Shell Programming

Shell Script

❖ 개요

- ✓ 셸 스크립트는 Unix나 Linux 또는 POSIX(Portable Operating System Interface)를 지원하는 운영체제인 macOS 등에서 일반적으로 사용하는 명령어들과 if, for와 같은 프로그래밍적인 요소로 이루어진 인터프리터 기반의 스크립트 언어
- ✓ 리눅스 나 유닉스가 설치되어 있는 곳이라면 어디서든 사용 가능
- ✓ 셸 스크립트도 일반적인 프로그래밍 언어와 비슷하게 변수, 반복문, 제어문 등을 사용할 수 있으며 별도로 컴파일하지 않고 텍스트 파일 형태로 셸에서 바로 실행할 수도 있어서 셸 스크립트는 주로 vi 에디터나 nano, gedit 등 으로 작성하는 경우가 많음
- ✓ 셸 스크립트를 공부하면 좋은 이유는 리눅스의 많은 부분이 셸 스크립트로 작성되었기 때문인데 GRUB의 설정 파일인 /boot/gmb.d/ 폴더 아래의 파일들도 셸 스크립트 문법을 사용

Shell Programming

- 작성
  - ✓ 자주 사용하는 에디터를 이용해서 작성
  - ✓ 셸 스크립트 파일의 확장명을 지정하지 않거나 다른 것으로 지정해도 되지만 사용자가 작성한 셸 스크립트 파일은 되도록 확장명을 sh로 지정하는 것을 권장하는데 파일 이름만으로 이 파일이 셸 스크립트 파일인지 알 수 있기 때문
  - ✓ 셸 스크립트 파일을 만들 때는 일반적으로 sh 확장자를 사용해서 셸 스크립트 파일임을 명시하고 시작 시 #!/bin/bash를 붙여 해당 파일이 셸 스크립트라는 것을 알려주며 실행하고자 하는 명령어들을 입력하고 저장한 후 빠져나오면 됨
  - ✓ name.sh 파일 작성

```
#!/bin/bash
echo "사용자 이름: " $USER
echo "홈 디렉터리: " $HOME
exit 0
```

    - 1행: 특별한 형태의 주석 (#!)으로 bash를 사용하겠다는 의미
    - 2행: echo 명령은 화면에 출력하는 명령으로 먼저 "사용자 이름 :"이라는 글자를 출력하고 옆에는 $USER라는 환경 변수의 내용을 출력
    - 4행: 종료 코드를 반환하게 해주는데 만약 다른 스크립트에서 이 스크립트를 호출한 후 제대로 실행되었는지 확인하려면 적절한 종료 코드를 반환하는 것이 중요한데 이 행은 실제 스크립트 실행과 무관하지만 셸 스크립트는 실행 중간에 문제가 생겨도 무조건 성공했다는 메시지를 반환하기 때문에 마지막 행에서 직접 성공인지 실패인지를 반환하는 것이 좋은데 0은 성공을 의미

Shell Programming

- ❖ 실행
  - ✓ sh 명령으로 실행
    - sh 스크립트파일 명령으로 실행할 수 있는데 셸 스크립트 파일의 속성을 변경할 필요가 없다는 장점이 있음
    - 이 경우 sh 대신에 bash 와 같은 셸 이름을 변경해서 사용해야 하는 경우도 있음
    - sh name.sh

      사용자 이름: adam

      홈 디렉터리: /home/adam

Shell Programming

- ❖ 실행
  - ✓ 실행 가능 속성으로 변경 후 실행
    - 셸 스크립트 파일에 실행 권한을 주고 직접 셸 스크립트를 실행하는 방법
    - chmod 명령어를 이용하여 생성한 셸 스크립트파일에 실행 권한(+x)을 주고 스크립트가 위치한 경로의 셸 스크립트 파일을 호출하면 실행 됨
    - ls -l name.sh

      -rw-rw-r-- 1 adam adam 83 May 20 00:20 name.sh
    - chmod +x name.sh
    - ls -l name.sh

      -rwxrwxr-x 1 adam adam 83 May 20 00:20 name.sh
    - ./name.sh

      사용자 이름: adam

      홈 디렉터리: /home/adam

Shell Programming

- ❖ 실행
  - ✓ 명령어와 함께 프롬프트에서 바로 실행
    - echo "사용자 이름: " $USER; echo "홈 디렉토리: " $HOME

Shell Programming

- ❖ 변수
  - ✓ 개요
    - 셸 스크립트에서는 변수를 사용하기 전에 미리 선언하지 않고 처음 변수에 값이 할당되면 자동으로 변수가 생성
    - 변수 이름은 대소문자를 구분
    - 변수에 값을 대입할 때 = 좌우에는 공백이 없어야 한함
    - 변수 타입은 필요하지 않으며 문자열을 저장하면 문자열이 저장되고 숫자를 저장하면 숫자가 저장됨
    - 문자열 사이에 공백이 있으면 "" 로 묶어야 함

Shell Programming

- ❖ 변수
  - ✓ 기본 변수 선언
    - 문자열 출력: 문자열을 출력할 때는 echo 뒤에 $뒤에 변수 이름을 입력

```
#!/bin/bash
language="Korean"
echo "I can speak $language"
```

    - 명령 수행: 디렉토리 생성

```
#!/bin/bash
language="Korea Amearica China"
mkdir $language
```

Shell Programming

- ❖ 변수
  - ✓ 숫자 계산
    - 변수에 넣은 값은 모두 문자열로 취급하는데 만약 변수에 들어 있는 값에 +. -, *, / 등의 연산을 하려면 expr 키워드를 사용해야 하고 수식과 함께 꼭 키보드 숫자 1 왼쪽에 있는 백틱으로 묶어줘야 함
    - 수식에 괄호를 사용하려면 그 앞에 꼭 역슬래시(₩)를 붙여야 함
    - /와 달리 곱하기 (*) 기호는 예외적으로 앞에 역슬래시(₩)를 붙여야 함

