리눅스 기본 명령어

date 날짜출력
hostname 서버이름 출력
uname 운영체제 출력
uname -a 운영체제 상세정보 출력
whoami 유저이름 출력
ls 디렉터리 내용을 리스트
ls -a 모든 파일과 디렉터리 리스트
ls -asl 모든파일 자세히 리스트
passwd 패스워드 변경
clear 화면 청소
man (명령어) 명령어 매뉴얼
pwd 현재 작업 디렉터리를 프린트
mkdir (폴더) 새디렉터리 생성
cd (경로) 현재 작업 디렉터리를 이동
cat 키보드 입력 내용을 출력
cat (파일) 파일 내용 출력
cat > (파일) 키보드 입력 내용을 파일에 저장
more (파일) 파일을 페이지 단위로 화면에 출력
head (파일) 파일의 앞부분(10줄)을 출력
tail (파일) 파일의 뒷부분(10줄)을 출력
wc (파일) 파일의 줄, 단어, 문자 갯수 출력
cp (파일1) (파일2) 파일1을 파일2에 복사
cp (파일) (폴더) 파일의 복사본을 폴더 내에 만듬
cp -i 대화형 옵션
mv (파일1) (파일2) 파일1의 이름을 파일2로 변경
mv (파일) (폴더) 파일을 지정된 폴더로 이동
rm (파일) 파일 삭제
rm -r (폴더) 폴더 삭제
ln [-s] (파일1) (파일2) 이름:파일2 파일1 바로가기 생성
-s 옵션은 심볼릭 링크 (바로가기)
chmod (oct 3) 접근권한 변경
chown (user) (파일)파일이나 폴더의 소유자 변경
chown (user) (폴더)
chgrp (group) (파일) 파일이나 폴더의 그룹 변경
chgrp (group) (폴더)

고정 IP 설정

가상머신 기본 Network 설정은 DHCP
*DHCP : IP 자동할당 재부팅시 ip 변경 가능

고정 IP 설정 방법
/etc/netplan/00-installer-config.yaml 파일 수정
addresses:
- 192.168.~.~/24
~~~~~~~~
routes:
- to: default
via: 192.168.0.1

yaml 파일이 존재하지 않을 경우
sudo netplan generate

파일 수정 후
sudo netplan apply

리눅스 파일 종류

* 일반파일
* 디렉터리(폴더)
* 특수 파일
  * 물리적인 장치들을 파일로 표현
  * 리눅스 운영체제 안에서 사용되는 프로세스간 통신 수단 등을 파일로 표현
* 심볼릭 링크 파일
  * 어떤파일을 가리키는 또 하나의 경로명을 저장하는 파일

접근권한 표현 : 8진수

* 접근권한 8진수 변환

사용자	그룹	기타
rwx	r-x	r-x
111	101	101
7	5	5

ex) chmod 644 (파일)
= user(rw-) group(r—) other(r—)

접근권한 표현 : 기호

구분	기호와 의미
사용자 범위	u(소유자), g(그룹), o(기타), a(all)
연산자	+(추가), -(제거), =(설정)
권한	r,w,x

ex) chmod u+r
= user(r—)

출력 재지정

* 명령어의 표준 출력 내용을 모니터 대신에 파일에 저장
명령어 > 파일
ex) who > names.txt

출력 추가

* 명령어의 표준 출력 내용을 모니터 대신에 파일에 추가
명령어 >> 파일
ex) who >> names.txt

입력 재지정

* 명령어의 표준입력을 키보드 대신에 파일에서 받음
명령어 < 파일
ex) wc < list1.txt

문서 내 입력

* 명령어의 표준 입력을 키보드 대신에 단어와 단어 사이의 입력 내용으로 받음
* 보통 스크립트 내에서 입력을 줄 때 사용
명령어 << 단어
ex) wc << end

파이프

* | : 두개 이상의 명령어의 입출력을 연결
명령어1 | 명령어2
ex) ls | sort -r
(명령어1 한거에 명령어 2)
~~~~~~~~

