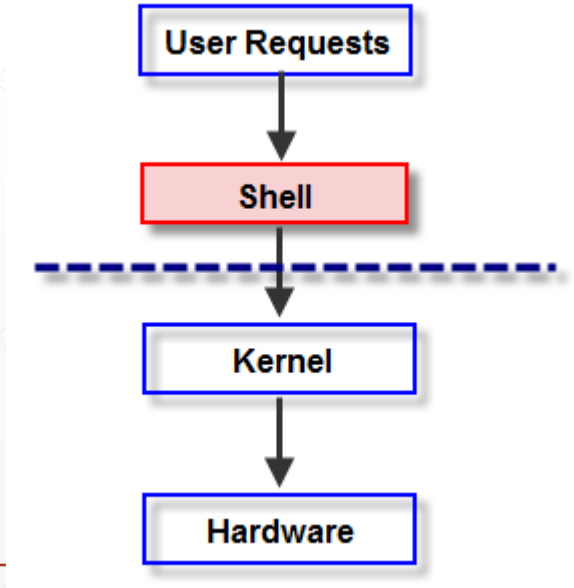
Shell Script

Linux Master

Shell 기본

Shell

- 사용자가 입력한 명령을 해석해 커널에 전달하거나, 커널의 처리 결과를 사용자에게 전달하는 역할



Bash Shell = CentOS 기본 Shell

- Alias 기능
- History 기능
- 연산 기능
- Job Control 기능
- 자동 이름 완성 기능
- 프롬프트 제어 기능
- 명령 편집 기능

환경 변수

- 변수
  - 변할 수 있는 값
  - 숫자, 문자, 문자열

- 환경 변수
  - 시스템의 속성을 기록한 변수
  - echo $환경변수

- 환경 변수의 확인
  - **echo $환경변수**

- 환경 변수의 변경
  - **export 환경변수=값**
  - 그 외의 환경변수
    - printenv 명령어를 실행하여 출력

주요 환경 변수

HOME	현재 사용자의 디렉터리	PATH	실행 파일을 찾는 디렉터리
LANG	기본 지원되는 언어	PWD	사용자의 현 작업디렉터리
TERM	로그인 터미널 타입	SHELL	로그인해서 사용하는 셸
USER	현재 사용자 이름	DISPLAY	X 디스플레이 이름
COLUMNS	현재 터미널의 컬럼 수	LINES	현재 터미널 라인 수
PS1	1차 명령프롬프트	PS2	2차 명령 프롬프트(대개 >)
BASH	Bash 셸의 경로	BASH_VERSION	Bash 버전
HISFILE	히스토리 파일의 경로	HISTSIZE	히스토리 파일에 저장 개수
HOSTNAME	호스트 이름	USERNAME	현재 사용자 이름
LOGNAME	로그인 이름	LS_COLORS	ls 명령어 확장자 색상 옵션
MAIL	메일을 보관하는 경로	OSTYPE	운영체제 타입

Shell Script 작성과 실행

- 확장자는 지정하지 않거나, 다른 것으로 지정해도 된다. (연습시엔 .sh 이용)

#!/bin/sh	**특별한 형태의 주석(#!)로 bash를 사용하겠다는 의미. 첫 행에 꼭 써야 한다.**
본문	**echo 명령어 등을 사용**
exit 0	**종료 코드를 반환한다.**

- **실행 방법 2가지**
  - 방법 1: sh 명령어로 실행
    - **sh** 파일명
  - 방법 2: 실행가능 속성으로 변경후 실행
    - **chmod +x 파일명**
    - **./파일명**

Shell Script 작성과 실행 예(name.sh)

>>> vi name.sh

```
#!/bin/sh
echo "사용자 이름: " $USERNAME
echo "호스트 이름: " $HOSTNAME
exit 0
```

실행 방법

- 방법 1
  - **sh** name.sh
- 방법 2
  - chmod +x name.sh
  - **./**name.sh

변수

Shell Script

변수

- 변수
  - 필요한 값을 계속 변경해 저장한다는 개념
- 변수의 기본
  - Shell Script에서는 변수를 사용하기 전에 미리 선언하지 않으며, **처음 변수에 값이 할당되면 자동으로 변수가 생성**된다.
  - **변수에 넣는 모든 값은 문자열**(String)으로 취급한다. 즉 숫자를 넣어도 문자로 취급한다.
  - **변수 이름은 대소문자를 구분**한다. 즉 $aa라는 변수 이름과 $AA라는 변수 이름은 다르다.
  - 변수를 대입할 때 **'=' 좌우에는 공백이 없어야** 한다.

