BASH 프로그래밍 입문 하우투

GMike

도운이

전혜진

이 문서는 셸 스크립트 프로그래밍의 기본적인 내용을 설명하자는 취지로 작성된 것이며, 제목을 보면 알 수 있겠지만 기초적인 내용을 다루고 있다. 내가 특별히 뛰어나거나 전문적인 셸 프로그래머라서 이런 글을 쓰게 되었다기 보다는, 내가 이 글을 쓰는 과정에서 다른 사람에게도 지식을 전달할 수 있으며 나에게도 공부가 될 것이라는 생각으로 이 글을 쓰기 시작한 것이다. 질문 사항이나 첨가할 내용이 있다면 언제라도 연락해 주었으면 한다. 그런 내용은 추후에 이 문서에 추가할 것이다.

1. 시작하면서

1.1. 이 문서의 최신 버전을 입수하려면?

http://www.linuxdoc.org/HOWTO/Bash-Prog-Intro-HOWTO.html

1.2. 이 글을 읽기 전에

아무리 BASH의 기초적인 내용을 다룬다고 해도, GNU/LINUX 명령 체계에 관한 지식은 있어야 한다. 리눅스의 기초까지 챙겨 주는 문서는 아니므로, 미리 명령 체계를 공부해 두고 이 문서를 읽도록 하자.....라고 거창하게 떠들고는 있지만, ls, mv 같은 기본 명령하고, vi 쓸 줄만 알면 되니 하나도 신경 쓸 것 없다. 그정도를 모르는 분이 kldp에 와 계실 가능성은 드물다고 생각한다.

1.3. 이 문서가 필요한 사람은

이 문서는 다음과 같은 사람들을 위한 것이다.

- 프로그램에 대한 아이디어를 셸 스크립트로 구현해 보고자 하는 사람
- 셸 프로그래밍에 관한 막연한 개념은 있지만 정리를 해 줄 필요가 있을 때

- 직접 프로그램을 작성하기에 앞서 예제와 참고할 만한 말들을 읽어 보고 싶을 때
- DOS/Windows 사용자로서, 배치 파일을 만들어 보고자 할 때
- 나와 있는 하우투란 하우투는 다 읽어 봐야 직성이 풀릴 만큼 호사가일 때
- 그리고 아마도, 위에 나와 있는 관심있는 사람들과는 상관없이, 숙제 때문에 보는 사람도 있을 지 모르겠다. ^_^

2. 아주 간단한 스크립트**!!**

이 하우투는 예제를 바탕으로 셸 스크립트 프로그래밍의 방법을 제시할 것이다.

먼저 이해하기 쉬운 간단한 스크립트를 사용하여 기본 문법을 알아보자.

2.1. hello world를 출력하는 스크립트

```
#!/bin/bash
echo Hello World
```

이 스크립트는 단지 두 줄로 되어 있을 뿐이지만, 스크립트로서 갖추어야 할 기본적인 요소는 갖추고 있다. 먼저 첫 번째 줄에서는 이 파일을 실행하기 위해 필요한 프로그램의 경로를 명기했다. 아마도 다들 알고 있겠지만, bash란 셸의 일종이며, /bin/bash는 이 셸의 실행 파일이 있는 경로이다. 셸은 명령어 해석기로 들어오는 명령을 해석하여 이것이 내부 명령어라면 바로 실행을, 외부 명령어라면 해당하는 실행 파일을 찾아 메모리에 필요한 부분을 적재한다. 이 부분이 없다면 다음에 오는 명령을 실행할 수는 없을 것이다.

두 번째 줄은 실제적인 명령을 내리는 부분이다. 여기에서는 'Hello World'라는 말을 터미널에 출력하라는 명령을 내렸다. 물론 다른 말로 바꾸어도 좋을 것이다.

여담! 우리의 리누스 토발즈 아저씨가 처음 만들었던 것도 화면에 문장을 출력하는 프로그램이었다. 아마 그 프로그램의 업버전은 'sarah is the best'라는 문장이었다나 뭐라나..... (사라는 리누스 아저씨의 동생 이름이다.)

만약에 **./hello.sh: Command not found.** 와 같은 꼴을 보게 되었다면, 아마도 첫 번째 줄의 '#!/bin/bash'가 문제일 것이다. 'whereis bash'나 'finding bash' 명령을 내려 bash의 경로를 확인해 보자. 그리고 이 경로를 첫 번째 줄에 기입한다.

2.2. 엄청나게 간단한 백업 스크립트

```
#!/bin/bash
tar -cZf /var/my-backup.tgz /home/me/
```

이 스크립트는 터미널에 메시지를 출력하는 대신 사용자의 홈 디렉토리를 tar-ball형식으로 묶는다. 이 스크립트는 실제로 사용할 만한 것은 아니다. 먼저 사용자에 따라 경로를 일일히 조절해야 하며, 여러 사람이 사용하면 백업 파일이 서로 덮어 쓰게 되어 백업이 망가질 수도 있다. 이런 백업 스크립트를 실제로 사용할 만 하게 만들어 놓은 것이 뒷 부분에 나올 것이다. 이것은 그냥 예제일 뿐이니까, 한번 시험해 보고는 바로 삭제하는 것이 좋을 것이다.