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

Shell Programming

- ❖ 변수
  - ✓ 변수의 종류
    - ● 위치 매개변수
      - ◆ 위치 매개변수는 $0, $1, $2 등의 형태를 갖는데 명령어에 포함된 모든 단어를 각자 $변수 형식으로 지정
      - ◆ sh myshell.sh hello world 명령을 실행한다고 가정하면 파라미터 변수는 다음과 같이 지정할 수 있음

        sh myshell.sh hello world

        $0 $1 $2
      - ◆ $*: 전체 인자 값으로 위의 경우 hello world
      - ◆ $@: 전체 인자 값으로 쌍따옴표 변수를 감싸면 다른 결과
      - ◆ $#: 매개변수의 총 개수

Shell Programming

- ❖ 변수
  - ✓ 변수의 종류
    - ● 위치 매개변수
      - ◆ myshell.sh 파일을 아래와 같이 작성

```
#!/bin/sh
echo "실행파일 이름은 <$0>이다"
echo "첫번째 파라미터는 <$1>이고, 두번째 파라미터는 <$2>다"
echo "전체 파라미터는 <$*> 다"
exit 0
```

      - ◆ sh myshell.sh

```
실행파일 이름은 <myshell.sh>이다
첫번째 파라미터는 <Hello>이고, 두번째 파라미터는 <World>다
전체 파라미터는 <Hello World> 다
```

Shell Programming

- ❖ 변수
  - ✓ 변수의 종류
    - ● 함수
      - ◆ 개요
        - ▪ 특정 동작이나 목적을 위해 만들어진 것이 함수
        - ▪ 함수를 사용하는 이유는 스크립트를 재사용하기 위함
        - ▪ 스크립트를 재사용함으로써 스크립트의 줄 수를 줄여주고 좀 더 효율적으로 스크립트를 만들 수 있음
        - ▪ 함수를 만들 때는 function이라는 단어로 시작하며 어떤 동작을 위한 함수인지를 나타내는 함수명 뒤에 ( )를 붙여준 후 함수의 시작과 끝을 알리는 { } 를 열고 닫음
        - ▪ 중괄호 안에 필요한 특정 동작들을 나열
        - ▪ 함수를 사용할 때는 함수에 필요한 파라미터와 함께 해당 함수를 호출

Shell Programming

- ❖ 변수
  - ✓ 변수의 종류
    - ● 함수
      - ◆ 문자열을 출력해주는 함수: sh 대신에 bash 명령이나 권한을 수정해서 실행

```
#!/bin/bash

function print() {
        echo $1
}

print "I can speak Korean"
```

Shell Programming

- ❖ 변수
  - ✓ 변수의 종류
    - ● 전역 변수
      - ◆ 함수 외부에 선언해서 함수 내부 와 외부 모두에서 사용할 수 있는 변수

```
#!/bin/bash

language="Korean"

function print() {
        echo "I can speak $language"
}

print
```

Shell Programming

- ❖ 변수
  - ✓ 변수의 종류
    - ● 지역 변수
      - ◆ 함수 내부에 선언해서 함수 내부에서만 사용할 수 있는 변수: 아래 내용을 실행하면 learn_language는 지역 변수이므로 세번째 문장에서 출력안됨

```
#!/bin/bash
language="Korean"
function learn() {
        local learn_language="English"
        echo "I can speak $learn_language"
}
function print() {
        echo "I can speak $1"
}
learn
print $language
print $learn_language
```

Shell Programming

- ❖ 변수
  - ✓ 변수의 종류
    - ● 환경 변수: 시스템을 위해 미리 시스템에서 사용하고 있는 변수
      - ◆ HOME: 사용자의 홈 디렉토리
      - ◆ PATH: 실행 파일을 찾을 디렉토리 경로
      - ◆ LANG: 프로그램 사용 시 기본으로 지원되는 언어
      - ◆ PWD: 현재 작업 중인 디렉토리
      - ◆ SHELL: 현재 사용 중 인 셸
      - ◆ USER 또는 USERNAME: 사용자 이름
      - ◆ HISTFILE: history 파일 경로
      - ◆ HISTFILESIZE: history 파일 크기
      - ◆ HISTSIZE: history에 저장되는 개수
      - ◆ HOSTNAME: 호스트 이름
      - ◆ OSTYPE: 운영체제 종류

Shell Programming

- 변수
  - ✓ 변수의 종류
    - 특수 매개변수: 현재 실행 중인 스크립트나 명령어의 프로세스 ID를 확인하거나 바로 앞에서 실행한 명령어나 함수 또는 스크립트 실행이 정상적으로 수행되었는지 여부를 확인할 수 있는 변수
      - $$: 현재 스크립트 또는 명령어의 PID
      - $?: 최근에 실행된 명령어, 함수, 스크립트의 종료 상태
      - $!: 최근에 실행된 백그라운드(비동기) 명령의 PID
      - $-: 현재 옵션 플래그

Shell Programming

- ❖ 변수
  - ✓ 변수의 종류
    - ● 특수 매개변수
      - ◆ 실습
        - ▪ echo $$

          88485
        - ▪ 최근에 수행한 명령어의 종료 코드: echo $?

          0
        - ▪ 명령어 실패: ls abc

          ls: cannot access 'abc': No such file or directory
        - ▪ echo $?

