

Introduction to AI Programming, Spring 2025

Week 2: Linux Basic

Prof. Seunghyun Yoon

(syoon@kentech.ac.kr)

01 명령어 설명을 보는 방법

man

- 원하는 명령어의 설명(manual)을 볼 수 있다. 특정 명령어의 사용방법을 확인할 수 있다.
- 운영체제의 기본 명령어가 아닌경우, -help, --help, 의 paramter로 설명을 제공하는 명령어들도 있다.
- ex) **man find** : find 명령어의 설명을 본다.
- manual이 한눈에 안들어온다면, vi key로 이동, 검색을 할 수 있다.

help parameter (-help, -help)

- 리눅스 기본 명령어도 지원을 하는 경우가 있다. 하지만, 기본 명령어는 man 이 기본이다

02 파일 시스템 이용을 위한 명령어 - 1

리눅스에서 경로를 표시하는 방법

- / root. 시스템의 가장 시작 .. 상위 디렉토리
- ~ home. 로그인한 유저의 home 경로. . 현재 디렉토리
 - 환경변수로 바꿀 수 있다.
- / 디렉토리를 구분하는 구분자 - 이전 위치

02 파일 시스템 이용을 위한 명령어 - 2

pwd

- Print Work Directory 의 약자
- 현재 터미널이 위치한 디렉토리 경로를 볼 수 있다.

/

root 디렉토리

~

현재 로그인한 유저의 home directory

02 파일 시스템 이용을 위한 명령어 - 3

ls

- List Segments 의 약자
- 디렉토리의 모든 파일 정보를 보여줌

ls -al

숨김파일과 파일의 모든 정보를 표시

※ ll(엘엘)이라는 alias로 편하게 사용.

ls -il

파일 또는 디렉토리의 inode number를 표시

※ inode란? 리눅스에서 파일 또는 디렉토리의 고유 식별 부호, 파일 이름이 바뀌어도 유지됨.

02 파일 시스템 이용을 위한 명령어 - 4

cd

- Change Directory 의 약자
- 지정한 디렉토리로 이동
- 기본은 현재 위치부터 상대 경로로 이동, root (/) 부터 모든 경로를 입력하면 절대 경로로 이동

.. 상위 디렉토리를 의미

02 파일 시스템 이용을 위한 명령어 - 5

mkdir

- Make Directory 의 약자
- 새 디렉토리(폴더)를 만든다.
- 기본은 상대경로로 생성.root (/) 부터 모든 경로를 입력하면 절대 경로에 디렉토리를 생성

02 파일 시스템 이용을 위한 명령어 - 6

rm

- Remove
- 지정한 파일 또는 디렉토리를 지운다
- 지울 것인지 한번 물어본다.
- -f 옵션으로 강제로(되묻지 않고) 지운다.
- -r (reculsive) 옵션으로 디렉토리와 디렉토리 안의 모든 내용을 지울 수 있다.
- 인터넷에 올린 질문에 간혹가다 장난으로 **rm -rf** / 명령어를 치라는 답변이 있는데,
이렇게하면 어떻게 될까?

02 파일 시스템 이용을 위한 명령어 - 7

rmdir

- Remove Directory
- 디렉토리를 삭제한다.
- rm -r 명령어로 대신할 수 있다.

02 파일 시스템 이용을 위한 명령어 - 8

df

- Disk Filesystem
- 디스크 공간에 대한 정보를 볼 수 있다. (used, free, mount 정보)
- -h : human readable 옵션과 함께 많이 쓴다.