3. 리디렉션의 모든 것

3.1. 정의와 기본 설명

stdin(표준 입력), stdout(표준 출력), stderr(표준 에러)라는 세 가지의 파일 디스크립터가 있다. 파일 디스크립터가 무엇인지 일일히 설명하기는 그렇고.... 간단하게 이해해 보자. 어떤 프로그램을 실행했을 때 화면에 주루룩 하고 원하는 결과가 뜨는 것이 표준 출력이다. 어떤 프로그램을 실행하기 위해 꼭 필요한 요소를 프로그램을 실행할 때 같이 입력해 주는 것이 표준 입력이다. 이 정도만 알고 시작하면 될 것이다.

먼저 다음과 같은 내용을 간단히 실행해 보자. 백문이 불여일견, 백견이 불여일행인 법. ^^

1. 표준 출력을 파일로 보내기
2. 표준 에러를 파일로 보내기
3. 표준 출력을 표준 에러로 보내기
4. 표준 에러를 표준 출력으로 보내기
5. 표준 에러와 표준 출력을 파일로 보내기
6. 표준 에러와 표준 출력을 표준 출력으로 보내기
7. 표준 에러와 표준 출력을 표준 에러로 보내기

먼저 간단히 말해두겠는데, 버퍼에 남아 있는 표준 출력이나 표준 에러를 화면에 출력해 보는 것은 아주 쉽지만, 그 시도와 동시에 내용이 날아간다는 점을 상기하자.

3.2. 예제 : 표준 출력을 파일로 보내기

프로그램의 실행 결과를 파일에 바로 저장하는 방법이다. 이 방법은 언제 쓰면 좋으냐 하면.... 그렇다, 수치해석 숙제 할 때 줄줄이 에러값 나오는 것, 이 방법으로 하면 파일에 기록하는 코드 없이도, 화면 캡처 따위 하지 않아도 아주 간단하게 해결된다. ^^

```
ls -l > ls-l.txt
```

이 내용을 실행하고 나면, 화면에 'ls -l' 명령의 실행 결과가 출력되는 대신 'ls-l.txt'라는 이름의 파일이 생긴다. 이 파일을 열어 보면 원래 'ls -l' 을 실행했을 때의 결과가 그대로 저장되어

있다. 이런 이름의 파일이 원래 있었다면, 파일의 앞부분에 그대로 출력 내용이 덮어 씌워져 버리니 주의하자.

3.3. 예제 : 표준 에러를 파일로 보내기

방법은 마찬가지이다. 이 경우에는 에러 메시지를 파일에 저장하는 것 뿐이다. 역시 같은 방법으로 다음과 같이 해 보자.

```
grep da * 2> grep-errors.txt
```

앞서 표준 출력을 파일로 보냈을 때와 마찬가지로, 'grep-errors.txt' 파일이 만들어지며 화면 대신 이 파일 안에 에러 메시지가 기록된다. 여기에서 숫자 2는 표준 에러를 뜻하는 파일 디스크립터이다. 2가 들어가지 않으면 표준 출력이 저장된다.

3.4. 예제 : 표준 출력을 표준 에러로 보내기

이 방법은 프로그램을 실행했을 때의 표준 출력을 표준 출력 디스크립터 대신 표준 에러와 같은 방식으로 출력하는 것이다.

```
grep da * 1>&2
```

이와 같은 방법으로 명령의 표준 출력이 표준 에러에게 넘겨졌다. 그런데 정말 이것만으로 이해가 착착 간다면 정말 이 문서 볼 필요 없다. 어쩔 수 없이 추가 설명을 달아야만 하는 부분인데..... 간단히 설명하겠다. 'grep da *'는 우리가 익히 알고 있는 단순한 명령이다. 이 뒤에 붙은 숫자는 파일 디스크립터이며 특히 1은 표준 출력을 의미한다. 2는 표준 에러 되겠다. 그런데 사실, 조금만 신경 쓰면 알 수 있는 부분이지만 '1>'는 그냥 '>'와 같은 것이다. 이유는 각자 생각해 보자. 금방 결론이 나올 것이다. 참고로 0은 표준 입력, 3 이후는 다른 파일이 입출력용으로 할당될 때 사용하는 것으로, 그냥 적어 넣으면 에러 메시지를 볼 수 있을 것이다.

리디렉션을 '2>&1'과 같이 주면 표준 에러를 표준 출력과 같은 곳으로 보내라는 뜻이며, '1>&2'의 경우는 표준 출력을 표준 에러와 같은 곳으로 보내라는 뜻이다.

3.5. 예제 : 표준 에러를 표준 출력으로 보내기

위 내용을 잘 섭렵해 왔다면 생각해 볼 필요도 없는 부분이다.

```
grep * 2>&1
```

자, 표준 에러를 표준 출력으로 보내어 함께 출력하였다. 만약에 이 결과를 파이프를 통해 more나 less 등으로 보낸다면, 표준 출력과 표준 에러가 뒤섞인 형태를 보게 될 것이다.

3.6. 예제 : 표준 출력과 표준 에러를 파일로 보내기

이 내용 역시 위와 똑.같.다. 라고 봐도 무방하지만..... 이것을 한번에 파일로 보내는 부분은 함께 보도록 하자.

```
rm -f $(find / -name core) &> /dev/null
```

따로이 디스크립터를 사용할 필요 없이 '&>'로 리디렉션 하는 것 만으로 해결된다. 이 방식은 표준 출력과 표준 에러를 함께 파일로 보낼때 사용한다. 이 명령을 크론에 넣어 두면, 일정한 시간마다 모든 디렉토리에 있는 core 파일을 삭제한다. 여기에서는 표준 출력과 표준 에러를 모두 '/dev/null'로 보내 실제로는 날려 버리지만, 굳이 이 내용을 확인하고 싶다면 적당한 파일로 출력하여 살펴보도록 하자.