          2

Shell Programming

- ❖ 변수
  - ✓ 매개변수 확장
    - ● 기본 변수 사용
      - ◆ &{변수}: 뒤에 오는 문자열 과 구분
        - ▪ $ AUTH_URL="www.example.com/"
        - ▪ $ echo "http://$AUTH_URLlogin.html"
          http://.html
        - ▪ $ echo "http://${AUTH_URL}login.html"
          http://www.example.com/login.html

Shell Programming

- ❖ 변수
  - ✓ 매개변수 확장
    - ● 변수 초기화를 위한 확장 변경자
      - ◆ 값이 없는 경우 치환을 위한 확장자: 콜론(:)을 생략하고 ${var-default}처럼 사용하면 변수가 선언은 되었으나 빈 문자열("")일 때를 값이 있는 상태로 취급하는데 보통은 빈 문자열도 값이 없는 것으로 처리하는 경우가 많으므로 콜론(:)을 붙여서 쓰는 것이 훨씬 안전
        - ${var:-default} default 반환 (변수 유지) 원래 var 값 반환
        - ${var:=default} var에 default 실제 할당 및 반환 원래 var 값 반환
        - ${var:?message} message 출력 후 스크립트 종료 원래 var 값 반환
        - ${var:+alternative} 아무것도 반환 안 함 alternative 반환 (변수 유지)

Shell Programming

- ❖ 변수
  - ✓ 매개변수 확장
    - ● 변수 초기화를 위한 확장 변경자
      - ◆ 문자열 추출 및 길이 확인 옵션: 문자열의 길이를 구하거나 특정 위치의 문자열을 잘라낼 때 사용 - var="HelloBash"
        - ▪ ${#var}: 변수에 저장된 문자열의 **길이(글자 수)**를 반환 ${#var} → 9
        - ▪ ${var:offset}: offset 위치(0부터 시작)부터 끝까지 잘라냄 ${var:5} → "Bash"
        - ▪ ${var:offset:length}:offset 위치부터 length만큼 잘라냄 ${var:0:5} → "Hello"
        - ▪ ${var: -length}: 뒤에서부터 length만큼 잘라냄 (마이너스 앞 공백 필수)
          ${var: -4} → "Bash"

Shell Programming

- ❖ 변수
  - ✓ 매개변수 확장
    - ● 변수 초기화를 위한 확장 변경자
      - ◆ 실습
        - $ OS_TYPE=redhat
        - $ echo ${OS_TYPE:-ubuntu}

        redhat
        - $ echo ${OS_TYPE-ubuntu}

        redhat
        - $ unset OS_TYPE
        - $ echo ${OS_TYPE:-ubuntu}

        ubuntu
        - $ echo ${OS_TYPE-ubuntu}

        ubuntu
        - $ OS_TYPE=""
        - $ echo ${OS_TYPE-ubuntu}
        - $ echo ${OS_TYPE:-ubuntu}

        ubuntu

Shell Programming

- ❖ 변수
  - ✓ 매개변수 확장
    - ● 변수 초기화를 위한 확장 변경자
      - ◆ 실습
        - $ OS_TYPE="Redhat Ubuntu Fedora Debian"
        - $ echo ${OS_TYPE:14}

        Fedora Debian
        - $ echo ${OS_TYPE:14:6}

        Fedora
        - $ echo ${OS_TYPE:(-6)}

        Debian
        - $ echo ${OS_TYPE:(-6):2}

        De
        - $ echo ${OS_TYPE:(-6):-2}

        Debi

Shell Programming

- ❖ 변수
  - ✓ 매개변수 확장
    - ● 변수 초기화를 위한 확장 변경자
      - ◆ 패턴 삭제 및 파일 경로 추출 옵션: 특정 문자열이나 패턴을 매칭하여 삭제하므로 파일 이름이나 확장자를 추출할 때 매우 유용 - var="/home/user/test.txt"
        - ▪ ${var#pattern}: 앞부분부터 패턴과 가장 처음 일치하는 부분 삭제 ${var#*/} → "home/user/test.txt"
        - ▪ ${var##pattern}: 앞부분부터 패턴과 가장 길게 일치하는 부분 삭제 ${var##*/} → "test.txt" (파일명 추출)
        - ▪ ${var%pattern}: 뒷부분부터 패턴과 가장 처음 일치하는 부분 삭제 ${var%.*} → "/home/user/test" (확장자 제거)
        - ▪ ${var%%pattern}: 뒷부분부터 패턴과 가장 길게 일치하는 부분 삭제 ${var%%/*} → "" (전체 삭제됨)

Shell Programming

- ❖ 변수
  - ✓ 매개변수 확장
    - ● 변수 초기화를 위한 확장 변경자
      - ◆ 문자열 치환 및 대소문자 변환: 변수 안의 특정 문자를 바꾸거나 대소문자를 변경 - var="apple_pie"
        - ▪ ${var/pattern/string}: 처음 매칭되는 패턴을 string으로 한 번 치환 ${var/apple/banana} → "banana_pie"
        - ▪ ${var//pattern/string}: 매칭되는 모든 패턴을 string으로 전부 치환 - ${var//_/ } → "apple pie" (공백 변환)
        - ▪ ${var^}:첫 글자만 대문자로 변환 ${var^} → "Apple_pie"
        - ▪ ${var^^}: 모든 글자를 대문자로 변환 ${var^^} → "APPLE_PIE"
        - ▪ ${var,}: 첫 글자만 소문자로 변환 (var="APPLE") ${var,} → "aPPLE"
        - ▪ ${var,,}: 모든 글자를 소문자로 변환 (var="APPLE") ${var,,} → "apple"