대입 I

testval = Hello

testval=Hello

대입 II

testval=Yes Sir

testval="Yes Sir"

대입 III

testval=7+5

변수에 값을 대입

- 오류! '=' 앞 뒤에 공백이 있음
- 오류! 값의 공백은 ""로 묶어야
- 오류! 정상이지만 "7+5"라는 문자열로 인식
  - 확인: echo $testval

변수

변수의 (입력과) 출력

- **$라는 문자가 들어간 글자를 출력하려면 ' '로 묶어주거나 앞에 '₩'를 붙여야 한다.**
- **" "로 변수를 묶어도 되고, 묶지 않아도 된다.**

숫자 계산

- 변수에 넣은 값은 모두 문자열로 취급한다.
- **연산**을 하려면 **'expr' 키워드**를 사용한다.
- 단 **수식과 함께 반드시 `(역따옴표)로 묶어야** 한다.
- 수식에 **괄호**를 사용하려면 반드시 **₩(역슬래시)**를 붙여야 한다.
- 다른 기호와 달리 ***(곱하기)**도 예외적으로 **₩(역슬래시)**를 붙여야 한다.

```
>>> var1.sh
#!/bin/sh
myvar="Hi Woo"
echo $myvar
echo "$myvar"
echo '$myvar'
echo ₩$myvar
echo 값 입력 :
read myvar
echo '$myvar' = $myvar
exit 0
```

변수의 입력과 출력

- 3행: 'Hi Woo'라는 정상값을 출력
- 4행: 3행과 동일한 효과
- 5행: '$myvar' 글자를 출력
- 6행: ₩$는 $를 글자로 취급
- 8행: 변수 myvar에 키보드로 값을 입력한다.

```
>>> numcalc.sh
#!/bin/sh
num1=100
num2=$num1+200
echo $num2
num3=`expr $num1 + 200`
echo $num3
num4=`expr ₩( $num1 + 200 ₩) / 10 ₩* 2`
echo $num4
exit 0
```

숫자 계산

- 3행: 문자열로 취급함
  - 모두 붙여서 써야 한다.
- 5행: 숫자로 취급해서 계산함
  - 각 단어마다 띄어쓰기를 해야
- 7행: 괄호와 * 앞에 역슬래시(₩)

>>> parvar.sh

```
#!/bin/sh

echo "실행파일 이름은 <$0>이다"

echo "첫번째 파라미터는 <$1>이고, 두번째
   파라미터는 <$2>다"

echo "전체 파라미터는 <$*>다"

exit 0
```

- **파라미터 변수는 $0, $1, $2 등의 형태를 갖는다.**
- **실행하는 명령의 부분 하나하나를 변수로 지정한다**는 의미이다.
- **명령 전체의 파라미터 변수는 $***로 표현한다.

파라미터 변수

- 실행 시
- sh parvar.sh 값1 값2 값3

if문 & case문

Shell Script

if문

기본 if문

```
if [ 조건 ]
then
    참일 경우 실행
fi
```

- [조건] 각 단어에는 모두 공백 있어야

기본 if~else문

```
if [ 조건 ]
then
    참일 경우 실행
else
    거짓일 경우 실행
fi
```

- 참일 경우와 거짓일 경우를 구분

```
>>> if1.sh
#!/bin/sh
if [ "woo" = "woo" ]
then
    echo "참입니다"
fi
exit 0
```

기본 if문

- 2행
  - [ ] 사이에는 참과 거짓을 구분하는 조건식이 들어간다.
  - =은 문자열이 같은지를 비교
  - !=는 문자열이 같지 않은지를 비교
  - 조건식이 참이므로 4행을 실행한다.

```
>>> if2.sh
#!/bin/sh
if [ "woo" = "woo" ]
then
    echo "참입니다"
else
    echo "거짓입니다"
fi
exit 0
```

if~else문

- 중복 if문을 위해서 else if가 합쳐진 elif 구문도 사용할 수 있다.