4. 파이프(pipes)

이 장에서는 파이프를 사용하는 간단하면서도 실질적인 예를 들어 보겠다.

4.1. 파이프로 무엇을 할 수 있을까

파이프는 정말 간단한 방법으로, 한 프로그램에서의 출력을 다른 프로그램의 입력으로 보낸다. 이렇게 "흘려 보내기"때문에 파이프다.... 라고 생각하면 딱 맞을 것이다. ^^

4.2. 예제 : sed를 이용한 아주 간단한 파이프 예제

다음은 파이프를 이용하는 가장 간단한 실례이다.

```
ls -l | sed -e "s/[aeio]/u/g"
```

그러면 한번 위 내용을 찬찬히 살펴보자. 먼저 'ls -l' 명령이 실행되었다. 그리고 이 결과는 화면에 출력되는 대신, 파이프를 타고 sed 프로그램에 전달되어 다시 한번 처리된 후 화면에 출력된다.

4.3. 예제 : 'ls -l *.txt'의 색다른 형태

아마도 이 방법은 'ls -l *.txt'를 사용하는 것 보다는 까다로운 것이겠지만, 파이프의 사용에 대해서만큼은 확실히 보여줄 수 있다.

```
ls -l | grep "\.txt$"
```

여기서 'ls -l'의 결과는 grep으로 넘어가, "\.txt$"라는 조건에 맞는 값만을 화면에 출력한다. 이 결과는 'ls -l *.txt'와 같다.

5. 변수

다른 프로그래밍 언어를 사용할 때와 마찬가지로, 셸 프로그래밍을 할 때도 변수를 사용할 수 있다. 게다가 bash에서는 데이터 타입을 미리 정할 필요 없이, 숫자나 문자, 혹은 문자열을 지정할 수 있다.

어떤 변수가 처음 사용되는 순간 참조가 생성되므로, 변수를 따로 선언할 필요는 없다.

5.1. 예제 : 변수를 사용한 Hello World! 출력

```
#!/bin/bash
STR="Hello World!"
echo $STR
```

2번째 줄에서 STR이라는 이름의 변수가 생성되며 "Hello World!"를 받았다. 이 변수에 들어 있는 값을 사용하기 위해서는 '$'를 변수 이름 앞에 사용하여 이것이 변수라는 사실을 알려 주어야 한다. 변수 이름 앞에 '$'를 빼놓았을 경우에는 예상했던 것과는 다른 결과가 나올 것이다. 예컨대, 이런 경우 셸은 이 변수 자체를 문자열로 인식해 버리거나 하기 때문에 주의해야 한다.

5.2. 예제 : 여전히 간단한 백업 스크립트

```
#!/bin/bash
OF=/var/my-backup-$(date +%Y%m%d).tgz
tar -cZf $OF /home/me/
```

이 스크립트는 앞서 살펴본 것과는 질적으로 다른 물건이다. 일단 이것은 생성 날짜가 파일 이름이 되기 때문에 파일명이 중복되지 않아, 백업하기 용이하다. 두 번째 줄의 '$(date +%Y%m%d)'가 바로 그 부분이다. 매일 파일 이름이 달라진다는 것을 알 수 있다. date는 원하는 대로 옵션을 조절하여 다른 포맷으로 출력할 수도 있으니 확인해 보자. 또한 이 스크립트를 실행하면 명령 실행 과정이 화면에 그대로 출력된다는 점도 참고해 두자.

참고할 만한 것을 좀 보자. 다음 내용을 각각 직접 실행해 보자.

```
echo ls
echo $(ls)
```

무엇이 다른지 확인하고, 왜 다른지도 알아보자.

5.3. 지역 변수

지역 변수는 **local**이라는 키워드를 사용하여 생성된다.

```
#!/bin/bash
HELLO=Hello
function hello {
local HELLO=World
echo $HELLO
}
echo $HELLO
hello
echo $HELLO
```

이 예제는 지역 변수를 사용하는 방법에 대한 것이다. 이미 프로그래밍 언어를 조금이라도 다루어 보았다면 이런 방식의 코드를 본 적이 있을 것이다. 여기에서는 함수를 호출하는 방법에도 주의하자.

6. 조건문

조건문은 어떤 일을 수행하는가 혹은 하지 않는가의 문제를 표현하기 위한 방법이다.

6.1. 요점만 말하자면

조건문은 다양한 형식으로 표현할 수 있지만 가장 대중적으로 사용하는 것은 다음과 같은 형식이다.

if expression then statement

'statement'는 'expression'에 해당하는 조건이 충족되었을 때만 실행된다. C언어 책 사서 첫 장만 보고 손 안댄 분이라 할지라도 구경해 보셨을 만한 형식이다.

조건문의 형식 중에는 다음과 같은 것도 있다.

if expression then statement1 else statement2.

이것 역시 많이들 보신 형태이겠지만, 'expression'에 해당하는 조건이 충족되면 'statement1'이 실행되고, 그렇지 않으면 'statement2'가 실행되는 방식이다.