02 파일 시스템 이용을 위한 명령어 - 9

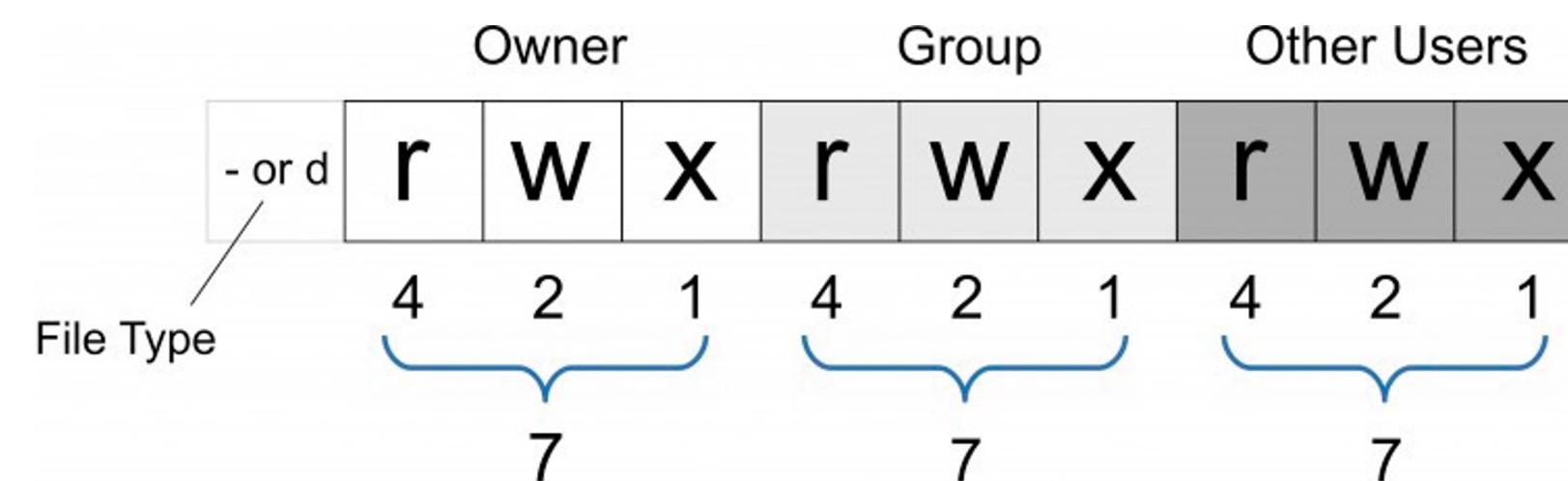
du

- Disk Usage
- 디스크 사용량, 사용율을 볼 수 있다.
- -h : human readable 옵션과 함께 많이 쓴다.
- -s : summary. 해당 경로 하위의 총합을 보여준다.
- -d : max-depth 를 지정해서 해당 depth 만큼 펼쳐서 보여준다.

03 파일 MODE 관리

chmod

- change mode 의 줄임말로, 파일의 접근 권한을 변경할 수 있다.
- 파일 권한의 종류는 r(read), w(write), x(execute) 세 가지 종류가 있다.
- 파일 권한의 범위는 파일 소유자 (u, user), 그룹 (g, group), 그 외 사용자 (o, other)가 있다.
이 모든 것을 포함하는 모든 사용자 (a, all) 도 있다.
- 변환하는 방법에는 추가(+), 제거(-), 지정(=)이 있다.
- user, group, others 각각 3자리의 2진수로 표현된 파일 모드를 지정할 수 있다.



04 파일 MODE 관리 *quiz*

다음 숫자의 mode 는 어떤 권한을 가지고 있을까요?

111

777

644

01 파일 관리를 위한 명령어 - 1

touch

- 빈 파일을 생성

cat

- 파일 내용을 출력
- 다른 명령어와 함께 사용하면 유용

01 파일 관리를 위한 명령어 - 2

head

- 파일 또는 파이프된 데이터의 시작점을 볼 수 있다.

01 파일 관리를 위한 명령어 - 2

tail

- 파일의 마지막 행부터 지정한 행까지 내용을 출력한다.

`tail -f $filename` 실시간으로 append 되는 내용을 확인할 수 있다.
로그를 실시간으로 볼 때 자주 활용한다.

