**강의명: 유닉스 시스템**

**실습 번호: 2**

**실습 제목: 쉘 프로그래밍(Shell Programming)**

**학생 이름: 황귀훈**

**학번: 201710885**

**1. 변수**

**1.1**



**1.2**



**1.3**

텍스트이(가) 표시된 사진

자동 생성된 설명

**1.4**

텍스트이(가) 표시된 사진

자동 생성된 설명

특수변수 $$는 현재 수행되고 있는 프로세스의 번호이다. 명령어 “echo $$”로 현재 $$변수의 값을 출력했고 “ps”명령어를 통해 bash의 프로세스 번호(pid)와 일치함을 확인했다.

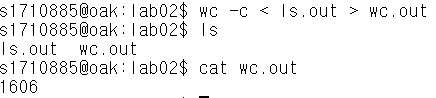
**2. 리디렉션**

**2.1**

테이블이(가) 표시된 사진

자동 생성된 설명

**2.2**



**2.3**

텍스트이(가) 표시된 사진

자동 생성된 설명

**3. 파이프라인**

**3.1**



파이프 라인은 ‘|’ 앞 명령(명령1)의 표준 출력이 ‘|’ 뒤 명령(명령2)의 표준 입력으로 연결된다. 위에서 명령1은 “cat /etc/passwd”이고 명령2는 “wc -c”이다. 명령1을 통해 나온 /etc/passwd의 내용들이 명령2의 표준 입력으로 들어가고 “wc -c”(명령2)가 수행되어 문자 단위로 세어서 출력한 값이 6465이다. 즉 명령1에서 나온 내용들이 6465단어들로 이뤄졌음을 알 수 있다.

**3.2**



위 명령에서 명령1은 “ls /etc”이고 명령2는 “wc -l”로 이뤄진 파이프 라인이다. 명령1의 표준 출력이 명령2의 표준 입력으로 들어가는데 명령1 “ls /etc”는 /etc의 list를 출력하라는 명령으로 그 수행 결과가 명령2의 표준 입력이 된다. 명령2 “wc -l”은 line단위로 표준 입력을 세서 출력하는 명령으로 /etc의 list를 line단위로 센 값이 279이다.

**4. 리스트**

**4.1**



명령 "cd /this && ls -l"은 op1로 ‘$$’을 사용했다. 여기서 ‘$$’는 logical AND 연산으로 앞의 “cd /this”를 성공하면 뒤의 “ls -l”을 수행할 수 있다. 현재 /this directory가 존재하지 않아 명령 “cd /this”가 오류가 났고 따라서 뒤의 명령 “ls -l”는 수행되지 않는다.

**4.2**

텍스트이(가) 표시된 사진

자동 생성된 설명

명령 "cd /this || ls -l"는 op1로 ‘||’을 사용했다. 여기서 ‘||’는 logical OR 연산으로 앞의 “cd /this”가 실패하면 뒤의 “ls -l”을 수행한다. 현재 /this directory가 존재하지 않아 명령 “cd /this”가 오류가 났고 뒤의 명령 “ls -l”가 수행되었다. 이 명령은 현재 directory인 /home/s1710885/unix/lab02의 파일 리스트를 출력했다.

**5. 경로명 확장**

**5.1**

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 결과는 빈 파일을 생성하는 “touch” 명령어를 사용하여 파일 chap1, chap10, chap12, chap3, chap6, chap9, file1, file2을 생성한 뒤 명령 “echo chap?”을 수행한 화면이다. 명령 “echo chap?”은 Pathname Expansions이 사용되었는데 Pathname Expansions은 해당 패턴 문자의 의미에 해당하는 이름의 파일이 존재하면 확장된다. 명령 “echo chap?”에서 패턴 문자는 ‘?’로 어떤 한 글자와 일치를 의미하므로 현재 디렉토리 내에서 이름이 chap으로 시작하고 ‘?’대신에 한 문자와 일치하는 파일을 출력한다. 그 결과 “chap1 chap3 chap6 chap9”가 출력되었다.