Shell Programming

- ❖ 변수
  - ✓ 매개변수 확장
    - ● 변수 초기화를 위한 확장 변경자
      - ◆ 실습
        - $ FILE_NAME="myvm_container-repo.tar.gz"
        - $ echo ${FILE_NAME#*_}

        container-repo.tar.gz
        - $ echo ${FILE_NAME##*-}

        repo.tar.gz
        - $ echo ${FILE_NAME%.*}

        myvm_container-repo.tar
        - $ echo ${FILE_NAME%%.*}

        myvm_container-repo

Shell Programming

- ❖ 변수
  - ✓ 매개변수 확장
    - ● 변수 초기화를 위한 확장 변경자
      - ◆ 실습
        - $ FILE_PATH="/etc/nova/nova.conf"
        - $ echo ${FILE_PATH%/*}

        /etc/nova
        - $ echo ${FILE_PATH##*/}

        nova.conf
        - $ echo ${#FILE_PATH}

        19

Shell Programming

- ❖ 변수
  - ✓ 매개변수 확장
    - ● 변수 초기화를 위한 확장 변경자
      - ◆ 실습
        - $ OS_TYPE="Redhat Linux Ubuntu Linux Fedora Linux"
        - $ echo ${OS_TYPE/Linux/OS}

        Redhat OS Ubuntu Linux Fedora Linux
        - $ echo ${OS_TYPE//Linux/OS}

        Redhat OS Ubuntu OS Fedora OS
        - $ echo ${OS_TYPE/Linux}

        Redhat Ubuntu Linux Fedora Linux
        - $ echo ${OS_TYPE//Linux}

        Redhat Ubuntu Fedora
        - $ echo ${OS_TYPE/#Redhat/Unknown}

        Unknown Linux Ubuntu Linux Fedora Linux
        - ${OS_TYPE/%Linux/Unknown}

        Redhat Linux Ubuntu Linux Fedora Unknown

Shell Programming

- ❖ 조건문
  - ✓ if
    - ● 개요
      - ◆ if문은 질문을 통해 질문에 대한 답이 참인지 거 짓인지를 판단하여 그 다음 행동에 영향을 줌
      - ◆ 두 변수의 값을 비교한다거나 변수의 값이 특정 값에 도달했는지를 판단할 경우 if문을 사용하면 유용
      - ◆ 운영체제가 Fedora 계열이면 yum install을 이용하여 애플리케이션을 설치하고 운영체제가 Debian 계열이면 apt-get install을 이용하여 애플리케이션을 설치할 경우에도 if문을 사용할 수 있음

Shell Programming

- ❖ 조건문
  - ✓ if
    - ● 기본 사용법

```
if [ 첫번째 조건식 ]
then
        수행할 내용
elif [ 두번째 조건식 ]
then
        수행할 내용
else
        수행할 내용
fi
```

- ◆ elif는 조건식이 여러 개인 경우 사용
- ◆ else는 일치하는 조건식이 없을 때 수행할 내용
- ◆ 조건식 앞 뒤로는 반드시 [ ]가 있어야 하며 대괄호와 조건식 사이에는 반드시 하나의 공백이 있어야 함

Shell Programming

- ❖ 조건문
  - ✓ if
    - ● 조건식 타입
      - ◆ if [$변수 연산자 $변수]; then: 일반적인 조건식 타입으로 두 변수의 값을 비교할 때 사용
      - ◆ if [$변수 연산자 r값]; then: 값이 고정인 경우
      - ◆ if [$변수 연산자]; then: 변수의 값이 문자열이거나 디렉토리와 같은 경우일 때 주로 사용
      - ◆ if [조건식] 연산자 if [조건식]; then: 여러 개의 조건을 AND 나 OR로 복합 연산할 때 사용

Shell Programming

- ❖ 조건문
  - ✓ if
    - 실습 – if_example.sh

```
#!/bin/hash

value1=10
value2=10
if [ $value1 = $value2 ]
then
  echo True
else
  echo False
fi

if [ $value1 = 0 ]
then
  echo True
else
  echo False
fi
```

Shell Programming

- ❖ 조건문
  - ✓ if
    - 실습 – if_example.sh

```
value=""
if [ -z $value ]
then
  echo True
else
  echo False
fi

value=5
if [ $value -gt 0 ] && [ $value -lt 10 ]
then
  echo True
else
  echo False
fi
```

Shell Programming

- ❖ 조건문
  - ✓ switch ~ case
    - ● 개요
      - ◆ switch-case문은 if문에 비해 사용 빈도는 떨어지나 변수의 값에 따라 분기를 해야 하는 경우에 주로 사용
      - ◆ switch-case문은 어떤 셸을 사용하느냐에 따라 기본 사용법이 조금씩 차이가 있음
    - ● 기본적인 사용법

```
case $변수 in
  조건값1)
  수행문1 ;;
  조건값2)
  수행문2 ;;
  조건값3)
  수행문3 ;;
  *)
  수행문
easc
```

Shell Programming

- ❖ 조건문
  - ✓ switch ~ case
    - case-example.sh

```
#!/bin/bash

case $1 in
  start)
  echo "시작"
  ;;
  stop)
  echo "중지"
  ;;
  restart)
  echo "재시작"
  ;;
  help)
  echo "도움말"
  ;;
  *)
  echo "명령어를 입력해주세요!!!"
esac
```

Shell Programming

- ❖ 조건문
  - ✓ switch ~ case
    - ● 실행
      - ◆ $ sh case_example.sh

        명령어를 입력해주세요!!!
      - ◆ $ sh case_example.sh start

        시작

Shell Programming

- ❖ 반복문
  - ✓ for~in
    - ● 기본 사용법
      - ◆ for ~ in: 리스트나 배열과 같은 특정 범위의 값들을 꺼내어 변수에 저장하고 해당 값을 모두 사용하면서 수행문을 처리하는 방식

        for 변수 in [범위(리스트 또는 배열, 묶음 등)]

        do

          반복할 수행문

        done
      - ◆ 초기값이 특정 조건에 해당할 때까지 값을 증가시키면서 수행문을 처리하는 방식

        for ((변수=초기값; 조건식; 증가값))

        do

          반복할 수행문

        done

Shell Programming

- ❖ 반복문
  - ✓ for~in
    - for-example.sh

```
#!/bin/bash

for num in 1 2 3
do
  echo $num;
done

numbers="1 2 3"
for num in $numbers
do
  echo $num;
done
```