실습

```
while sleep 1; do echo "hello world at $(date +%s)" >> example.log; done &
```

```
ps -ef | grep sleep
```

kill -9 \$pid

01 파일 관리를 위한 명령어 - 3

comm

- compare의 약자
- 두 파일을 라인별로 비교한다.
- 옵션과 함께 사용

cmp

- 두 파일을 바이트 단위로 비교한 결과를 stdout에 프린트

01 파일 관리를 위한 명령어 - 4

diff

- difference 의 약자
- 두 파일을 라인별로 비교해서 차이점만 보여준다.
- symbol로 어떤 차이인지 표시한다.

a add

c change

d delete

01 파일 관리를 위한 명령어 - 5

less

- 터미널 세션에 프린트하지 않고 파일 내용을 볼 수 있다.
- vi 처럼 파일 전체를 여는 것이 아니라 보고있는 부분만 열어서 파일이 큰 경우, vi 에 비해 빠르게 열어 내용을 확인할 수 있다.
- 양방향 탐색이 가능하므로 more 보다 편리하다.

01 파일 관리를 위한 명령어 - 6

In

- 심볼릭 링크를 만든다.
- 심볼릭 링크란?
: 절대 경로 또는 상대 경로의 형태로 된 다른 파일이나 디렉토리에 대한 참조를 포함하고 있는 특별한 종류의 파일
- In -s \$ORIGIN_PATH \$TARGET_PATH

01 파일 관리를 위한 명령어 - 7

alias

- 다른 문자열에 대체하는 단어를 지정
- 주로 복잡한 명령어의 약어를 지칭할 때 쓰임
- 환경변수와는 다름

실습

```
alias hellocmd="echo hello $@"
```

01 파일 관리를 위한 명령어 - 8

cp

- copy 의 약어
- 파일이나 디렉토리 복사
- -r 옵션으로 디렉토리 복사

mv

- move 의 약어
- 파일이나 디렉토리를 이동

01 검색에 사용하는 명령어 - 1

find

- 파일을 검색할때 사용
- `find . -name $filename` 현재 경로에서 \$filename 의 이름을 가진 파일이나 디렉토리를 찾음

01 검색에 사용하는 명령어 - 2

which

- \$PATH 에 등록된 경로 중에서, 주어진 이름의 실행 파일 위치를 찾는다.
- 사용하고 있는 명령어가 설치된 위치를 찾는데 유용

01 검색에 사용하는 명령어 - 3

grep

- 대량의 파일에서 주어진 텍스트 또는 정규 표현식 패턴에 일치하는 텍스트를 찾는 명령어
- 파이프와 함께 다양한 명령어와 조합하여 사용

01 검색에 사용하는 명령어 - 4

sed

- 텍스트를 필터링하거나 변환하는 스트림 에디터
- `sed `s/old_word/new_word/g` target_file` target_file에서 old_word 를 new_world로 모두 교체한 결과를 출력
- vi에서도 `:%s/old_word/new_word/g`를 이용하여 string을 교체할 수 있다.

01 시스템 활용을 위한 명령어 - 1

uname

- 이름, 버전, 기타 시스템 정보를 확인할 수 있습니다.
- `uname -a`

01 시스템 활용을 위한 명령어 - 2

ps

- 현재 실행중인 프로세스를 볼 수 있습니다.
- 어떤 프로세스에 문제가 있는지, 내가 실행한 어플리케이션 프로세스의 상태를 보는데 사용할 수 있습니다.
- 자주 사용하는 옵션
 - ps -ef: process들의 상태를 확인.
→ 형식: UID PID PPID C STIME TTY TIME CMD
 - ps aux: process들의 상태를 cpu, memory 사용률과 함께 확인.
→ 형식: USER PID %CPU %MEM VSZ RSS TT STAT STARTED TIME COMMAND
 - 파이프 후 grep 으로 원하는 프로세스만 볼 수 있다.
→ **ps aux | grep java** 실행하면 java 라는 단어를 명령어나 파라미터로 넣은 프로세스를 확인
- 더 자세한 설명은 다음 링크를 참고: <https://jhnyang.tistory.com/268>

01 시스템 활용을 위한 명령어 - 3

kill

- process 에 signal 을 전달합니다.
- 아래 명령어에서 숫자 또는 괄호안의 글자 중 한가지 사용.

`kill -9(SIGKILL) $pid` 프로세스를 강제로 종료합니다.

`kill -15(SIGTERM) $pid` 또는 `kill $pid` 종료하겠다는 signal을 보낸다. 이 signal을 처리하는 코드를 짤 수도, 무시 할 수도, 막을 수도 있다.

