메타문자 & Shell 프로그래밍

리다이렉션(Redirection) / 파이프라인(Pipeline)

리다이렉션(Redirection) 출력을 파일이나 스트림으로 전달하는데 사용한다.
파이프라인(Pipeline) 출력을 다른 프로그램 이나 유틸리티 로 넘길 때 사용

스트림(Streams)

모든 애플리케이션에는 외부 세계와 연결하는 세 가지 고유한 스트림(Streams)이 있다.
표준 입력(stdin), 표준 출력(stdout), 표준 에러(stderr)
세 가지 표준 스트림 모두 POSIX 사양에 정의되고 C 언어에서 사용되는 ID 값을 갖습니다.
표준 입력은 0, 표준 출력은 1, 표준 에러는 2
때로는 파일 대신 스트림으로 리다이렉션 하고싶으면, 파일 이름 대신 &가 접두사로 붙은 스트림ID 를 사용하면 된다.
>&1 : 표준 출력으로 리다이렉션
>&2 : 표준 에러로 리다이렉션
2>&1 : 표준 에러 스트림을 표준 출력 스트림으로 리다이렉션 (한 스트림에서 다른 스트림으로 리다이렉션 하는 것도 가능)

여러 스트림 리다이렉션

표준 출력과 표준 에러를 동시에 리다이렉션할 수 있다.
동일한 명령의 두 개의 다른 리다이렉션을 결합하기만 하면 된다.

```
$ ls -1 > stdout.log 2> stderr.log
$ cd /home/mydir > file1 2> file2
```

위 예시에서 두 스트림은 모두 개별적으로 리다이렉션 되고, 결과 파일도 각각 만들어진다.
표준 출력과 표준 에러를 동일한 파일로 리다이렉션 하려면 위에서 본것처럼 &>를 사용하거나 스트림 결합 연산자를 사용하면 된다.

```
$ cd /home/myFolder &> ./stdout.log
$ cd /home/myFolder > ./stdout.log 2>&1
```

메타문자(File Matching Metacharacters) 및 특수기호 정리

기호	의미
<, 0<, 0<<	• 0< : (Input Redirection) 입력 리다이렉션, 0은 생략가능 표준 입력을 키보드가 아닌 파일의 내용으로 받음. • << : (Here Document) 명령에 직접 여러 줄의 입력을 제공하는 데 사용된다. • <<< : (Here String) 명령에 한 줄의 입력을 제공하는 데 사용된다.
>, 1>, 1>>	• 1> : 출력 리다이렉션, 1은 생략가능 출력을 파일로 리다이렉션, 파일이 없으면 새 파일을 만들고, 있으면 기존 파일을 덮어쓰는 데 사용된다. • 1>> : 출력을 파일로 리다이렉션하는 데 사용되지만 출력을 덮어쓰는 대신 파일 끝에 추가(appending)합니다.
2>, 2>>	• 2> : 표준 에러를 파일로 리다이렉션, 파일이 없으면 새 파일을 만들고, 있으면 기존 파일을 덮어쓰는 데 사용된다. • 2>> : 표준 에러를 파일로 리다이렉션하는 데 사용되지만 출력을 덮어쓰는 대신 파일 끝에 추가(appending)합니다.
&>, >&	표준 출력과 표준 에러를 동시에 파일로 리다이렉션, 파일이 없으면 새 파일을 만들고, 있으면 기존 파일을 덮어씁니다.
&>>	표준 출력과 표준 에러를 동시에 파일로 리다이렉션, 그러나 출력을 덮어쓰는 대신 파일 끝에 출력을 추가(appending)합니다.
tee	화면에 출력을 계속 표시하면서 출력을 파일로 리다이렉션하는 데 사용됩니다.
<>	읽고, 쓰기 위해 파일을 여는 데 사용됩니다.
*	파일 대체 와일드카드, 0개 이상의 문자와 일치
?	파일 대체 와일드카드, 1개의 문자와 일치
[ ]	파일 대체 와일드카드, 포함된 문자 중 하나와 일치 또는 문자 범위 일치
`command` 또는 $(command)	Command Substitution 이라고 한다. 명령을 실행하고 표준 출력으로 대체한다. 서브셸(subshell)에서 명령을 실행하기 때문에 원래의 셸 환경에 영향을 주지 않는다. ex1) echo "Today is $(date)" 또는 echo "Today is `date`" ex2) touch report_$(date +%Y%m%d)_v1 또는 touch report_`date +%Y%m%d`_v2
| 또는 |&	파이프는 추가 처리를 위해 한 명령/프로그램/프로세스의 출력을 다른 명령/프로그램/프로세스로 보내는 기능 '|&'를 사용하면 표준 출력 외에 command1의 표준 오류가 파이프를 통해 command2의 표준 입력에 연결된다. 표준 출력에 대한 표준 오류의 암시적 리다이렉션은 command1에서 지정한 모든 리다이렉션 후에 수행된다. 1) command1 |& command2 2) command1 2>&1 | command2 1)과 2)는 같다. ex1) cat sample2.txt | head -7 | tail -5 ex2) ls -l | find ./ -type f -name "*.txt" -exec grep "program" {} ₩;
;	하나의 라인에 주어진 명령어들을 성공,실패와 관련 없이 전부 실행