앞서 말한 것 보다 조금 더 진화된 형태를 살펴보자면 다음과 같다. **if expression1 then statement1 else if expression2 then statement2 else statement3**. 이 방식에서도 'expression1'을 충족하면 'statement1'이 실행된다는 점에서는 앞서 살펴본 예와 같다. 그러나 'expression1'이 충족되지 않을 경우에는 'expression2'라는 새로운 조건이 나타난다. 만일 이렇게 'expression2'와 비교하여 조건을 충족하면 statement2'가 실행되지만 그렇지 않을 경우에는 'statement3'이 실행된다. 이 과정은 'else if'라는 녀석을 이용하는데, 'else if'를 얼마든지 추가하여 다양한 분기를 만들 수 있다.

간단한 문법 형식을 보도록 하자.

bash에서 if를 사용할 때에는 다음과 같은 형식으로 작성한다.

if [expression];

then

('expression' 이 참일때 실행할 코드)

fi

6.2. 예제 : 단순한 **if - then** 조건문

```
#!/bin/bash
if [ "foo" = "foo" ]; then
echo expression evaluated as true
fi
```

이 코드는 if 뒤에 나온 조건이 참일때 'expression evaluated as true' 라는 문장을 출력하는 내용이다. 보면 알겠지만 조건 뒤에는 'then' 이라고 적혀있고, 조건이 참일때 실행할 내용을 적은 뒤에는 'fi' 라고 적어 조건문이 끝났음을 알려준다.

6.3. 예제 : 단순한 **if-then-else** 조건문

```
#!/bin/bash
if [ "foo" = "foo" ]; then
echo expression evaluated as true
else
echo expression evaluated as false
fi
```

6.4. 예제 : 변수를 사용하여

```
#!/bin/bash
T1="foo"
T2="bar"
if [ "$T1" = "$T2" ]; then
echo expression evaluated as true
else
echo expression evaluated as false
fi
```

7. for, while, until을 사용하는 순환문

여기에서는 for, while, until을 사용한 순환문에 대해 알아보겠다.

for를 사용한 순환문의 경우 일반적인 프로그래밍 언어를 사용할 때와 좀 다른 점이 있다. 예컨대 문자열에서의 각 단어를 지나쳐가며 문자열이 끝날 때 까지 루프를 사용할 수도 있다.

while의 조건이 참인 동안에는 계속 루프 안의 명령이 동작한다. 하지만 조건이 거짓인 경우 루프는 바로 중지되고 루프 밖의 코드를 실행하는 것이다.

until도 따지고 보면 같은 맥락이기는 하지만, 이 경우에는 조건이 거짓인 동안 루프가 동작한다는 점이 다르다.

물론 사용하는 사람 입장에서는 while이나 until이나 경우에 따라 편하게 써 주면 되는 것이다. ^^

7.1. 예제 : for

```
#!/bin/bash
for i in $( ls ); do
echo item: $i
done
```

두 번째 줄에서 특이한 변수를 하나 볼 수 있다. 이것은 ls의 실행 결과를 통째로 변수로 받고 있다. 여기에서 $i는 ls 의 실행 결과를 순서대로 한 단어씩 받는다.

세 번째 줄은 루프가 진행되는 동안 실행할 내용이다. 여기에서는 한 줄 짜리로 되어 있지만 필요에 따라 몇 줄이고 추가할 수 있다. 얼마든지 추가한 다음, 마지막 내용의 다음 줄에 done을 입력하여 루프가 끝이라고 알려 주면 된다.

마지막 줄의 'done'은, $i가 지금 받아서 사용한 변수는 폐기하고 새로운 변수를 받아야 한다는 뜻이다. done 이전에 있던 내용은 실행되었고, 이제 다시 새 변수를 넣어 같은 내용을 진행할 것이다.

이 스크립트는 정말 단순하기 짝이 없지만, 루프에 대해 필요한 내용은 다 구색이 갖춰져 있다. 이 내용만 제대로 이해해도 기본적으로 루프를 사용하는 데에는 어려움이 없을 것이다.

7.2. 예제 : C언어 방식으로 사용하는 for

이런 방식으로 사용하는 for도 한번 생각해 보자. 이것은 C나 Perl에서 사용하는 방식과 비슷한 구석이 많다.

```
#!/bin/bash
for i in `seq 1 10`;
do
echo $i
done
```

7.3. 예제 : while

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]; do
echo The counter is $COUNTER
let COUNTER=COUNTER+1
done
```

이 스크립트는 C나 파스칼, Perl 등의 유명한 언어에서 사용하는 방식을 흉내내고 있다.

7.4. 예제.... 하나 더!

```
#!/bin/bash
COUNTER=20
until [ $COUNTER -lt 10 ]; do
echo COUNTER $COUNTER
let COUNTER-=1
done
```

8. 함수

대부분의 프로그래밍 언어에서는 논리적인 흐름과 재귀 표현 등을 위해 함수라는 이름으로 코드를 묶어 사용하는 일이 많다.

함수를 정의하는 데에는 심란하고 어려운 방법 같은 것은 전혀 필요하지 않다. 단지 이렇게 입력할 수만 있으면 된다. function my_func { my_code } 하고 말이다.

함수를 호출할 때는 다른 프로그램을 호출하는 것과 똑같이 이름을 적어주면 된다.

8.1. 함수 예제

```
#!/bin/bash
function quit {
exit
}
function hello {
echo Hello!
}
hello
quit
```