`kill -2(SIGINT) $pid` interrupt 를 의미하는 signal. process를 유지하고 있는 session에서 ctrl+c 를 누르면 이 signal이 전달된다.
→ 대화형 스크립트에서 ctrl+D 를 누르면 세션 종료가 가능한 것은 EOF(end-of-file)을 의미한다. ctrl+D는 signal이 아니다.

- 더 많은 내용은: <https://man7.org/linux/man-pages/man7/signal.7.html>

01 시스템 활용을 위한 명령어 - 5

shutdown

- 시스템 종료

halt

- 시스템 강제 종료

reboot

- 시스템 재시작

01 네트워크 활용을 위한 명령어 - 1

ss

- socket status 의 약어
- 네트워크, 소켓의 사용을 쉽게 확인할 수 있다. 주로 열려있는 포트를 확인할 때 사용
- 내용이 긴 경우 pipe() 해서 less 또는 grep 등과 함께 사용

ss 연결이 맺어진 (STATUS=ESTAB) 소켓 확인

ss -a 열려있는 모든 소켓

ss -l listening 중인 소켓만 표시

ss -t TCP socket 확인, 필요에 따라 -l, -a 등과 함께 사용

ss -u UDP socket 확인, 연결을 맺지 않는 UDP 특성상 항상 -a -l 등과 함께 사용

01 네트워크 활용을 위한 명령어 - 2

ss

ss -s 현재 소켓 상태의 통계

ss -p process name 과 pid 를 함께 표시

ss -e extended output

ss -m memory 사용량을 함께 표시. memory 표시는 skmem 형식 (man ss | grep skmem) 로 확인

01 I/O 관련 명령어 - 1

echo

- 터미널 콘솔에 텍스트를 출력한다.
- echo \$(command)로 다른 명령어의 결과를 확인할 수 있다.
- 파이프와 함께 여러 용도로 사용된다.
예) echo 와 redirection 연결해서 사용

01 I/O 관련 명령어 - 2

clear

- 터미널의 기존 내용을 지운다.

01 I/O 관련 명령어 - 3

history

- 지나간 터미널 명령어 기록을 보여준다.
- 로그아웃하거나 세션을 종료하면 다시 볼 수 없다.
- history에서 확인한 명령어의 숫자를 !number 형태로 쓰면 다시 불러올 수 있다.

01 I/O 관련 명령어 - 4

redirection

- 출력 결과를 파일로 저장
 - > 기존의 파일 내용을 지우고 저장
 - >> 기존 파일 내용 뒤에 저장
 - < 파일의 데이터를 명령어에 입력
 - << 지정한 단어까지의 데이터를 명령어에 입력

01 I/O 관련 명령어 - 5

stdout, stderr

- Process의 표준 입출력을 제어

1> stdout을 지정된 파일에 저장

2> stderr를 지정된 파일에 저장

2>&1 또는 &> stderr 를 stdout에 포함시켜서 저장

> /dev/null 출력 결과를 제거

01 vi에서 이동하기 - 1

명령(이동) 모드

1. 커서가 어디에 있던지, ESC를 누르면 명령(이동) 모드가 됨. 편집할 수 없음.
2. vi를 쓰면 수시로 ESC를 누르는 습관이 필요

라인 숫자 같이 보기

1. :set number 입력
2. :set nonumber 로 number 모드 해제

01 vi에서 이동하기 - 2

커서 이동 명령키

- h** 한문자 왼쪽으로 커서 이동
- J** 한문자 아래쪽으로 커서 이동
- k** 한문자 위쪽으로 커서 이동
- I** 한문자 오른쪽으로 커서 이동
- w** 다음 단어 첫 문자로 커서 이동
- b** 이전 단어 첫 문자로 커서 이동

01 vi에서 이동하기 - 3

행 이동 명령키

gg 첫 행으로 이동

G (command + g) 마지막 행으로 이동

^ 현재 행의 첫 문자로 이동

\$ 현재 행의 마지막 문자로 이동

:5 타이핑한 숫자에 해당하는 행으로 이동

숫자 타이핑하고 shift + G 타이핑한 숫자에 해당하는 행으로 이동.