쉘 프로그래밍 문법

변수

* 쉘 변수의 특징
  * 변수를 사용하기 전에 선언하지 않음
  * 변수의 값은 항상 문자열로 간주됨
  * 대소문자를 구별함
  * $를 변수 앞에 붙여서 변수의 값을 접근
  * read 명령어를 이용하여 사용자입력을 변수에 저장 가능
* 변수와 따옴표
  * 변수의 값이 하나이상의 공백문자를 포함하면 따옴표를 이용해 표시해야 함
  * “ ”와 ‘ ’ 모두 사용가능
    * “ “ : 변수값에 $변수명이 포함될때 변수를 저장된 값으로 대체
    * ‘ ‘ : $변수명을 그대로 출력
    * \ : $의 의미를 제거
* 환경변수
  * 쉘 스크립트가 시작할 때 환경으로부터 얻은 값으로 초기화 되는 변수
  * 일반적으로 대문자 사용
  * 계정 사용자에 의해서 추가 가능
  * 기본적인 환경 변수
    * $HOME : 현재 사용자의 홈 디렉토리
    * $PATH : 명령을 검색할 디렉토리들을 콜론으로 구분해 놓은 목록
    * $PS1 : 명령 프롬프트. 추가적인 입력을 받아들이는 프롬프트(일반적으로 >)
    * $PS2 : 두번째 프롬프트. 추가적인 입력을 받아들이는 프롬프트 (일반적으로 >)
    * $IFS : 입력 필드 구분자. 쉘이 입력을 읽을때 단어들을 구분하는 문자 목록
    * $0 : 쉘 스크립트의 이름
    * $# : 전달된 매개변수의 개수
    * $$: 쉘 스크립트의 프로세스 ID. 주로 임시 파
    * 일 이름을 유일하게 생성 하기 위해 사용됨
  * 매개변수
    * 스크립트를 실행할 때 전달되는 변수
    * 매개변수가 전달되지 않더라도 $#은 0
    * 매개변수 종류
      * $1, $2, … : 스크립트에 주어진 매개변수
      * $* : 모든 배개변수들의 목록을 하나의 변수로 나타냄. 환경 변수 IFS에 저장된 첫번째 문자로 매개변수 구분
      * $@ : $*의 확장형. IFS가 비어 있다면 매개 변수들은 함께 나열됨 IFS가 설정되어도 구분하여 나타냄

조건

* test, [ ]
  * ex) test string1 = string2
  * ex) [string1 = string2] ([띄어쓰기 필수])
* 문자열 비교
  * [string] : 빈 문자열이 아니라면 참
  * [string1 = string2] : 두 문자열이 같다면 참
  * [string1 != string2] : 두 문자열이 다르면 참
  * [-n string] : 문자열이 null이 아니라면 참
  * [-z string] : 문자열이 null이라면 참
* 산술비교
  * [a -eq b] : 두 표현식 값이 같다면 참 (EQual)
  * [a -ne b] : 두 표현식 값이 다르면 참 (Not Equal)
  * [a -gt b] : a > b 이면 참 (Greater Then)
  * [a -ge b] : a >= b 이면 참 (Greater Equal)
  * [a -lt b] : a < b 이면 참 (Less Then)
  * [a -le b] : a <= b 이면 참 (Less Equal)
  * [! a] : not a
  * [a -a b] : a and b (And)
  * [a -o b] : a or b (Or)
* 파일조건
  * [-b file] : file이 블럭 디바이스면 참
  * [-c file] : file이 문자 디바이스면 참
  * [-d file] : file이 디렉토리면 참
  * [-e file] : file이 존재하면 참
  * [-f file] : file이 존재하고 정규파일이면 참
  * [-g file] : file이 set-group-id 파일이면 참
  * [-h file] : file이 심볼릭 링크면 참
  * [-L file] : file이 심볼릭 링크면 참
  * [-k file] : file이 Sticky bit가 셋팅되어 있으면 참
  * [-p file] : file이 파이프로 지정되어있으면 참
  * [-r file] : 현재 사용자가 읽을 수 있는 file이면 참
  * [-s file] : file이 비어있지 않으면 참
  * [-S file] : 소켓디바이스이면 참
  * [-t FD] : FD가 열려진 터미널이면 참
  * [-u file] : file 이 set-user-id 파일이면 참
  * [-w file] : 현재사용자가 쓸수있는 파일이면 참
  * [-x file] : 현재사용자가 실행할 수 있는 파일이면 참
  * [-O file] : file의 소유자가 현재사용자이면 참
  * [-G file] : file의 그룹이 현재사용자의 그룹과 같으면 참
  * [file1 -nt F] : file1이 file2보다 새로운 파일이면 참
  * [file1 -ot F] : file1이 file2보다 오래된 파일이면 참
  * [file1 -ef F] : file1이 file2의 하드링크파일이면 참