기호	의미
||	앞에서부터 순차적으로 실행하되, 명령 실행에 성공하면 뒤에 오는 명령을 실행하지 않는다.
&&	앞에서부터 순차적으로 실행하되, 명령 실행에 실패할 경우 뒤에 오는 명령은 실행하지 않는다.
(list)	명령어를 그룹화, 명령이 그룹화되면 전체 명령 목록에 리다이렉션이 적용될 수 있다. Command Substitution와 다르다 서브셸(subshell)에서 명령을 실행하기 때문에 원래의 셸 환경에 영향을 주지 않는다.
{ list; }	명령어를 그룹화, 명령이 그룹화되면 전체 명령 목록에 리다이렉션이 적용될 수 있다. 중괄호를 사용하여 그룹화하면 현재 셸에서 명령들이 실행된다. 서브셸을 만들지 않는다.
&	executes command in the background and will display the assigned Pid.
#	주석
$	To expand the value of a variable.
₩	백슬래시 '₩'는 Bash 이스케이프 문자이다. 줄바꿈을 제외하고 뒤에 오는 다음 문자의 리터럴 값(literal value)을 유지한다.
''(Single Quotes)	문자를 작은따옴표로 묶으면 안에 있는 각 문자의 리터럴 값이 유지된다. 백슬래시가 앞에 오는 경우에도 작은따옴표 사이에는 작은따옴표가 올 수 없다.
""(Double Quotes)	문자를 큰따옴표로 묶으면 따옴표 안에 모든 문자의 리터럴 값이 유지된다. 예외로 '$', '`'(그레이브 액센트), '₩'는 리터럴로 바뀌지 않는다.(특별한 의미를 유지)
/dev/null	출력을 폐기하고 "black hole"로 보내는 데 사용됩니다.

Shell에서 예약어(Reserved Words)

if	then	elif	else	fi	time
for	in	until	while	do	done
case	esac	coproc	select	function	
{	}	[[	]]	!	

Shell에서 변수 타입

변수 이름에는 알파벳 대소문자(a-z, A-Z), 숫자(0-9), 또는 언더스코어(_)만 사용할 수 있다.
관례상 대문자를 많이 쓴다.

예) _ALI, TOKEN_A, VAR_1, VAR_2

변수_이름=값
(띄어쓰기가 없어야 한다)

```
#!/bin/sh


NAME="Zara Ali"
echo $NAME
```

unset 변수이름
(변수 설정 해제)

```
#!/bin/sh


NAME="Zara Ali"
unset NAME
echo $NAME
```

- Local Variables
  로컬 변수는 셸의 현재 인스턴스 내에 있는 변수다. 셸에 의해 시작된 프로그램에는 사용할 수 없고 터미널(프롬프트)에서 설정된다.
- Environment Variables
  환경 변수는 셸의 모든 자식 프로세스에서 사용할 수 있다. 일부 프로그램은 올바르게 작동하려면 환경 변수가 필요하다.
- Shell Variables
  셸 변수는 셸에 의해 설정되고 올바르게 작동하기 위해 셸에 필요한 특수 변수이다. 일부는 환경 변수이고 나머지는 지역 변수다.