숫자 타이핑하고 엔터 해당 숫자만큼 아래로 행 이동.

01 vi에서 이동하기 - 4

문자 검색을 이용한 이동 (정규식 이용가능)

/문자열 커서 다음부터 문자열 검색, enter 누르면 커서 이동

?문자열 커서 이전으로 문자열 검색, enter 누르면 커서 이동

n 검색반복

N 역방향으로 검색반복

02 vi에서 편집하기 - 1

입력 관련된 명령어

- | i 현재 커서 위치에 글자 삽입 가능. i 를 누르고 난 이후에 쓰는 글은 커서 위치에 쓰여짐
- | I 현재 줄 처음 글자에 삽입
- a 현재 커서 다음 위치에 추가
- A 현재 줄 마지막 글자에 추가
- o 아랫 줄에 추가
- 윗 줄에 추가
- s 현재 커서 글자 지우고 입력 모드로 전환
- r 현재 커서 글자 지우고 한 글자 입력 받아 바꾼뒤 명령(이동)모드로 들어감

02 vi에서 편집하기 - 2

삭제 관련된 명령어

- ✗ 현재 커서 위치 문자 삭제
- ✗ 현재 커서 위치 이전 문자 삭제
- dw** 현재 커서 위치 단어 삭제, 숫자 dw 로 쓰면 숫자만큼의 단어가 삭제됨
- db** 현재 커서 위치 이전 단어 삭제
- dd** 현재 커서 위치 줄 삭제, 숫자 dd 로 쓰면 숫자만큼의 줄이 삭제됨
- d^** 현재 줄에서 현재 커서 위치 이전 문자열을 마지막 문자까지 삭제
- d0** 현재 줄에서 현재 커서 위치 이전 문자열을 끝까지(0번째 column까지) 삭제
- d\$** 현재 줄에서 현재 커서 위치 이후 끝까지(마지막 column까지) 삭제

02 vi에서 편집하기 - 3

복사 붙여넣기

- y** 복사, w,b, ^, 0, \$ 등 다른 이동 표현과 함께 쓰면 그만큼 복사가 됨.
- yw** 현재 커서 이후 단어 복사, 숫자 yw 로 쓰면 숫자만큼의 단어가 복사됨
- yb** 현재 커서 이전 단어 복사, 숫자 yb 로 쓰면 숫자만큼의 단어가 복사됨
- yy** 현재 줄 복사, 숫자 yy 로 쓰면 숫자만큼의 줄이 복사됨
- p** 복사된 항목을 현재 커서 위치 이후에 붙여 넣기, 삭제된 항목도 붙여 넣기 가능
- P** 복사된 항목을 현재 커서 위치 이전에 붙여 넣기, 삭제된 항목도 붙여 넣기 가능

02 vi에서 편집하기 - 4

undo, redo

u esc를 누른 명령(이동)모드에서 누르면, undo

Ctrl + r esc를 누른 명령(이동)모드에서 누르면, redo

03 vi로 여러파일 동시에 보기 - 1

여러 파일 한번에 열고, 파일을 이동하기

- `vi file1 file2 file3` 명령을 사용하면 동시에 여러 소스코드 파일을 열 수 있음

`:n` 다음 버퍼 파일로 이동

`:N` 이전 버퍼 파일로 이동

`:ls` 현재 열려있는 버퍼의 리스트를 보여줌

`:b숫자` 숫자에 해당하는 버퍼로 이동

`:bd숫자` 숫자에 해당하는 버퍼를 삭제

`:bw` 현재 버퍼를 삭제

03 vi로 여러파일 동시에 보기 - 2

Vi안에서 새로운 파일 열기

:e [tab] 또는 \$filename

해당 파일을 현재 vi 창에서 새로운 버퍼로 연다

:cd [tab] 또는 \$filename

(안보이지만) 해당 디렉토리로 이동 한다. :e 로 다음 파일을 찾을 때 적용

03 vi로 여러파일 동시에 보기 - 3

화면 분할해서 보기

- :split 현재 열린 파일과 같은 파일을 수평으로 분할해서 하나 더 연다
- :split \$file 새로운 파일을 현재 위치에 열고 현재 열린 파일은 수평으로 분할해서 연다
- :vsplit 현재 열린 파일과 같은 파일을 수직으로 분할해서 하나 더 연다
- :vsplit \$file 새로운 파일을 현재 위치에 열고 현재 열린 파일은 수직으로 분할해서 연다