if 문

* 조건에 따라 수행할 부분을 결정

구조)

```
if 조건
then
   if 조건
   then
      명령어
   else
      명령어
   fi
elif 조건
then
   명령어
else
   명령어
fi
```

ex)

```
if [ $timeofday = “yes” ]
then
   echo “Good morning”
elif [$timeofday = “no” ]
then
   echo “Good afternoon”
else
   echo “Sorry, $timeofday not recognized.”
   exit 1
fi
```

case 문

* 값에 따라 수행될 부분을 결정

구조)

```
case 변수 in
패턴1)
   명령어
   …
   ;;
패턴2)
   명령어
   …
   ;;
*)
   명령어
   …
   ;;
esac
```

ex)

```
case “$timeofday” in
   “yes”) echo “Good morning”;;
   “no”) echo “Good afternoon”;;
   *) echo “Sorry, answer not recognized”;;
esac
```

for 문

* 패턴 리스트 수에 따라 반복을 수행

구조)

```
for 변수 in 값(리스트);
do
   명령어
   …
done
```

ex)

```
for file in $(ls f*);
do
   echo $file
done
```

*(())을 이용하여 C, Java같은 형식의 for문 사용가능

ex)

```
for ((i=0;i<=10;i++));
do
   echo $i
done
```

*{..}을 이용하여 range함수 구현 가능

ex)

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

whlie 문

* 조건이 맞는 동안 반복을 수행

구조)

```
while 조건
do
   명령어
   …
done
```

ex)

```
while [ “$trythis” != “secret” ];
do
   echo “Sorry, try again”
   read trythis
done
```

until 문

* 조건이 맞을때까지 반복을 수행

구조)

```
until 조건
do
   명령어
   …
done
```

ex)

```
until [ ! $try ];
do
   read try >> try.txt
done
```

리스트

* 명령어를 일렬로 연결
* AND 리스트
  * 명령어1 & 명령어2 & 명령어3 & …
  * 왼쪽부터 명령어가 false를 반환할때까지 실행
  * false 반환 받으면 뒤에는 실행안됨
* OR 리스트
  * 명령어1 | 명령어2 | 명령어3 | …
  * 왼쪽부터 명령어가 true를 반환할때까지 실행
  * true 반환 받으면 뒤에는 실행안됨

ex)
```
case “$timeofday” in
  yes | y | Yes | YES) echo “Good morning”;;
  n* | N*) echo “Good afternoon”;;
  *) echo “Sorry, answer not recognized”;;
esacs
```

함수

구조)
* 함수 정의

```
함수이름 (){
  명령어
  …
}
```

* 함수 호출

```
함수이름
```

ex)
```
foo(){
  echo “Function foo is executing”
}

foo
```

* 매개변수 전달
  * 함수이름 매개변수1 …
  * 함수내부에서는 매개변수를 $*, $1, $2 … 로 접근
* 값 반환
  * return 값

ex)
```
foo(){
  echo “Hi, $1”
}

foo james
```

명령수행

* 쉘 스크립트에서는 두가지 형식의 명령 수행 가능
  * 외부명령 : 명령 프롬프트에서 수행할 수 있는 명령
  * 내부명령 : 쉘 내부적으로 구현되어 명령 프롬프트에서는 실행되지 않는 명령
* 명령어들

break	반복문 종료
:	null 명령, true의 별칭
continue	반복 계속
eval	인자의 연산
exec	현재 쉘을 다른 프로그램으로 대체
exit n	종료 코드 반환
	0 성공
	1 ~ 125 에러코드
	127 명령 찾을 수 없음
	128이상 신호발생
export	특정 변수를 하위 쉘의 환경변수로
expo	표현식 연산
printf	echo 상위호환
return	함수 값 반환
set	쉘에 매개변수를 설정
shift	매개 변수의 순서를 이동
trap	신호를 받았을때 작동 지정
find	파일 검색
grep	파일에서 문자열 검색

export

* export 뒤에 명령어를 다음 bash 환경변수로 보낸다.