문자열 비교 연산자

문자열 비교	결과
“문자열1”=“문자열2”	두 문자열이 같으면 참
“문자열1”!=“문자열2”	두 문자열이 같지 않으면 참
-n “문자열”	문자열이 NULL(빈 문자열)이 아니면 참
-z “문자열”	문자열이 NULL(빈 문자열)이면 참

산술 비교 연산자

산술 비교	결과
수식1 –eq 수식2	두 수식(또는 변수)이 같으면 참
수식1 –ne 수식2	두 수식(또는 변수)이 같지 않으면 참
수식1 –gt 수식2	수식1이 크다면 참
수식1 –ge 수식2	수식1이 크거나 같으면 참
수식1 –lt 수식2	수식1이 작으면 참
수식1 –le 수식2	수식1이 작거나 같으면 참
!수식	수식이 거짓이라면 참

```
>>> if3.sh
#!/bin/sh
if [ 100 -eq 200 ]
then
    echo "100과 200은 같다."
else
    echo "100과 200은 다르다."
fi
exit 0
```

조건문에 들어가는 비교 연산자

- 문자열 비교 연산자
- 산술 비교 연산자

파일과 관련된 조건

파일 조건	결과
-d 파일이름	파일이 디렉터리면 참
-e 파일이름	파일이 존재하면 참
-f 파일이름	파일이 일반 파일이면 참
-g 파일이름	파일에 set-group-id가 설정되면 참
-r 파일이름	파일이 읽기 가능이면 참
-s 파일이름	파일 크기가 0이 아니면 참
-u 파일이름	파일에 set-user-id가 설정되면 참
-w 파일이름	파일이 쓰기 가능 상태이면 참
-x 파일이름	파일이 실행 가능 상태이면 참

```
>>> if4.sh
#!/bin/sh
fname=/lib/systemd/system/httpd.service
if [ -f $fname ]
then
    head -5 $fname
else
    echo "웹 서버가 설치되지 않았습니다."
fi
exit 0
```

파일과 관련된 조건

- 2행: fname 변수에 /lib/systemd/system/httpd.service 저장
- 3행: httpd.service 파일이 일반 파일이면 참이므로 5행이 실행, 그렇지 않으면 거짓이므로 7행이 실행
- 5행: fname에 들어 있는 파일의 앞 5줄을 출력

```
>>> case1.sh
#!/bin/sh
case "$1" in
    start)
        echo "시작~~";;
    stop)
        echo "중지~~";;
    restart)
        echo "다시 시작~~";;
    *)
        echo "뭔지 모름~~";;
esac
exit 0
```

- **다중 분기**: if문을 계속 중복해서 사용해야 하는 경우

case~esac문

- 2행: 첫번째 파라미터 변수인 $1의 값에 따라 3행, 5행, 7행, 9행으로 분기함
- 4행: 3행에서 start)일 경우에 실행. 주의할 점은 맨 뒤에 세미콜론을 2개 붙여야 함
- 9행: 그 외의 것들
- 11행: case문 종료를 표시

```
>>> case2.sh
#!/bin/sh
echo "리눅스가 재미있나요? (yes / no)"
read answer
case $answer in
    yes | y | Y | Yes | YES)
        echo "다행입니다"
        echo "더욱 열심히 하세요 ^^";;
    [nN]*)
        echo "안타깝네요. ㅠㅠ";;
    *)
        echo "yes 아니면 no만 입력했어야죠"
        exit 1;;
esac
exit 0
```

case~esac문

- 3행: answer 변수에 입력한 값을 받는다.
- 5행: 입력된 값이 yes면 6~7행 실행
- 6행: **실행할 구문이 더 있으므로 ;;를 붙이지 않는 점에 주의**
- 7행: 실행할 구문이 없으므로 ;; 추가
- 8행: n 또는 N이 들어가는 모든 단어
- 12행: 정상적인 종료가 아니므로 exit 1로 종료(필수는 아님)

AND, OR 관계 연산자

- 조건문에서는 and와 or의 의미를 갖는 관계연산자를 사용할 수 있다.
- **and: -a 또는 &&**
- **or: -o 또는 ||**
- -a나 -o는 텍스트문 [ ] 안에서 사용할 수 있는데, 이때 괄호 등의 특수 문자 앞에는 ₩(역슬래시)를 붙여줘야 한다.

```
>>> andor.sh
#!/bin/sh
echo "보고 싶은 파일명을 입력하세요"
read fname
if [ -f $fname ] && [ -s $fname ] ; then
    head -5 $fname
else
    echo "파일이 없거나, 크기가 0입니다"
fi
exit 0
```

AND, OR 관계 연산자

- **입력한 파일 이름이 일반 파일(-f)이고, 크기가 0이 아니라면(-s)라면 5행이 실행**
- then 구문은 다음 줄에 작성해도 되며, 세미콜론 이후에 작성해도 됨
- **세미콜론은 앞뒤 구문을 행으로 구분해주는 기능**