Shell에서 특별한 변수들

기호	의미
$0	실행된 스크립트 파일이름

기호	의미
$n	스크립트가 호출된 인수에 해당한다 위치 매개변수(Positional Parameter) 예를들어 첫 번째 파라미터는 $1, 두 번째 파라미터는 $2, 10번째 부터는 중괄호로 감싸야 한다 ${10}, ${11}, ...
$#	파라미터의 개수
$*	모든 위치 매개변수를 담고있는 문자열
$@	All the arguments are individually double quoted 위에 '$*'와 비슷한데 개별적으로 따옴표로 묶여있다
$$	현재 Shell의 PID
$!	마지막 백그라운드 명령의 프로세스 ID
$?	마지막으로 실행된 명령의 종료 상태(exit status)

이것 말고도 $PWD, $PPID, $PATH, $OLDPWD... 등등 많다.

배열 선언과 접근

배열이름[index]=값
배열이름=(값1 값2 ... 값N)

${배열이름[index]} <- 특정 인덱스에 접근
${array_name[*]} 또는 ${array_name[@]} <- 모든 요소 가져오기

```
#!/bin/sh


NAME[0]="Zara"
NAME[1]="Qadir"
NAME[2]="Mahnaz"
NAME[3]="Ayan"
NAME[4]="Daisy"
echo "First Index: ${NAME[0]}"
echo "Second Index: ${NAME[1]}"
echo "First Method: ${NAME[*]}"
echo "Second Method: ${NAME[@]}"
```

Shell 기본 연산자

- Arithmetic Operators
- Relational Operators
- Boolean Operators
- String Operators
- File Test Operators

산술 연산자(Arithmetic Operators)

변수 a와 b를 각각 10, 20이라 가정하자

연산자	설명	예제
+ (Addition)	플러스 연산	`expr $a + $b` will give 30
- (Subtraction)	마이너스 연산	`expr $a - $b` will give -10
* (Multiplication)	곱하기 연산	`expr $a ₩* $b` will give 200 역슬래쉬(₩)에 주목!!
/ (Division)	나누기 연산	`expr $b / $a` will give 2
% (Modulus)	나머지 연산	`expr $b % $a` will give 0
= (Assignment)	대입 연산	a = $b would assign value of b into a
== (Equality)	비교 연산	[$a == $b] would return false. 모든 조건식은 대괄호 [ ] 안에 있어야 하고 띄어쓰기가 있어야 한다 [$a == $b] <- YES [$a==$b] <- NO
+ (Addition)	플러스 연산	`expr $a + $b` will give 30
** (exponentiation)	거듭 제곱	
id++, id--, ++id, --id	후위 증감(post-increment and post-decrement), 전위 증감(pre-increment and pre-decrement)	
~, &, ^, |	비트연산자 각각 bitwise negation, bitwise AND, bitwise exclusive OR, bitwise OR 이다	
!, &&, ||	논리연산자 각각 NOT, AND, OR 이다.	

연산자	설명	예제
expr ? expr : expr	조건연산자	[["string1" == "string2"]] && echo "Equal" || echo "Not equal"
= *= /= %= += -= <<= >>= &= ^= |=	대입연산자	[$VAR =~ .*Linux.*]
expr1 , expr2	콤마연산자	

Shell 에서 Brackets는 여러가지고 지원하는 Shell도 다르다.
그리고 모든 조건식은 공백이 있는 대괄호 안에 있어야 한다.
[조건식]
[[조건식]]
((조건식))
$((산술식))

관계 연산자(Relational Operators)

▶ 숫자 값에 특정한 관계 연산자를 지원한다. 숫자가 아닌 문자열에 대해서는 동작하지 않는다.