ex)
* export2

```
echo “$foo”
echo “$bar”
```

* export1

```
foo=“abcd”
export bar=“efgh”
./export2
```

* 실행결과

```
efgh
```

expr, printf

* expr
  * 표현식 수행
    * x=`expr $x + 1`
    * x=$(expr $x + 1)
    * x=$(($x+1))
  * expr로 수행할수 있는 표현식
    * |, &, =, >, >=, <, <=, !=, +, -, *, /, %
* printf
  * printf “형식 문자열” 매개변수1 매개변수2 …
  * 이스케이프 시퀀스
    * \\, \a, \b, \f, \n, \r, \t, \v
  * 변환지정자
    * d, c, s, %
  * ex)
    * printf “%s\n” hello
    * - “hello”
    * printf “%s %d\t%s” Hi There” 15 people
    * - “Hi There 15 people”
    * printf “%05d” 1
    * - 00001

find

* 컴퓨터에서 파일 검색
  * find 경로 옵션 테스트 작동
  * ex) find ./ -name wish -print
* 주요 옵션
  * -depth : 디렉토리 자체를 살펴보기전에 디렉토리의 내용을 검색
  * -follow : 심볼릭 링크를 따라가서 검색
  * -maxdepths -N : 최대 N 수준의 디렉토리를 검색
  * -mount (or -xdev) : 다른 파일 시스템의 디렉토리는 검색하지 않음
* 주요 테스트
  * -atime N : 파일이 N일 이전에 마지막으로 액세스 된것을 검색
  * -mtime N : N일 이전에 수정된 것 검색
  * -name "패턴" : 경로를 제외한 파일의 이름이 주어진 패턴에 일치하는 것 검색
  * -newer otherfile : otherfile 보다 최신인 파일 검색
  * -type C : 파일 형식이 C 인것 (f : 파일, d : 디렉토리)
  * -user 사용자 이름 : 주어진 사용자가 소유한 파일 검색
* 연산자
  * !, -not
  * -a, -and
  * -o, -or
* ex)
  * find . -newer first -print
  * find . -newer first -type f -print
  * find . \(-name "_*" -or -newer first \) -type f -print
* 작동
  * -exec : 특정 명령 실행
  * -ok : exec와 동일, 사용자에게 명령 실행 여부 물어봄
  * -print : 화면에 출력
  * -ls : 현재 파일에 대해서 ls a명령 실행

grep

* 파일에서 문자열 검색
  * grep 옵션 패턴 파일
* 옵션
  * -c : 일치하는 줄을 모두 출력하지 않고, 일치하는 줄의 개수를 출력
  * -E : 확장 표현식 적용
  * -h : 각 출력 줄에 파일이름을 붙이는 작업을 수행하지 않음
  * -i : 대소문자 구별 안함
  * -l : 일치된 줄에 해당하는 파일 명을 나열. 일치된 줄은 출력하지 않음
  * -v : 일치되지 않은 줄만을 선택
* ex)
  * grep print second
  * grep -c print first second
  * grep -c -v print first second

명령 실행

* 쉘 스크립트 안에서 명령을 실행하고 그 결과를 쉘 스크립트에서 사용
  * $(명령)
  * `명령`

산술 확장, 변수 확장

* 변수확장
  * 변수의 일부를 잘라내거나, 변수에 문자를 붙일 수 있음
  * 확장 방법
    * ${변수:-default} : 변수가 널이라면 default 값으로 대체
    * ${#변수} : 변수의 길이
    * ${변수%패턴} : 변수 값 끝에서부터 패턴에 일치하는 부분을 탐색하고 짧은 부분 제거
    * ${변수%%패턴} : 변수 값 끝에서부터 패턴에 일치하는 부분을 탐색하고 긴부분 제거
    * ${변수#패턴} : 변수 값 앞에서부터 패턴에 일치하는 부분을 탐색하고 짧은부분 제거
    * ${변수##패턴} : 변수 값 앞에서부터 패턴에 일치하는 부분을 탐색하고 긴 부분 제거
    * ex)
      abc=document/study/makeFile.sh
      a=""
      a:-default : default
      #abc : 26
      abc%/* : document/study (/makeFile.sh 제거)
      abc%%/* : document (/study/makeFile.sh 제거)
      abc#*/ : study/makeFile.sh (document/ 제거)
      abc##*/ : makeFile.sh (document/study/ 제거)

디버깅

* 일반적인 디버깅 방법
  * 쉘 스크립트의 에러가 발생하면 에러를 포함하고 있는 줄의 번호 출력
  * 에러 원인으 알수 없을 때는 echo를 이용해서 변수의 내용을 출력 하거나 대화모드 쉘 실행 방법을 이용해서 의심나는 부분을 실행 해봄
* sh의 실행 옵션을 이용
  * sh -n 스크립트 : 문법 에러만 검사. 명령 실행 안함
  * sh -v 스크립트 : 명령 실행 전에 스크립트 내용 에코
  * sh -x 스크립트 : 명령 실행 후 명령 에코