Shell Programming

- ❖ 반복문
  - ✓ for~in
    - for-example.sh

```
for file in $HOME/*
do
  echo $file;
done

for num in {1..5}
do
  echo $num;
done

for num in {1..10..2}
do
  echo $num;
done
```

Shell Programming

- ❖ 반복문
  - ✓ for~in
    - for-example.sh

```
array=("apple" "banana"   "pineapple")
for fruit in ${array[@]}
do
  echo $fruit;
done

for ((num=0; num<3; num++))
do
  echo $num;
done
```

Shell Programming

- ❖ 반복문
  - ✓ for~in
    - bash for-example.sh

```
1
2
3

1
2
3

/home/adam/shell
/home/adam/test

1
2
3
4
5
```

Shell Programming

- ❖ 반복문
  - ✓ for~in
    - bash for-example.sh

```
1
3
5
7
9

apple
banana
pineapple

0
1
2
```

Shell Programming

- ❖ 반복문
  - ✓ while
    - while문은 특정 범위를 사용하는 것이 아니라 변수의 값이 특정 조건에 맞을 때까지 계속 반복하는 경우에 주로 사용
    - 기본 사용

```
while [$변수1 연산자 $변수2]
do
수행할 문장
done
```

Shell Programming

- ❖ 반복문
  - ✓ while
    - while-example.sh

```
#!/bin/bash

num=0

while [ $num -lt 3 ]
do
  echo $num
  num=$((num+1))
Done
```

    - bash while-example.sh

```
0
1
2
```

Shell Programming

- 기타 제어문
  - ✓ until: 반복문
  - ✓ break는 주로 반복문을 종료할 때 사용
  - ✓ continue는 반복문의 조건식으로 돌아가게 됨
  - ✓ exit는 해당 프로그램을 완전히 종료
  - ✓ return은 함수 안에서 사용할 수 있으며 함수를 호출한 곳으로 돌아가게 함

Shell Programming

- 연산자
  - ✓ 문자열 연산자
    - -z: 문자열의 길이가 0이면 참 – [-z 문자열변수]
    - -n: 문자열의 길이가 0이 아니면 참 – [-n 문자열변수]
  - ✓ 비교 연산자
    - 정수 비교 연산자
      - -eq, -ne, -gt, -ge, -lt, -le: [$변수1 –eq $변수2]
      - >, >=, <, <=: 이 연산자를 사용하는 경우에는 중첩 소괄호를 사용 – (($변수1 > $변수2))
    - 문자열 비교 연산자
      - =, ==, !=: [$변수1 = $변수2]
      - >, <: [[$변수1 > $변수2]]

Shell Programming

- ❖ 연산자
  - ✓ 논리 연산자
    - and 연산
      - ◆ [조건식1 –a 조건식2]
      - ◆ [조건식1] && [조건식2]
      - ◆ [[조건식1 && [조건식2]]
    - or 연산
      - ◆ [조건식1 –o 조건식2]
      - ◆ [조건식1] || [조건식2]
      - ◆ [[조건식1 || [조건식2]]

Shell Programming

- ❖ 연산자
  - ✓ 파일 관련 연산자
    - -d: 파일이 디렉토리이면 참
    - -e: 파일이 존재하면 참
    - -f: 파일이면 참
    - -L: 파일이면서 심볼릭 링크이면 참
    - -g: 파일에 set-group-id가 설정되면 참
    - -u: 파일에 set-user-id가 설정되면 참
    - -r: 읽기 권한이 있으면 참
    - -w: 쓰기 권한이 있으면 참
    - -x: 실행 권한이 있으면 참
    - -s: 크기가 0보다 크면 참
    - -O: 스크립트 실행 소유자와 동일한 소유자이면 참
    - -G: 스크립트 실행 소유자와 동일한 그룹이면 참

Shell Programming

- ❖ 연산자
  - ✓ 파일 비교 연산자
    - -nt: 두 개의 파일을 비교해서 앞의 파일이 최신 파일이면 참
    - -ot: 두 개의 파일을 비교해서 앞의 파일이 오래된 파일이면 참
    - -ef: 두 개의 파일을 비교해서 동일한 파일이면 참

Shell Programming

- ❖ Maker Function
  - ✓ eval
    - ● 문자열을 명령문으로 인식하고 실행
    - ● eval.sh

```
#!/bin/sh

str="ls -l eval.sh"
echo $str
eval $str
exit 0
```

    - ● sh eval.sh

```
ls -l eval.sh
-rw-rw-r-- 1 adam adam 59 Jun  2 07:40 eval.sh
```

Shell Programming

- ❖ Maker Function
  - ✓ export
    - 외부 변수로 선언
    - 선언한 변수를 다른 프로그램에서도 사용할 수 있게 함
    - exp1.sh

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

    - exp2.sh

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

    - sh exp2.sh

외부 변수

Shell Programming

- Maker Function
  - ✓ printf
    - C 언어의 printf 함수와 유사하게 사용
    - printf.sh

```
#!/bin/sh

var1=100.5
var2="재미있는 리눅스~~"
printf "%5.2f ₩n₩n ₩t %s ₩n" $var1 "$var2"
exit 0
```

    - sh printf.sh

```
100.50


        재미있는 리눅스~~
```

Shell Programming

- ❖ Maker Function
  - ✓ set 과 ${명령)
    - 리눅스 명령을 결과로 사용하려면 '$(명령)' 형식을 사용
    - 결과를 파라미터로 사용하고자 할 때는 set 명령과 함께 사용
    - set.sh

```
#!/bin/sh

echo "오늘 날짜는 $(date) 입니다"
set $(date)
echo "오늘은 $3 요일입니다"
exit 0
```