```
echo foo
```

2~4줄은 'quit' 함수이며, 5~7줄은 'hello' 함수이다. 이 스크립트가 무슨 뜻인지 잘 이해가 가지 않는다면, 백문이 불여일견이다. 한번 실행해 보면 무슨 말인지 다 알 것이다.

함수를 선언하는 데에는 특별히 신경 쓸 만한 일은 전혀 없다. 넘어가자.

이 스크립트가 실행되면 먼저 hello 함수가 호출되고 다음으로 quit 함수가 호출된다. 10번째 줄은 결코 실행되지 않을 것이다.

8.2. 인자를 받는 경우

```
#!/bin/bash
function quit {
exit
}
function e {
echo $1
}
e Hello
e World
quit
echo foo
```

이 스크립트는 방금 전에 살펴 본 것과 매우 비슷하다. 가장 큰 차이는 'e'라는 이름의 함수이다. 이 함수는 받아들인 첫 번째 인자를 출력한다.

9. 사용자 인터페이스

9.1. select를 이용하여 간단한 메뉴 만들기

```
#!/bin/bash
OPTIONS="Hello Quit"
select opt in $OPTIONS; do
if [ "$opt" = "Quit" ]; then
echo done
exit
elif [ "$opt" = "Hello" ]; then
echo Hello World
else
clear
echo bad option
```

```
fi
done
```

이 스크립트를 실행하면 여러분은 프로그래머의 로망(-_-;;;)인 텍스트 기반의 메뉴를 보게 될 것이다. 또한 위 내용은 for를 사용할 때와 마찬가지로 $OPTION을 이용하여 각 단어를 받았지만, 사용자에게 입력을 받아 선택하게 할 수 있는 것이다.

9.2. 명령 행을 사용하여

```
#!/bin/bash
if [ -z "$1" ]; then
echo usage: $0 directory
exit
fi
SRCD=$1
TGTD="/var/backups/"
OF=home-$(date +%Y%m%d).tgz
tar -cZf $TGTD$OF $SRCD
```

이 프로그램의 첫 번째 분기는 이것이 ($1)의 값을 반환할 수 있는가를 확인하여 값을 반환할 경우에는 프로그램을 종료한다. 그렇지 않을 경우에는 짧은 경고문과 함께 스크립트를 계속 진행해 나간다.

10. Misc

10.1. read로 사용자 입력 받기

많은 경우 프롬프트 상에서 사용자의 입력을 받아야 할 지도 모른다. 다음은 프롬프트에서 사용자의 입력을 받아들이는 방법이다.

```
#!/bin/bash
echo Please, enter your name
read NAME
echo "Hi $NAME!"
```

이 방법은 2개 이상의 입력을 동시에 받아 들일 수 있다.

```
#!/bin/bash
echo Please, enter your firstname and lastname
read FN LN
echo "Hi! $LN, $FN !"
```

10.2. 산술 계산

명령 행이나 셸에서 다음과 같이 입력해 보자.

echo 1 + 1

뭐가 나오겠는가? 입력한 그대로 나와 버린다! 흠, 이 경우에 2라는 결과가 나온다면 뭔가 실수한 거다. 그렇다면 어떻게 하면 이 계산의 결과를 화면에서 볼 수 있을까. 방법은 간단하다. 이것이 문자열이 아니라 계산이라는 것을 인식하면 되는 것이고, 더 나아가 이 계산을 변수에 넣어 버리면 echo 명령으로 볼 수 있는 것은 인지상정인 것이다. ^^ 그렇다~! 답은 바로 저것이다!

echo $((1+1))

좀 더 논리적이고 제대로 된 방법은 저것이다. 수학적인 표현으로 한 방에 인식해 버린다. 효과는 확실하다!!

echo $[1+1]

나눗셈을 제대로 하고 싶거나 혹은 더 복잡한 수학 계산을 하고 싶다면 수학적인 계산을 bc에서 처리해 버리는 것이 현명하다. 아는 분은 다 아는 이야기이지만 모르는 분을 위해 아주 짧게 떠들어 보겠다. bc는 터미널에서 그냥 입력해서 사용할 수 있는 계산기로서, 아주 정밀한 소수점 계산까지 할 수 있고, 공학 계산도 식만 세울 수 있다면 웬만하면 풀 수 있는데다가 마치 수학 패키지를 사용하듯이, 어느 정도 간단한 스크립트 형태를 만들어 수학 계산을 할 수도 있는, 그야말로 끝내주는 물건이다. 이야기가 길어졌는데, 백견이 불여일행이라고 뭐가 다른지 한번 보기나 하자.

3/4를 계산한다고 "echo $[3/4]" 라고 입력해 보자. 원래 정상적인 답은 0.75이겠지만 여기에서는 멍청하게도 0이라고 나올 것이다. 그도 그럴 것이, 배쉬에서는 오직 정수 계산만 할 수 있기 때문이다. "echo 3/4|bc -l"라고 입력하여 이 내용을 bc로 보내면 0.75라는 답을 얻을 수 있다.

10.3. bash의 경로 찾기

이 부분은 마이크가 추가해 준 부분이다.

우리는 대개 스크립트의 첫 줄에 #!/bin/bash 라고 입력하지만, 여기에서는 다른 경로에 bash가 있을 때 그것을 찾아내는 방법을 간단히 짚고 넘어가겠다. 대부분의 시스템에서는 /bin/bash의 경로에 있겠지만, 모든 시스템이 그렇다고는 말할 수 없기 때문이다.

’find ./ -name bash’ 명령은 루트 디렉토리부터 모든 디렉토리를 쓸어내리며 bash를 찾는다. 그러나 이렇게 까지 할 것은 없다. bash가 있는 위치라는 것은 대개 다음 중 하나인 것이다. 이 경로를 모두 확인해 보고도 없을 때에야 find를 써 보자. 사실, 모든 디렉토리를 확인하는 것은 좀 시간 낭비다.

ls -l /bin/bash

ls -l /sbin/bash

ls -l /usr/local/bin/bash

ls -l /usr/bin/bash

ls -l /usr/sbin/bash

ls -l /usr/local/sbin/bash

혹은 'which bash'를 사용하여 찾을 수도 있다. 취향에 따라 써 보자.

10.4. 프로그램이 되돌리는 값 받기

bash에서, 프로그램이 되돌리는 값은 특별한 변수인 $?로 받는다.

다음은 어떻게 프로그램이 되돌리는 값을 받아내느냐를 보여 주는 것이다. 나는 여기에서 **dada** 라는 디렉토리가 존재하지 않는다고 가정하겠다. 화면에 뭔가 나오지 않는가?