반복문

Shell Script

반복문

기본 for~in문

for 변수 in 값1 값2 값3

do

반복할 문자

done

while문

- while문은 조건식이 참인 동안에 계속 반복되는 특성을 갖는다.
- 조건식 위치에 [1] 또는 [:]가 오면 항상 참이다.

```
>>> forin1.sh
#!/bin/sh
hap=0
for i in 1 2 3 4 5 6 7 8 9 10
do
    hap=`expr $hap + $i `
done
echo "1부터 10까지의 합: "$hap
exit 0
```

for~in문

2행: 합계 누적할 변수를 0으로 초기화

3행: i 변수에 1~10까지를 반복해 넣으면서 5행을 10회 실행

5행: hap에 i 변수의 값을 누적

```
>>> forin2.sh
#!/bin/sh
for fname in $(ls *.sh)
do
    echo "~~~~~~$fname~~~~~~"
    head -3 $fname
done
exit 0
```

for~in문

- 2행: fname 변수에 ls *.sh의 실행 결과를 하나씩 넣어서 4~5행을 반복
- 4행: 파일 이름을 출력
- 5행: 파일의 앞 3줄을 출력

```
>>> while1.sh
#!/bin/sh
while [ 1 ]
do
    echo "CentOS 7"
done
exit 0
```

while문

- 2행: 조건식 위치에 [1] 또는 [:]가 오면 항상 참이다.

```
>>> while2.sh
#!/bin/sh
hap=0
i=1
while [ $i -le 10 ]
do
    hap=`expr $hap + $i`
    i=`expr $i + 1`
done
echo "1부터 10까지의 합 : "$hap
exit 0
```

while문

1부터 10까지의 합계

- 2행: 누적할 hap 변수를 초기화
- 3행: 1에서 10까지 증가할 i 변수를 선언
- 4행: i가 10보다 작거나 같으면 6~7행을 실행
- 6행: hap에 i의 값을 누적해 저장
- 7행: i 변수 값을 1씩 증가

```
>>> while3.sh
#!/bin/sh
echo "비밀번호를 입력하세요"
read mypass
while [ $mypass != "1234" ]
do
    echo "틀렸음. 다시 입력하세요"
    read mypass
done
echo "통과~~"
exit 0
```

while문

비밀번호를 입력받고, 맞을 때까지 계속 입력받는 스크립트

- mypass 변수에 값을 입력받음
- 변수 값이 1234가 아니면 6~7행을 실행. 맞으면 while문 종료
- 다시 mypass 변수에 값을 입력받음

반복문

until문

- while문과 용도는 같다.
- **until문은 조건식이 참일 때까지 계속 반복한다.**
- while2.sh를 동일용도의 until문으로 바꾸려면 변경
  - 4행: until [$i –gt 10]

break, continue, exit, return

- break
  - 주로 **반복문을 종료**할 때 사용한다.
- continue
  - **반복문의 조건식**으로 돌아가게 한다.
- exit
  - 해당 프로그램을 **완전히 종료**한다.
- return
  - 함수 안에서 사용한다.
  - **함수를 호출한 곳으로 돌아가게** 한다.

```
>>> bce.sh
#!/bin/sh
echo "무한반복 입력을 시작합니다. (b: break, c: continue, e: exit)"
while [ 1 ] ; do
  read input
  case $input in
    b | B)
      break;;
    c | C)
      echo "continue를 누르면 while의 조건으로 돌아감"
      continue;;
    e | E)
      echo "exit를 누르면 프로그램(함수)를 완전히 종료함"
      exit 1;;
  esac
done
echo "break를 누르면 while을 빠져나와 지금 이 문장이 출력됨"
exit 0
```

break, continue, exit, return

- 3행: 무한 반복
- 5행: 4행에 입력 값에 따라 분기
- 6~7행: break 실행되어 16행 실행
- 8~10행: continue 실행되어 3행의 조건식으로 돌아감
- 11~13행: exit 실행되어 프로그램 종료

기타 알아둘 내용

Shell Script

기타 알아둘 내용

사용자 정의 함수

- 사용자가 직접 함수를 작성하고 호출할 수 있다.
- 형식
  - **함수이름 ( ) {** → 함수를 정의
    - 내용들 …
  - **}**
  - **함수이름** → 함수를 호출

함수의 파라미터 사용

- 함수의 파라미터, 즉 인자를 사용하려면 함수를 호출할 때 뒤에 파라미터를 붙여서 호출한다.
- 함수 안에서는 $1, $2, …로 사용한다.
- 형식
  - **함수이름( ) {**
    - **$1, $2 …** 등을 사용
    - **}**
  - **함수이름 파라미터1 파라미터2 …**

```
>>> func1.sh
#!/bin/sh
myFunction () {
    echo "함수 안으로 들어왔음"
    return
}
echo "프로그램을 시작합니다"
myFunction
echo "프로그램을 종료합니다"
exit 0
```