    - sh set.sh

```
오늘 날짜는 Tue Jun  2 08:01:01 AM UTC 2026 입니다
오늘은 2 요일입니다
```

정규 표현식

- 개요
  - ✓ 정규 표현식(Regular Expression, 줄여서 Regex)은 특정한 규칙을 가진 문자열의 집합을 표현하기 위해 사용하는 형식 언어
  - ✓ 방대한 텍스트 중에서 내가 원하는 조건의 문자열만 골라내거나 바꾸고 싶을 때 사용하는 문법

정규 표현식

- POSIX 기본 및 확장 문법
  - ✓ 개요
    - 패턴을 기술하는 문자열 내의 각 문자들은 메타 문자(특별한 의미)나 정규 문자로 이해됨
    - 정규식 "a." 을 표현했다고 하면 "a."는 a 와 일치하는 문자 하나와 뉴라인을 제외한 모든 문자 하나를 갖는 문자열과 일치시키는 메타 문자
  - ✓ 일치 및 위치 지정 메타 문자
    - .: 임의의 한 문자로 줄바꿈 문자를 제외한 어떤 글자든 딱 한 글자와 매칭

      a.b → aab, a1b (O) / ab (X, 가운데 글자 없음)
    - ^: 문자열의 시작으로 문자열이 반드시 이 기호 뒤의 문자로 시작해야 함

      ^Hello → Hello world (O) / Say Hello (X)
    - $: 문자열의 끝으로 문자열이 반드시 이 기호 앞의 문자로 끝나야 함

      bye$ → Good bye (O) / bye everyone (X)
    - |: OR (또는)의 의미로 A 또는 B 중 하나와 일치

      apple|banana → apple 또는 banana

정규 표현식

- POSIX 기본 및 확장 문법
  - ✓ 반복 관련 메타 문자(수량자)
    - *: 0번 이상 반복으로 앞의 문자가 없어도 되고, 여러 개 있어도 됨
      lo*v → lv, lov, loov, looov 모두 일치
    - +: 1번 이상 반복으로 앞의 문자가 최소한 1개 이상은 있어야 함
      lo+v → lov, loov (O) / lv (X)
    - ?: 0번 또는 1번 존재해야 하는데 앞의 문자가 있거나 없을 때 사용
      apples? → apple (O), apples (O)
    - {n}: 정확히 n번 반복해야 하는데 지정한 숫자만큼 딱 맞춰 반복되는 경우
      a{3} → aaa와 일치
    - {n,}: n번 이상 반복
      a{3,} → aaa, aaaa와 일치
    - {n,m}: n번에서 m번 반복해야 하는데 최소 n번, 최대 m번 반복되는 범위를 지정
      a{2,4} → aa, aaa, aaaa와 일치

정규 표현식

- ❖ POSIX 기본 및 확장 문법
  - ✓ 문자열을 묶거나 범위를 한정
    - [ ] (문자 클래스): 대괄호 안에 넣은 문자들 중 딱 한 글자와 매칭
      - ◆ [abc] → a, b, c 중 한 글자
      - ◆ [a-z] → 알파벳 소문자 중 한 글자 (하이픈-은 범위를 뜻함)
      - ◆ [^0-9] → 숫자가 아닌 한 글자 (대괄호 안에서 ^는 부정을 뜻함)
      - ◆ ( ) (그룹화): 여러 문자를 하나의 단위로 묶을 때 사용하는데 반복 기호를 세트로 적용할 때 유용한데 (tom)+ → tom, tomtom, tomtomtom 등 'tom' 단위가 반복되는 문자열
    - 기타
      - ◆ ₩: 특수 문자를 원래의 문자 의미대로 해석
      - ◆ ₩b: 단어 끝의 공백
      - ◆ ₩B: 라인 끝의 공백
      - ◆ ₩<: 단어 시작에서 공백
      - ◆ ₩>: 단어 끝에서 공백

정규 표현식

- POSIX 문자 클래스
  - ✓ [:digit:]: 모든 숫자 [0-9] 또는 ₩d
  - ✓ [:lower:]: 알파벳 소문자 [a-z]
  - ✓ [:upper:]: 알파벳 대문자 [A-Z]
  - ✓ [:alpha:]: 모든 알파벳(대소문자) [a-zA-Z]
  - ✓ [:alnum:]: 알파벳과 숫자 전체 [a-zA-Z0-9]
  - ✓ [:xdigit:]: 16진수 문자 [0-9a-fA-F]
  - ✓ [:space:]: 공백 문자 (스페이스, 탭, 줄바꿈 등) [₩t₩r₩n₩v₩f] 또는 ₩s
  - ✓ [:blank:]: 스페이스와 탭(줄바꿈 제외) [₩t]
  - ✓ [:punct:]: 문장 부호 및 특수문자 [! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [₩] ^ _ ` { | } ~]
  - ✓ [:print:]: 출력 가능한 모든 문자(공백 포함) [₩x20-₩x7E]
  - ✓ [:graph:]: 공백을 제외하고 눈에 보이는 모든 문자 [₩x21-₩x7E]
  - ✓ [:cntrl:]: 제어 문자(화면에 표시되지 않는 문자) [₩x00-₩x1F₩x7F]

정규 표현식

❖ 예제

✓ 메타 문자 .을 이용하는 경우

- C로 시작해서 U로 끝나는 3글자: grep 'C.U' expression.txt

  CPU model is Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz

- C로 시작해서 e로 끝나는 4글자: grep 'C..e' expression.txt

  CPU model is Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz

✓ 메타 문자 *, ₩, ? 와 문자 클래스 [:lower:]를 이용하는 경우

- q로 시작하고 ?로 끝나야 하며 중간에는 영문 소문자가 개수 상관없이 존재하는 경우: grep -E 'q[[:lower:]]*₩?' expression.txt

  Do you have any questions? or Do you need any help?