03 vi로 여러파일 동시에 보기 - 4

화면 분할 상태에서 윈도우 이동하기

Ctrl+w, j 아래 윈도우로 이동

Ctrl+w, k 위 윈도우로 이동

Ctrl+w, l(엘) 오른쪽 윈도우로 이동

Ctrl+w, h 왼쪽 윈도우로 이동

03 vi로 여러파일 동시에 보기 - 5

화면 분할 상태에서 창 크기 조절하기 - 좌우

Ctrl+w, = 모두 균일한 상태로 이동

Ctrl+w, > 오른쪽으로 1칸 확장

Ctrl+w, \$num> 오른쪽으로 \$num 칸 확장

Ctrl+w, < 왼쪽으로 1칸 확장

Ctrl+w, \$num< 왼쪽으로 \$num 칸 확장

03 vi로 여러파일 동시에 보기 - 6

화면 분할 상태에서 창 크기 조절하기 - 위아래

Ctrl+w, = 모두 균일한 상태로 이동

Ctrl+w, + 위로 1칸 확장

Ctrl+w, \$num+ 위로 \$num 칸 확장

Ctrl+w, - 아래로 1칸 확장

Ctrl+w, \$num- 아래로 \$num 칸 확장

Ctrl+w, _ 해당 윈도우가 위아래로 모든 칸을 차지

03 vi로 여러파일 동시에 보기 - 7

추천 프로그램

tmux Mac, linux 사용자는 tmux 라는 프로그램을 설치하면, 터미널을 여러개를 동시에 열고, 화면분할과 전환 등의 기능으로 동시에 여러 작업을 하나의 터미널 세션에서 쉽게 작업할 수 있다.

01 환경변수란?

환경변수는 운영체제 수준에서 선언하는 변수이다.

운영체제 해당 환경에서 실행되는 프로세스가 모두 참조할 수 있다.

대표적으로 다음과 같은 경우에 사용한다.

- 변수에 자주사용하는 경로를 저장한다. 예) HOME
- 기존에 있는 변수를 이용한 새로운 변수를 저장한다. 예) PATH=\$PATH:newpath
- 프로세스가 구동중에 참조할 값을 미리 환경변수에 할당하고 프로세스를 실행한다.
- 여러개의 프로세스가 참조해야하는 값을 환경변수에 할당한다. 예) \$LD_LIBRARY_PATH

02 환경변수를 임시로 선언하는 법

`export 환경변수명=값` 값을 환경변수로 임시 선언

이 경우 시스템 재부팅 또는 로그아웃을 하면 환경 변수 값이 사라지게 된다.

03 환경변수를 유저레벨로 선언하는 법

환경 변수를 특정 유저에게만 영구적으로 적용하고 싶은 경우에는 `~/.bash_profile` 파일을 수정한다.

`~/.bash_profile` 은 user 가 처음 login 할 때 수행된다.

단, bash shell로 접속했을 때만 동작한다. sh 또는 zsh로 접속하면 동작하지 않는다.

- `vi ~/.bash_profile`

04 환경변수를 영구히 선언하는 법

환경 변수를 모든 유저에게만 영구적으로 적용하고 싶은 경우에는
`/etc/profile` 파일을 수정한다.

- `sudo vi /etc/profile`
- `export LECTURE="data-engineering"`
- `echo $LECTURE`

05 \$PATH의 이해

운영체제가 명령어의 실행파일을 찾는 경로

- 절대/상대 경로 없이, 단독으로 명령어를 수행할 수 있다는 것은 해당 명령어의 실행파일이 운영체제의 \$PATH에 등록된 디렉토리들 중에 포함되어 있다는 의미이다.
- which가 명령어 파일 위치를 찾는 원리
- 내가 만든 실행파일의 위치도 \$PATH에 경로를 등록하면, 경로를 매번 입력하거나 찾지 않아도 바로 쓸 수 있다는 의미.