사용자 정의 함수

- 2~5행: 함수를 정의. 단, 6행에서 호출되기 전에는 실행되지 않음.
- 6행: 프로그램 시작
- 7행: 함수 이름을 사용하면 함수가 호출

```
>>> func2.sh
#!/bin/sh
hap () {
    echo `expr $1 + $2`
}
echo "10 더하기 20을 실행합니다"
hap 10 20
exit 0
```

함수의 파라미터 사용

- 3행: 넘겨받은 파라미터 $1과 $2를 더한 값을 출력
- 6행: 호출할 때 함수 이름에 넘겨줄 파라미터를 공백으로 분리해서 차례로 적음

```
>>> eval.sh
#!/bin/sh
str="ls -l anaconda-ks.cfg"
echo $str
eval $str
exit 0
```

eval

문자열을 명령문으로 인식하고 실행

- 3행: str 변수값인 ls –l ... 글자를 그대로 출력
- 4행: str 변수값인 ls –l ...를 명령어로 인식하고 실행

```
>>> exp1.sh
#!/bin/sh
echo $var1
echo $var2
exit 0
```

export

외부 변수로 선언한다.

즉 선언한 변수를 다른 프로그램에서도 사용할 수 있게 한다.

- 2~3 행: var1과 var2 변수를 출력

```
>>> exp2.sh
#!/bin/sh
var1="지역 변수"
export var2="외부 변수"
sh exp1.sh
exit 0
```

export

- 2행: var1에 값을 넣는다. 일반변수(지역변수)이므로 현재에서만 사용. 즉, exp1.sh의 var1과 우연히 이름만 같을 뿐 다른 변수
- 3행: var2를 외부변수로 선언하고 값을 입력. 외부에서도 사용 가능
- 4행: exp1.sh 실행

```
>>> printf.sh
#!/bin/sh
var1=100.5
var2="재미있는 리눅스~~~"
printf "%5.2f ₩n₩n ₩t %s ₩n" $var1 "$var2"
exit
```

printf

C 언어의 printf( )함수와 비슷하게 형식을 지정해서 출력할 수 있다.

- 3행: 공백이 있으므로 “”로 묶어줘야
- 4행
  - %5.2f는 총 5자리, 소수점 아래 2자리까지 출력
  - ₩n은 1줄을 넘기는 개행문자
  - ₩t는 Tab 문자
  - %s는 문자열 출력
  - 주의: $var2 경우, 값 중간에 공백이 있으므로 변수이름을 “”로 묶어야

기타

set과 $(명령어)

- 리눅스 명령어를 결과로 사용하려면 **$(명령어)** 형식을 사용해야 한다.
- 결과를 파라미터로 사용하고자 할 때는 **set**와 함께 사용한다.

shift

- 파라미터 변수를 왼쪽으로 한 단계씩 아래로 쉬프트(이동)시킨다.
- 모든 파라미터 변수를 출력하고 싶거나, 특히 10개가 넘는 파라미터 변수에 접근할 때 사용한다.
- 단, $0 파라미터 변수는 변경되지 않는다.

```
>>> set.sh
#!/bin/sh
echo "오늘 날짜는 $(date)입니다"
set $(date)
echo "오늘은 $4 요일입니다"
exit 0
```

set과 $(명령어)

- 2행: $(date)는 date 명령어를 실행한 결과를 보여줌
- 3행: $(date)의 결과가 $1, $2, $3 ... 등의 파라미터 변수에 저장
- 4행: 4번째 파라미터인 요일이 출력

```
>>> shift1.sh
#!/bin/sh
myfunc () {
    echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11
}
myfunc AAA BBB CCC DDD EEE FFF GGG HHH III JJJ
   KKK
exit 0
```

shift

- 3행: 전달받은 파라미터 11개 출력
  - 그런데 $10과 $11은 예상과 다름
  - 이유는 $10을 $1에 0문자를 붙인 것으로 핵석하기 때문
- 5행: 11개 파라미터로 MYFUNC 함수를 호출

```
>>> shift2.sh
#!/bin/sh
myfunc () {
    str=""
    while [ "$1" != "" ] ; do
      str="$str $1"
      shift
    done
    echo $str
}
myfunc AAA BBB CCC DDD EEE FFF GGG HHH III JJJ
    KKK
exit 0
```

shift

- 3행: 결과를 누적할 str 변수를 초기화
- 4행: $1 파라미터가 비어 있지 않은 동안에 반복 실행
- 5행: str 변수에 $1을 추가
- 6행: 전체 파라미터를 왼쪽으로 쉬프트
- 8행: while문이 끝나면 누적한 str 변수를 출력