정규 표현식

- 예제
  - ✓ 메타 문자 + 와 ^를 이용하는 경우
    - - 다음에 숫자 2가 반복되고 -로 끝나는 단어를 포함하는 경우: grep -E '₩-2+₩-' expression.txt

      phone: 010-2222-5668
    - #으로 시작하는 경우: grep -E '^#' expression.txt

      #==========#

      # Date: 2020-05-05

      # Author: NaleeJang

      # Description: regular expression test file

      #==========#

      # System Information

      # Help

      # Contacts

정규 표현식

❖ 예제

✓ 메타 문자 ^, {N}, {N,} 그리고 문자 클래스 [:alpha:]를 이용할 경우

- 영문 5글자 뒤에 :이 오는 단어 검색: grep -E '^[[:alpha:]]{5}:' expression.txt

  phone: 010-2222-5668

- 알파벳 5글자 이상으로 시작하고 뒤에 공백을 가진 단어 검색: grep -E '^[[:alpha:]]{5,}[[:space:]]' expression.txt

  Today is 05-May-2020.

  Current time is 6:04PM.

  Memory size is 32GiB

정규 표현식

❖ 예제

✓ 메타 문자 {N, M} 그리고 문자 클래스 [:alpha:], [:digit:]를 이용할 경우

- 영문 4글자 이상 6글자 이하로 끝나는 단어 검색: grep -E '[[:alpha:]]{4,6}$' expression.txt

  Regular Expression

  # Author: NaleeJang

  # Description: regular expression test file

  # System Information

  # Help

  # Contacts

- 숫자 4글자 이상 6글자 이하와 .으로 끝나는 단어 검색: grep -E '[[:digit:]]{4,6}.$' expression.txt

  Today is 05-May-2020.

스크립트 예제

- ❖ 사용자 계정 생성
  - ✓ 개요
    - ● 필요한 정보
      - ◆ 사용자 계정 ID와 패스워드 필요
      - ◆ 사용자 계정 생성 명령어: useradd
      - ◆ 패스워드 설정 명령어: passwd
    - ● 프로세스
      - ◆ 사용자 계정과 패스워드를 입력받음
      - ◆ 입력 정보가 없으면 에러 메시지를 보여주고 셸 스크립트를 종료
      - ◆ 여러 명의 사용자 계정을 생성할 경우에는 for문을 이용
      - ◆ 생성하고자 하는 사용자 계정이 있는지 확인
      - ◆ 사용자 계정이 없으면 사용자 계정을 생성하고 패스워드를 설정
      - ◆ 만일 사용자 계정이 있으면 계정이 있다고 메시지를 출력

스크립트 예제

- 사용자 계정 생성
  - ✓ 스크립트: adduser.sh

```
#!/bin/bash

# 반드시 루트 권한(sudo)으로 실행했는지 확인
if [[ $EUID -ne 0 ]]; then
    echo "Error: 이 스크립트는 반드시 root 권한(sudo)으로 실행해야 합니다."
    exit 1
fi

# 사용자 계정 및 패스워드가 입력되었는지 확인
if [[ -n $1 ]] && [[ -n $2 ]]
then
  # 공백을 기준으로 배열 생성
  UserList=($1)
  Password=($2)
```

스크립트 예제

- 사용자 계정 생성
  - ✓ 스크립트: adduser.sh

```
# for문을 이용하여 사용자 계정 생성
for (( i=0; i < ${#UserList[@]}; i++ ))
do
  # getent 명령어를 사용하는 것이 /etc/passwd를 cat하는 것보다 안전하고 정확합니다.
  if ! getent passwd "${UserList[$i]}" > /dev/null
  then
    # 사용자 생성
    useradd -m "${UserList[$i]}"

    # Ubuntu/CentOS 모두 호환되는 패스워드 설정 방식 (chpasswd 사용)
    echo "${UserList[$i]}:${Password[$i]}" | chpasswd

    echo "Success: User ${UserList[$i]} has been created."
  else
    # 사용자가 이미 존재할 때
    echo "Warning: This user ${UserList[$i]} already exists."
  fi
done
```

스크립트 예제

- 사용자 계정 생성
  - ✓ 스크립트: adduser.sh

```
else
  # 사용법 안내
  echo -e 'Please input user id and password.₩nUsage: sudo ./adduser-script.sh "user01 user02" "pw01 pw02"'
fi
```

스크립트 예제

❖ 사용자 계정 생성

✓ 실행

- bash ./adduser.sh "user01" "pw01"

  Error: 이 스크립트는 반드시 root 권한(sudo)으로 실행해야 합니다.

- sudo bash ./adduser.sh "user01" "pw01"

  Warning: This user user01 already exists.

- sudo bash ./adduser.sh "user02" "pw02"

  Success: User user02 has been created.