변수 a와 b를 각각 10, 20이라 가정하자

연산자	설명	예제
-eq	두 값이 같으면 true	[$a -eq $b] is not true.
-ne	두 값이 다르면 true	[$a -ne $b] is true.
-gt	왼쪽이 오른쪽보다 크면 true	[$a -gt $b] is not true.
-lt	왼쪽이 오른쪽보다 작으면 true	[$a -lt $b] is true.
-ge	왼쪽이 오른쪽보다 크거나 같으면 true	[$a -ge $b] is not true.
-le	왼쪽이 오른쪽보다 작거나 같으면 true	[$a -le $b] is true.

부울 연산자(Boolean Operators)

변수 a와 b를 각각 10, 20이라 가정하자

연산자	설명	예제
!	NOT 연산(논리적 부정)	[! false] is true.
-o	OR 연산	[$a -lt 20 -o $b -gt 100] is true.
-a	AND 연산	[$a -lt 20 -a $b -gt 100] is false.

문자열 연산자(String Operators)

변수 a와 b를 각각 "abc", "efg"이라 가정하자

연산자	설명	예제
= 또는 ==	두 문자열이 같으면 true.	[$a = $b] is not true.
!=	두 문자열이 다르면 true.	[$a != $b] is true.
string1 < string2	string1이 사전순으로(lexicographically) string2보다 앞서면 true.	[$a < $b] is true.
string1 > string2	string1이 사전순으로 string2보다 뒤에있으면 true.	[$a > $b] is true.
-z	문자열의 길이가 0이면(zero length) true.	[-z $a] is not true.
-n	문자열의 길이가 0이 아니면(nonzero length) true.	[-n $a] is not false.
str	빈 문자열(empty string)이 아닌지 확인 비어 있으면 false	[$a] is not false.

파일 테스트 연산자(File Test Operators)

변수 file을 이름은 "test"이고 크기가 100바이트이고, 읽기(read), 쓰기(write), 실행(execute) 권한이 있는 실제 존재하는 파일이라고 가정하자

연산자	설명	예제
-a file, e file	파일이 존재하면 true	[-a $file] is true.
-b file	블록 특수 파일이면 true	[-b $file] is false.
-c file	문자 특수 파일이면 true	[-c $file] is false.
-d file	파일이 디렉토리이면 true	[-d $file] is not true.
-f file	파일이 디렉토리나 특수 파일이 아닌 그냥 일반 파일이면 true	[-f $file] is true.
-g file	파일이 그룹 ID(SetGID) 비트가 설정되어 있으면 true	[-g $file] is false.
-k file	파일이 Sticky Bit가 설정되어 있으면 true	[-k $file] is false.
-p file	파일이 named pipe (FIFO)이면 true	[-p $file] is false.
-t file	파일 설명자(descriptor)가 열려 있고 터미널과 연결되어 있으면 true	[-t $file] is false.
-u file	파일이 SetUID 비트가 설정되어 있으면 true	[-u $file] is false.
-r file, -w file, -x file	각각 읽기, 쓰기, 실행 권한이 있으면 true	[-r $file] is true. [-w $file] is true. [-x $file] is true.

연산자	설명	예제
-s file	파일 크기가 0보다 크면 true	[-s $file] is true.
-e file	파일(일반파일, 디렉토리파일, 특수파일)이 존재하면 true	[-e $file] is true.
-L file	파일이 존재하고 심볼릭 링크인 경우 true	[-L $file] is true.
-S file	파일이 존재하고 소켓 파일인 경우 true	[-S $file] is true.
file1 -ef file2	file1 과 file2가 같은 장치이고 inode 번호가 같으면 true(동일한 파일에 대한 하드 링크이면 true)	if ["$f1" -ef "$f2"]
file1 -nt file2	file1이 file2 보다 최신 버전(수정 날짜 기준)이거나 file1은 존재하고 file2는 존재하지 않는 경우 true	if ["$f1" -nt "$f2"]
file1 -ot file2	file1이 file2보다 오래된 경우 또는 file2는 존재하고 file1은 존재하지 않는 경우 true	if ["$f1" -ot "$f2"]