**5.2**

텍스트이(가) 표시된 사진

자동 생성된 설명

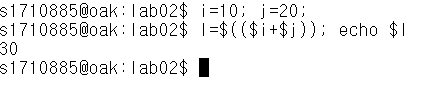
위 결과의 세 명령은 모두 Pathname Expansions을 사용했다. 첫 번째 명령”echo chap\*”은 패턴 문자 ‘\*’을 사용했는데 ‘\*’은 null 스트링 포함한 어떤 스트링과 일치하는 이름의 파일로 확장을 의미한다. 따라서 첫 번째 명령은 “chap”으로 시작하는 파일들을 출력하여 “chap1 chap10 chap12 chap3 chap6 chap9”가 출력 됐다.

두 번째 명령 “echo chap[1-6]”은 패턴 문자 ‘[…]’을 사용했다. 대괄호 안의 어떤 한 문자와 일치하면 확장되는데 두 번째 명령의 대괄호 안에는 “1-6”이 들어가 있고 이것은 1부터 6사이의 한 문자와 일치하면 확장된다. 따라서 두 번째 명령의 결과로 “chap1 chap3 chap6”이 출력 되었다.

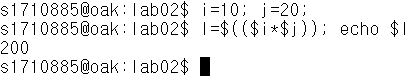
세 번째 명령 “echo \*[12]\*”은 패턴 문자로 ‘\*’과 ‘[…]’이 사용되었는데 어떤 스트링이 앞에 있고 중간에 문자 ‘1’또는 ‘2’또는 ‘12’가 포함되고 뒤에 어떤 스트링이 오면 확장된다. 따라서 세 번째 명령의 결과로 “chap1 chap10 chap12 file1 file2”가 출력 되었다.

**6. 연산 대치**

**6.1**



**6.2**



**7. 테스트**

**7.1**



위 명령은 정수 테스트로 정수 연산자 ‘-eq’를 사용하였다. 테스트의 사용 형식은 [[식]] 혹은 [식]으로 식의 계산 결과가 true면 0 false면 1을 반환한다. 위 명령에서 i에는 10을 저장하고 j에는 20을 저장했고 정수연산자 ‘-eq’로 i의 값과 j의 값이 같은지를 테스트했는데 10 ≠ 20이므로 테스트 결과 1(=false)가 나온다. 특수 변수 ‘$?’은 최근 수행한 명령어 혹은 함수의 반환 값으로 “echo $?” 의 결과로 1이 출력 되었다.

**7.2**



위 명령은 문자열 테스트로 문자열 연산자 ‘==’을 사용하였다. 테스트의 사용 형식은 [[식]] 혹은 [식]으로 식의 계산 결과가 true면 0 false면 1을 반환한다. str1에는 boy를 str2에는 girl을 저장했다. 문자열 연산자 ‘==’을 사용하여 str1의 값과 str2의 값이 같은지를 테스트했는데 boy ≠ girl 이므로 테스트 결과 1(=false)가 나온다. 특수 변수 ‘$?’은 최근 수행한 명령어 혹은 함수의 반환 값으로 “echo $?” 의 결과로 1이 출력 되었다.

**7.3**



위 명령은 정수 테스트로 정수 연산자 ‘-le’과 논리 연산자 ‘-o’를 사용했다. 테스트의 사용 형식은 [[식]] 혹은 [식]으로 식의 계산 결과가 true면 0 false면 1을 반환한다. 위 명령에서 논리연산자 앞은 ‘10 ≤ 20’인지 true/false판단이고, 뒤는 ‘20 ≤ 10’인지 true/false판단이다. 10은 20보다 작으므로 true이고 20은 10보다 작지 않으므로 false인데 true와 false의 논리 연산 ‘or’은 true이므로 테스트 결과는 0(=true)이다. 특수 변수 ‘$?’은 최근 수행한 명령어 혹은 함수의 반환 값으로 “echo $?” 의 결과로 0이 출력 되었다