- id user02

  uid=1002(user02) gid=1002(user02) groups=1002(user02)

스크립트 예제

- ❖ 패스워드 생성 규칙을 적용할 때
  - ✓ 상황
    - 보안의 가장 기본은 패스워드 보안
    - 대부분의 시스템이나 웹 사이트에서는 복잡한 패스워드를 요구하는 곳이 많음
    - 리눅스 역시 생성되는 계정의 패스워드를 설정할 때 패스워드를 복잡하게 설정하여 외부로부터 침입하기 어려운 패스워드를 만들도록 설정할 수 있음
    - 복잡한 패스워드 설정은 pam_pwquality라는 라이브러리에 의해 설정할 수 있음

스크립트 예제

- ❖ 패스워드 생성 규칙을 적용할 때
  - ✓ 방법
    - ● 패키지 관련
      - ◆ 패스워드 생성 법칙은 pam_pwquality라는 라이브러리에 의해 설정되고 관리됨
      - ◆ 페도라 계열의 리눅스에서는 pam_pwquality라는 라이브러리가 기본적으로 탑재되어 있어 별도의 구성이 필요하지 않지만 데비안 계열의 리눅스에서는 패스워드 생성 법칙을 적용하기 위해서 libpam-pwquality라는 패키지를 별도로 설치해야 함

스크립트 예제

- ❖ 패스워드 생성 규칙을 적용할 때
  - ✓ 방법
    - ● 필요한 정보
      - ◆ 설정 파일 경로
        - ▪ 페도라 계열: /etc/pam.d/system-auth
        - ▪ 데비안 계열: /etc/pam.d/common-password
      - ◆ 패스워드 생성 법칙 항목들과 의미
        - ▪ Retry: 패스워드 입력 실패 시 재시도 횟수
        - ▪ Minlen: 최소 패스워드 길이
        - ▪ Difok: 이전 비밀번호와 유사한 문자 개수
        - ▪ Lcredit: 소문자 최소 요구 개수
        - ▪ Ucredit: 대문자 최소 요구 개수
        - ▪ Dcredit: 숫자 최소 요구 개수
        - ▪ Ocredit: 특수 문자 최소 요구 개수
        - ▪ Enforce_for_root: root 사용자 패스워드 생성 법칙 적용

스크립트 예제

- ❖ 패스워드 생성 규칙을 적용할 때
  - ✓ 방법
    - ● 프로세스
      - ◆ 운영체제 타입을 확인
      - ◆ 페도라 계열의 리눅스면 /etc/pam.d/system -auth 파일에 설정을 적용
      - ◆ 데비안 계열의 리눅스라면 우선 /etc/pam.d/common -password 파일이 있는지 확인
      - ◆ 파일이 없으면 libpam-pwquality 패키지를 설치
      - ◆ 파일이 있으면 /etc/pam.d/common-passwordoil 설정을 적용

스크립트 예제

- 패스워드 생성 규칙을 적용할 때
  - ✓ 스크립트 작성 및 실행
    - 스크립트 파일 작성: conf-pwpolicy.sh

```
#!/bin/bash

# 운영체제 타입 확인
ostype=$(cat /etc/*release| grep ID_LIKE | sed "s/ID_LIKE=//;s/₩"//g")

# 운영체제가 페도라 계열일 경우
if [[ $ostype == "fedora" ]]; then
  # 설정 여부 체크
  conf_chk=$(cat /etc/pam.d/system-auth | grep 'local_users_only$' | wc -l)
  # 설정이 안되어 있으면 설정 후 설정 내용 확인
  if [ $conf_chk -eq 1 ]; then
    sed -i 's/₩(local_users_only$₩)/₩1 retry=3 authtok_type= minlen=8 lcredit=-1
ucredit=-1 dcredit=-1 ocredit=-1 enforce_for_root/g' /etc/pam.d/system-auth
    cat /etc/pam.d/system-auth | grep '^password[[:space:]]*requisite'
  fi
```

스크립트 예제

- 패스워드 생성 규칙을 적용할 때
  - ✓ 스크립트 작성 및 실행
    - 스크립트 파일 작성: conf-pwpolicy.sh

```
# 운영체제가 데비안 계열일 경우
elif [[ $ostype == "debian" ]]; then
  # pam_pwquality.so가 설치되어 있는지 설정파일을 통해 확인
  conf_chk=$(cat /etc/pam.d/common-password | grep 'pam_pwquality.so' | wc -l)
  # 설치가 안되어 있으면 libpam-pwquality 설치
  if [ $conf_chk -eq 0 ]; then
     apt install libpam-pwquality
  fi
  # 설정 여부 체크
  conf_chk=$(cat /etc/pam.d/common-password | grep 'retry=3$' | wc -l)
  # 설정이 안되어 있으면 설정 후 설정 내용 확인
  if [ $conf_chk -eq 1 ]; then
     sed -i 's/₩(retry=3$₩)/₩1 minlen=8 maxrepeat=3 ucredit=-1 lcredit=-1
dcredit=-1 ocredit=-1 difok=3 gecoscheck=1 reject_username enforce_for_root/g'
/etc/pam.d/common-password
     echo "================================================="
     cat /etc/pam.d/common-password | grep '^password[[:space:]]*requisite'
  fi
fi
```

스크립트 예제

❖ 패스워드 생성 규칙을 적용할 때

✓ 스크립트 작성 및 실행

- 스크립트 실행 전 확인: sudo bash ./adduser.sh "user03" "pw03"

  Success: User user03 has been created.

- 스크립트 실행: sudo bash conf-pwpolicy.sh

  No VM guests are running outdated hypervisor (qemu) binaries on this host.
  ==========
  password requisite pam_pwquality.so retry=3 minlen=8 maxrepeat=3 ucredit=-1 lcredit=-1 dcredit=-1 ocredit=-1 difok=3 gecoscheck=1 reject_username enforce_for_root
  password requisite pam_deny.so

- 스크립트 실행 후 확인: sudo bash ./adduser.sh "user04" "pw04"

  BAD PASSWORD: The password contains less than 1 uppercase letters
  BAD PASSWORD: The password contains less than 1 uppercase letters
  BAD PASSWORD: The password contains less than 1 uppercase letters
  chpasswd: (user user04) pam_chauthtok() failed, error:
  Have exhausted maximum number of retries for service
  chpasswd: (line 1, user user04) password not changed
  Success: User user04 has been created.