```
#!/bin/bash
cd /dada &> /dev/null
echo rv: $?
cd $(pwd) &> /dev/null
echo rv: $?
```

10.5. 명령 실행 결과 저장하기

먼저 MySQL이 깔려 있다는 가정 하에서 하는 말이지만, 이번에 설명할 간단한 스크립트는 이 시스템에 존재하는 모든 데이터 베이스의 모든 테이블 목록을 볼 수 있게 하는 것이다. 물론 mysql 명령을 사용하는 데 필요한 사용자 이름과 패스워드는 있어야 할 것이다. ^^

```
#!/bin/bash
DBS=`mysql -uroot -e"show databases"`
for b in $DBS ;
do
mysql -uroot -e"show tables from $b"
done
```

10.6. 다중 소스파일 사용하기

여러 파일에 걸쳐 명령 소스를 입력하여 실행할 수도 있다.

필자는 이 부분을 그냥 알아서 해 보라고 비워 두었고 아직까지 채울 기미가 보이지를 않지만..... 뭐, 말하자면 간단하다. 예컨대, 인터넷에서 원하는 만화 그림 파일을 매일 아침 다운로드 받는 스크립트가 있다고 가정하자. 이 만화 그림을 매일 바탕 화면에 깔아 놓고 싶다면 이 내용을 새로운 스크립트에 통째로 집어 넣어도 상관 없겠지만, 원하는 그림을 바탕 화면에 자동으로 깔아 놓는 스크립트에서 이 그림을 다운받는 스크립트를 불러들여 실행해도 아무 상관이 없다는 뜻이다. 이와 같은 다중 소스파일은 다른 프로그램에서도 같은 기능을 또 사용할 때 새 프로그램 안에서 불러 들이는 것 만으로도 충분히 제 기능을 하기 때문에 편리하게 이용할 수 있다.

11. Tables

11.1. 문자열 비교 연산자

(1) s1 = s2

(2) s1 != s2

(3) s1 < s2

(4) s1 > s2

(5) -n s1

(6) -z s1

지금 설명하는 내용은 문자열을 비교하는 연산자이다. 이 연산자를 문장 형대로 바꾸면 다음과 같다.

(1) s1 matches s2

(2) s1 does not match s2

(3) __TO-DO__

(4) __TO-DO__

(5) s1 is not null (contains one or more characters)

(6) s1 is null

11.2. 문자열 비교의 예제

다음은 문자열 $S1과 $S2를 비교하는 예제이다.

```
#!/bin/bash
S1='string'
S2='String'
if [ $S1=$S2 ];
then
echo "S1('$S1') is not equal to S2('$S2')"
fi
if [ $S1=$S1 ];
then
echo "S1('$S1') is equal to S1('$S1')"
fi
```

보통의 프로그래밍 언어에서 사용하는 방법과 아주 비슷하다.

이 부분에 대해서는 안드레 베크의 메일을 참고하였다.

이것은 상당히 좋은 방법이다. $S1이나 $S2 둘 중 하나라도 비어 있을 때에는 에러가 발생할 것이다. x$1=x$2 이나 "$1"="$2" 와 같이 쓰는 편이 좋다.

11.3. 산술 연산자

+

-

*

/

% (remainder)

11.4. 산술 관계 연산자

-lt (<)

-gt (>)

-le (<=)

-ge (>=)

-eq (==)

-ne (!=)

C 언어를 사용하는 사람이라면 이것이 무슨 의미인지 한눈에 알 수 있을 것이다. 그렇지 않더라도 이 글을 읽을 정도의 사람이라면 분명 무슨 말인지 알고 있을 테니 넘어가자. ^^

11.5. 쓸모있는 명령

이 부분은 키즈가 내용을 수정하여 다시 써 주었다.

이 곳의 몇몇 명령은 대부분 프로그래밍 언어에 포함되어 있는 것이다. 물론 여기 나오는 설명은 정말 눈물나게 간단한 것이므로, 이와 같은 명령에 대한 자세한 설명은 맨 페이지를 참고하자.

sed

sed는 비대화형(non-interactive) 모드의 라인 편집기이다. 또한 필터를 사용할 수 있어 셸 프로그래밍에서 아주 잘! 사용할 수 있는 도구이다. 먼저 간단히 예제를 살펴보자.