**7.4**

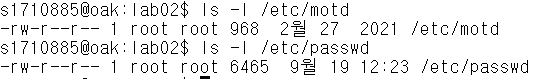


위 명령은 파일 테스트로 파일 연산자 ‘-e’가 사용되었다. 테스트의 사용 형식은 [[식]] 혹은 [식]으로 식의 계산 결과가 true면 0 false면 1을 반환한다. 파일 연산자 ‘-e’는 주어진 파일이 존재하는 지를 테스트하는데 /etc/motd는 존재하기 때문에 테스트 결과 0(=true)가 나온다. 특수 변수 ‘$?’은 최근 수행한 명령어 혹은 함수의 반환 값으로 “echo $?” 의 결과로 0이 출력 되었다.

**7.5**



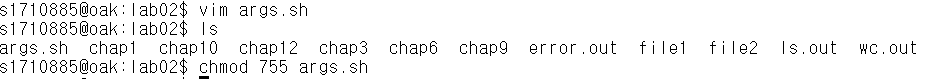
위 명령은 파일 테스트로 파일 연산자 ‘-nt’가 사용되었다. 테스트의 사용 형식은 [[식]] 혹은 [식]으로 식의 계산 결과가 true면 0 false면 1을 반환한다. 파일 연산자 ‘-nt’는 ‘-nt’앞의 파일이 뒤의 파일보다 최근에 수정되었는지 테스트한다. 특수 변수 ‘$?’은 최근 수행한 명령어 혹은 함수의 반환 값으로 “echo $?” 의 결과로 1(=false)이 출력 되었기 때문에. /etc/passwd가 /etc/motd보다 최근에 수정된 것을 의미한다.



실제 확인결과. /etc/passwd가 /etc/motd보다 최근에 수정되었다.

**8. 쉘 프로그램**

**8.1**



텍스트이(가) 표시된 사진

자동 생성된 설명

vim을 사용하여 “args.sh”을 작성하고 파일 mode를 755로 변경했다.

**8.2**

텍스트이(가) 표시된 사진

자동 생성된 설명

명령 ./args.sh one two “long three”는 세개의 인수 “one”, “two”, “long three”로 args.sh를 실행하라는 명령이다. args.sh는 특수 변수들을 사용하여 인수를 받아와 i에 저장하여 “echo”명령어로 출력하도록 프로그램 되어있다. (for i in 특수변수 do echo done)

첫 번째 단락은 인수를 특수 변수 ‘$\*’로 받아오는데 ‘$\*’은 $1(첫 번째 인수)부터 모든 인수가 연결된 스트링을 의미한다. 따라서 첫 번째 단락에서 ‘$\*’은 one two long three와 같고 네 번의 “echo”명령어를 수행하여 space로 구별하여 각 스트링을 출력한다.

두 번째 단락은 인수를 특수 변수 ‘$@’로 받아오는데 ‘$@’는 ‘$\*’과 마찬가지로 $1(첫 번째 인수)부터 모든 인수가 연결된 스트링을 의미한다. 따라서 두 번째 단락에서 ‘$@’은 one two long three와 같고 네 번의 “echo”명령어를 수행하여 space로 구별하여 각 스트링을 출력한다.

세 번째 단락은 인수를 특수 변수 ‘“$\*”’로 받아오는데 ‘“$\*”’는 1개의 인수가 된다. 즉 “one two long three”을 하나의 인수로 취급해서 “echo”명령어가 한 번 수행되고 one two long three가 출력된다.

네 번째 단락은 인수를 특수 변수 ‘“&@”’로 받아오는데 ‘“$@”’는 1개의 인수가 되는 것이 아니라 인수 개수만큼의 스트링 역할을 한다. 즉 네 번째 단락에서 “&@”은 인수 “one”, “two”, “long three”로 “echo”명령어는 세번 수행된다.