8. 쉘 스크립트

Diff

쉘 스크립트란?

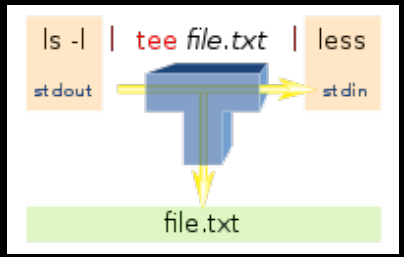
- 종종 똑같은 작업을 여러번 해야될 때가 있다.

  간단한 프로그램 실행부터 `XMODIFIERS= java -Xmx1024M -cp minecraft.jar net.minecraft.LauncherFrame`

  여러 명령어들을 입력해서 무언가 더 복잡한 것들을 해야될 때
- 이럴 때 여러 명령어들을 한 파일에 모아놓아 단일 명령으로 쓸 수 있게 만들어 놓은 것이 쉘 스크립트이다.

자주 쓰는 명령어들

- `echo` : 문자열을 출력한다. (`echo 'Hello, world!'`)
- `grep` : 문자열을 찾는 데 사용한다.
- `file 파일이름` : 파일 타입을 출력한다.
- `tee 파일이름` : 표준 출력을 파일에 입력한다.



- `basename 파일이름`, `dirname 파일이름`

쉘 스크립트를 찾아보자

- sparcs.org 서버에는 많은 장난감들이 존재한다.
- 이 장난감들은 `/SPARCS/bin`에 존재한다.
- 뭐가 있는지 한번 보자. `ls /SPARCS/bin`
- `*`를 이용하여 각각의 파일의 정보를 확인하자. `file /SPARCS/bin/*`
- 또는 `(cd /SPARCS/bin; file *)`
- "Shell script"라고 적혀있는 파일들만 골라보자.
- `(cd /SPARCS/bin; file * | grep -i "Shell script")`

쉘 스크립트는 어떻게 생겼을까?

- 한번 아무 장난감이나 골라서 보자. 기본적으로 스크립트는 모두 텍스트 파일이다.
- `vi /SPARCS/bin/seminar`

```
#!/bin/bash
# SPARCS seminar information manager
# Author:  Jaeho Shin <netj@sparcs.kaist.ac.kr>
# Created: 2003/04/08
  Version="1.0"
# $Id$
#
# TODO:
# - Timestamp XML files, and show them when listing or viewing.
#
# Changes:
# $Log$
#

# some configurations
SeminarRoot=/SPARCS/seminar
SeminarXMLSuffix=.xml
LibraryPath=/SPARCS/lib/seminar
XSLTList=$LibraryPath/list.xsl
XSLTView=$LibraryPath/view.xsl
XSLTWeb=$LibraryPath/web.xsl
XMLTemplate=$LibraryPath/template.xml
XMLDTD=$LibraryPath/seminar.dtd
IndexName=index.xml
IndexPath=$SeminarRoot/$IndexName
TermCharset=UTF-8
umask 022

## Begin of Subroutines ########################################################
seminar_exists() {
    local id="$1"
    [ -f "$SeminarRoot/$id$SeminarXMLSuffix" ]
}
```

1,1 꼭대기

쉘 스크립트 작성법 0

- 첫줄은 `#!<인터프리터>`와 같은 식이다.
- 인터프리터 : 소스 코드를 읽어서 실행해주는 녀석.
- 쉘 스크립트라면 `/bin/sh`를, 파이썬이라면 `/usr/bin/python`을 쓰면 된다.
- 그냥 `#!`만 있다면 `/bin/sh`로 해석해준다.
- 다른 `#`로 시작하느 줄들은 모두 주석.

한번 만들어보자! - Hello, world!