```
$sed 's/to_be_replaced/replaced/g' /tmp/dummy
```

Sed 는 /tmp/dummy 파일을 읽어들여 'to_be_replaced' 라는 문자열을 'replaced' 로 간단히 바꿔버린다. 이 결과는 그냥 화면에 출력될 뿐이므로, 파일로 저장하기 위해서는 위 명령의 끝에 '> 파일이름' 을 추가해야 할 것이다. 적당한 파일 이름을 넣어 주면 그 이름으로 이 결과가 저장될 것이다.

```
$sed 12, 18d /tmp/dummy
```

Sed 는 원본 파일에서 12에서 18줄 까지를 제외한 모든 줄을 보여준다.

awk

awk는 C 언어와 비슷한 방식으로 작동하는, 텍스트 처리 언어이다. awk는 패턴을 검색하여 같은 것을 찾아 처리한다는 점에서는 간단한 기능을 가진 것 처럼 보이지만, 사실 아주 정교하게 동작할 수 있다.

자, 다음 행을 /tmp/dummy 파일에 집어넣자.

"test123

test

tteesstt"

```
$awk '/test/ {print}' /tmp/dummy
```

test123

test

awk는 test가 포함된 줄을 화면에 출력하였다. 그러나 이것은 awk의 기능 중에서는 가장 단순한 부분에 속한다. 자세한 내용은 다른 문서를 참고하도록 하고, 여기에서는 한 가지만 더 살펴 보고 넘어가겠다.

```
$awk '/test/ {i=i+1} END {print i}' /tmp/dummy
```

3

다양한 패턴을 검색하기 위해 '-f file.awk'와 같은 식으로 파일 이름을 지정하고, 이 파일 안에 필요한 패턴을 모두 입력해 사용할 수도 있다.

grep

물론 grep에 대해서는 따로 설명할 필요가 없을 지도 모르겠다. 해당되는 문자열이 있는 행을 반환하는 grep 명령은 널리 쓰이고 있으니까. 하지만 grep은 다른 다양한 기능도 발휘할 수 있다.

```
$grep "look for this" /var/log/messages -c
```

12

"look for this"라는 문장이 /var/log/messages라는 파일 안에서 12번 발견되었다는 뜻이다.

wc

이 명령은 단어 수를 세어 주는 명령이다. 어느 외국 추리 소설에서, 단어당 5센트를 받는 작가의 이야기가 나오기도 했는데, 정말 원고료를 책정할 때 사용할 지도 모르겠다. 다음과 같은 내용을 dummy 파일로 저장하고 명령을 실행해 보자. **"bash introduction howto test file"**

```
$wc --words --lines --bytes /tmp/dummy
```

2 5 34 /tmp/dummy

sort

그러면 다음과 같은 내용이 들어 있는 임시 파일 /tmp/dummy를 만들어 보자.

"b

c

a"

```
$sort /tmp/dummy
```

위 명령의 결과로 다음과 같은 출력을 볼 수 있을 것이다.

a

b

c

또한 쉽지만은 않은 녀석으로 **bc**도 있다.

Bc는 명령 행에서 사용하는 계산기이다. 파일이나 직접 입력 혹은 리디렉션으로 입력받아 처리한다. -q 옵션을 사용하면 처음 시작할 때 나오는 메시지를 안 보고 넘어갈 수 있다.

```
$bc -q
```

1 == 5

0

0.05 == 0.05

1

5 != 5

0

2 ^ 8

256

sqrt(9)

3

while (i != 9) {

i = i + 1;

print i

}

123456789

quit

tput (initialize a terminal or query terminfo database)

12. 또 다른 스크립트들...

12.1. 예제 : 초간단 백업 스크립트-조금 발전된 버전

```
#!/bin/bash
SRCD="/home/"
TGTD="/var/backups/"
OF=home-$(date +%Y%m%d).tgz
tar -cZf $TGTD$OF $SRCD
```

12.2. 예제 : 파일이름 바꾸는 프로그램(renna)

```
#!/bin/sh
# renna: 여러 파일의 이름을 규칙에 따라 한번에 바꿀 수 있는 프로그램
# 페릭스 허드슨이 2000년 1월에 만들었다.


if [ $1 = p ]; then

prefix=$2 ; shift ; shift


if [$1 = ]; then
echo "no files given"
exit 0
fi

for file in $*
do
mv ${file} $prefix$file
done

exit 0
fi

if [ $1 = s ]; then
suffix=$2 ; shift ; shift

if [$1 = ]; then
```

```
echo "no files given"
exit 0
fi

for file in $*
do
mv ${file} $file$suffix
done

exit 0
fi

if [ $1 = r ]; then

shift

if [ $# -lt 3 ] ; then
echo "usage: renna r [expression] [replacement] files... "
exit 0
fi

OLD=$1 ; NEW=$2 ; shift ; shift

for file in $*
do
new=`echo ${file} | sed s/${OLD}/${NEW}/g`
mv ${file} $new
done
exit 0
fi

echo "usage;"
echo " renna p [prefix] files.."
echo " renna s [suffix] files.."
echo " renna r [expression] [replacement] files.."
exit 0
```

12.3. 간단하게 파일 이름 바꾸기

파일 이름을 바꾸기 위해서는 mv 명령을 사용할 수 있지만, 다음의 방법을 사용하여 파일 이름을 변경해 보자.

