=/var/mail/redfox
LANG=ko
<중략…>
PWD=/home/redfox
USER=redfox
cecom% echo $USER
redfox
cecom% echo $TERM
ansi
cecom% echo $PATH
/usr/bin:
cecom% echo $HOME
/home/redfox
cecom% setenv NAME shcho
cecom% echo $NAME
shcho
cecom% unsetenv NAME
cecom% echo $NAME
NAME: 정의되지 않은 변수
```

그림 4-20. 환경변수 확인, 설정, 해제 방법

작업 제어

- 다중 작업(multitasking)은 유닉스의 좋은 특징 중 하나
- 현재 프로세스들의 목록을 얻고 그 프로세스들의 동작을 제어하는 것이 중요함
- 다음과 같은 유틸리티와 내장 명령어가 있음(자세한 내용은 해당 쉘 부분에서 언급)
  - bg는 전면에서 수행중인 명령어를 후면에서 수행
  - fg는 후면에서 수행중인 명령어를 전면에서 수행
  - jobs는 후면 처리되고 있는 작업들의 목록을 보여줌
  - kill은 후면 처리 중인 작업을 중지시킴
  - ps는 수행중인 프로세스 목록을 보여줌

작업 제어

- 쉘(shell)에서 명령 줄에 "&"를 추가시킴으로써 언제든지 작업을 후면에서 수행 가능
  - 이것은 새로운 버전의 Bourne 쉘(*sh*)에서도 마찬가지이다.
  - [ce:/home1/gildong]% find / -name "*.txt" -print &
- 이미 수행되고 있는 명령이라 하더라도 그 작업을 멈추기 위해 사용자가 ^Z(Control-Z)를 친 후 "*bg*"라고 입력하면 역시 같은 효과

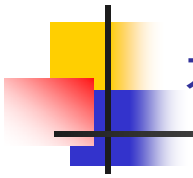
```
[ce:/home1/gildong]% find / -name "*.txt" -print
find: 디렉터리 /lost+found를 읽을수 없음: 권한이 거부됨
find: 디렉터리 /usr/lost+found를 읽을수 없음: 권한이 거부됨
^Z
중단됨 (사용자)
[ce:/home1/gildong]% bg
[1]   + find / -name *.txt -print &
```


작업 제어

- 위의 작업을 다시 전면(foreground)에서 수행하고자 한다면 ^Z(Control-Z)를 친 후 "*fg*"라고 입력
- 사용자는 후면에서 실행되고 있는 많은 작업을 가질 수 있음
  - 작업이 후면에서 실행되고 있는 동안에는 그 작업은 더 이상 입력을 위해 키보드와 연결되어 있지 않다.
  - 그러나 터미널과는 계속 연결되어 있기 때문에 작업 결과들이 수시로 터미널에 보여질 수 있다. 이와 같이 작업 결과 혹은 처리 과정에 대한 메시지가 흩어져서 출력되기 때문에 이를 파일로 리다이렉션하는 것이 좋다.

```
[ce:/home1/gildon]% find / -name "*.txt" -print > result &
```

- 이와 같이 명령을 주면 *find* 명령에 의해 수행된 결과가 result 파일로 생성
  - 이것은 나중에 편집기로 살펴볼 수 있다.



작업 제어

- 내장된 명령인 *jobs*는 후면처리되고 있는 작업들의 목록을 보여 줌

```
[ce:/home1/gildong]% jobs
[1]  + Running  find / -name *.txt -print
```

- 사용자는 이를 kill 명령으로 후면처리 작업을 중지 가능
  - 위에서 [1]은 첫 번째 후면처리 작업임을 의미
  - 이 번호와 함께 %n 표기법을 이용하여 그 작업을 중지 가능

```
[ce:/home1/gildong]% kill %1
[ce:/home1/gildong]% jobs
[1] 종료됨(Terminated) find / -name *.txt -print > result
```