- `vi hello`
- 한줄 한줄 터미널에 명령을 적듯이 적으면 된다.
- 맨 첫줄에 `#!/bin/sh` 또는 `#!`를 적는 걸 잊지 말자.

```
1 #!/bin/sh
2
3 echo 'Hello, world!' # 안녕 세상아
4 echo 'Goodbye, world!' # 안녕 세상아
```

- `./hello` - 어라? 실행이 안 된다!
- 그 이유는 `ls -l`을 해 보면 알 수 있다.
- 실행 권한이 없어서 실행시키려 하면 에러가 난다.
- 실행 권한을 주자. `chmod +x hello` 한 뒤 다시 시도해보자.
- 실행 권한을 줄 수 없다면? `sh hello`처럼 직접 적으면 된다.

한번 만들어보자! - findscript

- 장난감 중 쉘 스크립트로 짜여진 것들을 보여주는 스크립트.

```
1 #!/bin/sh
2
3 cd /SPARCS/bin
4 file * | grep -i "Shell script"
```

- 저장하고 권한 주면 잘 돌아간다.
- 그런데, 파이썬 스크립트로 짜여진 장난감들을 보고 싶으면?
- nugu처럼 perl 스크립트로 짜여진 장난감들은?
- 뭔가 프로그래밍 언어를 정할 수 있으면 좋을텐데..

쉘 스크립트 작성법 1 - 인자

- 많은 프로그램들은 인자를 받아 어떻게 작동할 것인지를 결정한다.
- 예: `mv foo bar`
- 쉘 스크립트에서는 이러한 인자들을 받아서 처리할 수 있다.
- `$1` : 첫번째 인자, `$2` : 두번째 인자, ...
- `touch test; chmod +x test; vi test`
- `echo $1 $2 $3`
- `./test a b c`는 "a b c"를 출력한다.
- 한번 "Shell script" 대신 찾을 내용을 인자로 받게 `findscript`를 수정해 보자.
- `./findscript python` : 파일 타입에 'python'이 들어간 것들을 찾는다.
- `./findscript`하면 모든 장난감들이 출력된다. 대신 설명을 출력하게 하고 싶은데..

쉘 스크립트 작성법 2 - 조건문

- 쉘 스크립트에서 if문은 다음과 같다.

```
if [ 테스트할것 ]
then
  테스트할 것이 참이라면 할 것들
elif [ 테스트할것2 ]
  테스트할 것 2가 참이라면 할 것들
else
  테스트할 것이 거짓이라면 할 것들
fi
```

- 대괄호의 띄어쓰기, `then`를 다음 줄에 적어야 된다는 것에 명심하자!
- 다음 줄에 적기 싫다면 `;`를 이용해서 `if [ ~~ ]; then ~~` 처럼 적을 수 있다.

findscript v2

```
1 #!/bin/sh
2
3 if [ $1 ]
4 then
5     cd /SPARCS/bin
6     file * | grep -i "$1"
7 else
8     echo "사용법 : $0 종류"
9 fi
```

- `$0` : 스크립트를 실행한 경로.
- 이것도 역시 잘 돌아간다. `./findscript`, `./findscript perl`, `./findscript "shell script"`
- 그런데 `./findscript`라고만 쳤을 때 다시 `./findscript`를 쳐야 된다는 사실이 귀찮다.
- `./findscript`만 쳤을 때 사용자에게 종류를 입력할 수 있게 할 기회를 주자.

쉘 스크립트 작성법 3 - 변수

- 우선 사용자에게 입력받은 것을 저장할 수 있어야 한다.

  → 변수!

- 변수를 지정할 때에는 `변수명=값`으로 해 준다. (등호 양쪽에 공백이 없어야 됨)
- 변수에 접근할 때에는 `$변수명`으로 해 준다.
- 다시 `test`를 편집해보자.

```
foo=Hello
bar="Hello, world!"
echo $foo $bar
```

- `$변수명` 대신 `${변수명}`를 쓸 수도 있다.

미리 정의된 변수들

- 미리 정의되어있는 변수가 있다.
  - `$#` : 주어진 인자의 수
  - `$@`, `$*` : 입력받은 모든 인자
  - `$$` : 현재 프로세스의 ID
  - `$?` : 바로 전 실행한 명령어의 결과 코드.