```
#!/bin/bash
# renames.sh
# 간단한 파일 이름 바꾸기 프로그램

criteria=$1
```

```
re_match=$2
replace=$3

for i in $( ls *$criteria* );
do
src=$i
tgt=$(echo $i | sed -e "s/$re_match/$replace/")
mv $src $tgt
done
```

12.4. 아침마다 신문 만화를 배경 화면으로 사용하기

이 프로그램은 GNOME에서 제대로 동작한다. 다른 환경에서 사용할 때에는 환경 설정파일을 새로 만드는 부분을 수정해야 할 것이다. 이 프로그램은 그 날의 날짜로 파일 이름을 만드는 신문 만화의 규칙성을 이용하여 파일을 받는 프로그램으로, crontab에 지정해 두면 더욱 편리하다.

```
#!/bin/bash
# bgimg.sh
# 그날의 날짜로 된 신문 만화를 받아온다. 아래의 주소 부분에는
# 상황에 따라 적어 준다.
filename="http://www.domain.com/pwd/"$(date +%Y%m%d)"07_0.jpg"
wget $filename

mv $(date +%Y%m%d)*07_0.jpg ./bgimages

imagename="/mypwd/bgimages/"$(date +%Y%m%d)".jpg"
confname="./.gnome/Background"
rm $confname
touch $confname

echo "[Default]" >> $confname
echo "color1=#5477a0" >> $confname
echo "color2=#5477a0" >> $confname
echo "simple=solid" >> $confname
echo "gradient=vertical" >> $confname
echo "wallpaperAlign=1" >> $confname
echo "Enabled=true" >> $confname

comm="wallpaper="$imagename

echo $comm >> $confname
```

13. 문제가 생겼을 때(디버깅 방법)

13.1. 프로그램 진행과정 보기

어떤 프로그램이 이해할 수 없는 동작을 할 경우, 프로그램의 첫 줄을 다음과 같이 수정한다.

```
#!/bin/bash -x
```

이런 옵션은 실행 과정에서의 정보를 하나하나 출력해 주어 디버그에 도움을 준다. 각 행이 실행될 때의 반응을 모두 볼 수 있어, 어느 부분에서 문제가 일어났는지를 확실하게 확인할 수 있다.

14. 이 문서에 대하여

이 문서에 추가하거나 정정하는 것, 혹은 이 문서를 보고 생각나는 것은 그냥 편하게 알려 줬으면 한다. 이런 내용은 모으고 모아서 가급적 빨리 문서에 추가해 두겠다.

.....라고 말한 사람이 1999년에 번역한 문서에서 토씨 하나 안 틀리고 2002년까지 두고 있단 말인가!!!!!-0- 그냥 기대하지 않는 것이 속 편하다. 어차피 Just for FUN이지, 의무로 하는 일도 아니지 않는가. 냅두자. -_-+

14.1. 이 문서에 대한 책임

이 문서는 어떤 종류의 책임도 지지 않는다. 다시 말하자면 이 문서를 따라하다가 모종의 불상사가 생기더라도 항의하지 말라는 말이다. 말은 편하게 하는군.... 이라는 생각이 들 지는 모르겠지만, 여기 있는 내용은 대체로 안전하니까 그냥 따라 해도 무방할 것이다. ^^

14.2. 번역

이탈리아 어 : 윌리엄 겔피 이곳을 참조 (http://web.tiscalinet.it/penguin_rules)

프랑스어 : 로랑 마뗄 (이 문서는 링크가 깨져있다.) (http://)

한국어 : 전혜진 http://kldp.org

스페인어 : 불명 http://www.insflug.org

더 많은 번역본이 있으리라 생각하지만, 내가 아는 것은 이 정도 선이다. 만약에 이 문서를 번역했다면, 그 사실을 알려주면 고맙겠다.

14.3. 여러분께 감사드립니다.

- 다른 여러 나라 말로 이 문서를 번역해 준 모든 분들
- 잘못된 부분을 정말 열심히 잡아 준 나산 허스트
- 수학적인 표현에 대해 추가 설명을 보내준 존 애보트
- **renna** 스크립트를 만들어 준 허드슨
- 내용 중 잘못된 것을 교정하여 보내주고 명령어 부분을 다시 써 주기까지 한 키스 반 덴 브뢰크
- BASH에 대한 조언을 해주고 코드를 테스트 해 준 마이크 핑크
- 루프 부분을 작성하는 데 도움을 주었던 피에시
- 몇 가지 에러에 대해 조언을 해 주었던 라이언
- 수정을 해주고 주석을 달아준 안드레아스 베크

14.4. 업뎃 과정....

부분적인 수정을 하고 번역 문서에 대해 새 정보를 넣었다.

유용한 명령에 대해 키스가 다시 써 준 원고를 추가하였다.

수정을 하고, 의견을 반영하였다.

문자열 쪽에 예제를 추가하였다

v0.8 버전 매기기를 중단하였다. 널럴하게.....

v0.7 수정을 더 하고 제대로 쓰지 않았던 예제 부분을 추가했다.

v0.6 부분 수정....

v0.5 리디렉션 부분을 추가하였다.

v0.4 문서를 linuxdoc.org 사이트로 자리를 바꾸어 올려 놓았다.

14.5. 추가 정보

배쉬 입문 http://org.laol.net/lamug/beforever/bashtut.htm

본 셸 프로그래밍 http://207.213.123.70/book/