쉘 스크립트 작성법 4 - 입력

- `read 변수명`
- 예를 들어 `read name; echo "Hello, ${name}!"`는 이름을 입력받은 뒤 그 사람에게 인사를 건넨다.
- 이제 다시 findscript 스크립트를 작성해보자.
- 만약 인자로 무언가 주어졌다면 그것을 타입으로 가지는 장난감들을,
- 아무 인자도 입력되지 않았다면 입력을 받게 한 뒤 그것을 타입으로 가지는 장난감들을 찾아보자.

findscript v3

```
1 #!/bin/sh
2
3 cd /SPARCS/bin
4
5 if [ $1 ]
6 then
7     file * | grep -i "$1"
8 else
9     echo "검색할 종류를 입력하세요."
10    read findType
11    file * | grep -i "$findType"
12 fi
```

- 이게 최선입니까?
- 확실해요?
- 줄 수를 줄여보자.

영 좋지 못한 방법

```
1 #!/bin/sh
2
3 cd /SPARCS/bin
4
5 findType=$1
6 if [ "$findType" ]
7 then
8     findType=$findType
9 else
10     echo "검색할 종류를 입력하세요."
11     read findType
12 fi
13 file * | grep -i "$findType"
```

- then을 비워둘수는 없어서 의미 없는 줄로 채웠다.
- 별로 바람직하지는 않은 것 같다. 심지어 줄 수도 늘어났다.
- 어떻게 해야 될까?

쉘 스크립트 작성법 5 - 조건문 테스트

- 다음과 같은 유용한 것들이 있다.
- 파일식 : 파일의 속성을 검사한다.

  `[ -f "file" ]` : file이 파일인지 아닌지 테스트한다.

  `[ -x "/usr/games/sl" ]` : /usr/games/sl이 실행파일인지 아닌지 테스트한다.
- 문자열식 : 문자열을 검사한다.

  `[ -z "$var" ]` : `$var`의 문자열 길이가 0인지 테스트한다.

  `[ -n "$var" ]` : `$var`의 문자열 길이가 0이 아닌지 테스트한다.

  `[ "$a" = "$b" ]` : `$a`와 `$b`가 같은지 확인한다.
- 숫자 비교를 위한 연산자들 : `-eq`(=), `-ne`(≠), `-ge`(≥), `-gt`(>), `-le`(≤), `-lt`(<)

findscript (마지막)

```
1 #!/bin/sh
2
3 cd /SPARCS/bin
4
5 findType=$1
6 if [ -z "$findType" ]
7 then
8     echo "검색할 종류를 입력하세요."
9     read findType
10 fi
11 file * | grep -i "$findType"
```

- 더 짧은 걸 좋아한다면

```
1 #!/bin/sh
2
3 cd /SPARCS/bin
4
5 f=$1
6 [ -z $f ] && echo "검색할 종류를 입력하세요." && read f
7 file * | grep -i "$f"
```

- `명령1 && 명령2`, `명령1 || 명령2` : 각각 '명령1이 참이라면 명령2를 실행', '명령1이 거짓이라면 명령2를 실행'이라는 뜻이다.

쉘 스크립트 작성법 6 - case문

```
case $변수 in
  패턴1) 명령;;
  패턴2) 명령;;
...
esac
```

- 패턴1부터 변수를 비교하면서 패턴에 해당하는 명령을 수행한다.
- `;;`로 명령들의 끝을 나타낸다.
- 패턴에는 `*`, `?`, 정규표현식, 문자열, 변수등을 쓸 수 있다.
- C 언어에서 `default:`는 `*)`로 쓸 수 있다.

쉘 스크립트 작성법 7 - 숫자의 사칙연산

- 두 방법이 가능하다.

```
x=6
y=9
let z=x*y
```

```
x=6
y=9
z=$(($x*$y))
```

- 주의사항 : `/bin/sh` 대신 `/bin/bash`를 써야 한다.

한번 만들어보자! - 계산기

- `touch calc; chmod +x calc; vi calc`

  (이것도 귀찮으면 쉘 스크립트로 만들어보자.)
- 스펙 : `./calc 4 + 5`를 입력하면 9가 실행된다. (수1 연산자 수2)
- 사칙연산은 `+`, `-`, `*`, `/`가 가능하다.
- 연산자가 부적절하다면 에러코드 42를 내고 종료하자. (`exit 42`)
- 힌트 : `"*"`와 `*`의 차이는 뭘까? (터미널에서, case 문에서)
- 곱셈은 `./calc 6 "*" 7`이나 `./calc 6 \* 7`처럼 한다.

쉘 스크립트 작성법 8 - 반복문

```
while [ 조건문 ]
do
  할 것들
done
```

```
until [ 조건문 ]
do
  할 것들
done
```

```
for 변수 in [ 목록 ]
do
  할 것들
done
```

한번 만들어보자! - addline

- 디렉토리를 하나 만들고 그 안에 잡다한 .txt 파일 여러개를 만들어보자. (`/home/differ/al`)
- 스펙 : `./addline`을 한 다음 한 줄을 입력받아, 스크립트가 실행중인 디렉토리의 모든 .txt 파일에 그 줄을 더한다.
- 힌트 : `for i in *.txt`
- 힌트 : `echo "$line" >> $i`

예제 : 구구단

```bash
#!/bin/bash

i=2
while [ $i -le 9 ]
do
    j=1
    while [ $j -le 9 ]
    do
        let k=i*j
        echo "$i * $j = $k"
        let j=j+1
    done
    let i=i+1
done
```

```bash
#!/bin/bash

for i in `seq 2 9`
do
    for j in `seq 1 9`
    do
        let k=i*j
        echo "$i * $j = $k"
    done
done
```

- `seq` : 숫자 목록을 출력한다.
- 대신 `for i in {2..9}`를 써도 된다.

쉘 스크립트 작성법 9 - 함수

```
함수이름()
{
  할 것들
}
```

- 함수 안에서는 함수 밖의 변수를 볼 수 있다.
- 나중에 일반 명령어처럼 함수이름을 적어주면 된다.
- 리턴은 `return 값`으로 해준 다음 나중에 `$?`로 확인할 수 있다.

  다만 이렇게 하면 정수만 반환이 가능하다. 외부에서 변수를 만들어 그 변수를 바꿔주는 식으로 값을 리턴해주자.

쉘 스크립트 작성법 10 - 배열

- `declare -a arr` (크기 지정 없이 배열을 선언한다.)
- `arr[1]=2`와 같이 접근하며, 불러올 때에는 `echo ${arr[1]}`과 같이 쓴다.

마지막으로 한번 만들어보자! - 맥주 99병

- 이 노래의 가사를 출력하면 된다. (http://99-bottles-of-beer.net/lyrics.html)
- 대소문자는 틀려도 상관 없다.
- 함수를 적절히 활용하자. ('99 bottles of beer' 같은 게 자주 나오는 것 같다.)
- 병 갯수가 1개일 때 "bottles"가 아니라 "bottle"이라는 것에 주의하자.
- `{99..0..-1}`로 99부터 0까지의 숫자를 얻을 수 있다.
- 출력 예는 `/home/differ/99.txt`에 있다.

```
#!/bin/bash

line=""

getText()
{
    bottles="bottles"
    if [ $i -eq 1 ]; then
        bottles="bottle"
    elif [ $i -eq 0 ]; then
        i="no"
    fi
    line="$i $bottles of beer"
}

for i in {99..0..-1}
do
    getText
    echo "$line on the wall, $line."
    if [ $i = "no" ]; then
        break
    fi
    let i=i-1
    getText
    echo "Take one down and pass it around, $line on the wall."
    echo
done
i=99
getText
echo "Go to store and buy some more, $line on the wall."
~
~
~
~
                                                              30,1          모두
```

참고 자료

- http://www.lug.or.kr/docs/LINUX/others/10-4.htm
- 작년 세미나 자료