01 Shell script란?

Shell script는 unix shell의 command line interpreter가 실행할 수 있는 명령어(리스트)를 의미한다.

02 Shell script 선언하는 법

파일 맨 위에 다음과 같은 표시가 있다면 shell script 파일이라고 할 수 있다.

#!/bin/sh /bin/sh (bourne shell)를 사용하는 shell script

#!/bin/bash /bin/bash 를 사용하는 bash shell script

#!/bin/zsh /bin/zsh 를 사용하는 zshell shell script

#! 는 shebang 이라고 읽는다.

03 Shell script 실행하는 법

- `./$FILENAME` 을 이용하여 실행
- 실행이 안된다면 ?
→ 로그인한 user에게 `+x` 권한이 있는지 확인

04 가장 많이 쓰이는 shell script 는?

sh 또는 bash shell을 가장 많이 쓴다. Mac의 monetary 부터는 zsh이 기본 shell이 되었다. Docker 환경에서 이미지 사이즈를 줄이기 위해 alpine linux를 많이 사용하는데, alpine 리눅스에는 bash shell이 기본으로 설치되어있지 않기 때문에, 기본 shell (sh) (Bourne shell)의 활용이 더욱 중요해졌다. 따라서 이후 실습에서는 기본 쉘 스크립트 기준으로 실습을 진행한다. bash는 sh와 거의 비슷하지만, 몇 가지 편의 기능, 문법이 더해졌다고 생각하면 좋다.

01 변수

- 영문, 숫자, _ 만 사용 가능
- Unix shell 의 변수 명은 대문자로 작성하는 것이 convention 이다
- **변수명=변수값** 으로 변수 선언 가능

\$변수명 변수 사용

readonly **변수명** 읽기전용 변수로 선언 (읽기 전용 변수 값을 바꾸려고 시도하면 에러 발생)

unset **변수명** 변수 할당 해제

- Script 내에서 선언한 변수 뿐만 아니라, 환경 변수, 쉘 변수(접속한 shell에서 export로 선언한 변수)도 \$변수로 읽어올 수 있다.

02 특수목적 변수

\$\$ 현재 shell 의 프로세스 아이디

\$0 현재 script 의 파일 이름

\$n script 실행시 넘겨준 n번째 인자

\$# script 에 넣어준 인자의 개수

\$* 모든 인자를 “” 로 감싸서 반환

\$@ 각 인자를 “” 로 감싸서 반환

\$? 마지막으로 실행된 명령어의 종료 상태

\$! 마지막 백그라운드 명령어의 프로세스 아이디

03 조건문 - if

If는 expression 을 이용해서 자유롭게 조건을 놓는다.

```
If [ condition ]; then  
    ${if script}  
elif [ condition ]; then  
    ${elif script}  
else  
    ${else script}  
fi
```

- if 는 항상 fi 로 닫는다.
- 조건 뒤에는 ; then 으로 다음 수행을 알려준다.
- elif, else 는 선택적으로 사용한다.

04 조건문 - case

case 는 미리 정해진 pattern의 선택지를 사용한다.

```
case word in  
pattern1)  
Statement(s) to be executed if pattern1 matches  
;;  
pattern2)  
Statement(s) to be executed if pattern2 matches  
;;  
*)  
Default condition to be executed  
;;  
esac
```

- case 는 항상 esac 로 닫는다.
- case 대상 변수 뒤에는 in 과 pattern 이 와야한다.
- Pattern 은 글자 그 자체, 또는 regular expression 이 가능하다.
- 위에서부터 차례대로 확인한다.
- *) 는 default, 위의 어떤 조건도 아닐 때 수행한다.

05 기본 연산자 - 0

Bourne shell에는 arithmetic operations이 없기 때문에 외부 프로그램을 이용해야 한다.
대표적으로 awk / expr이 있다.

ex)

```
val = `expr 2 + 2`  
echo "Total value : $val"
```

> Total Value : 4

05 기본 연산자 - 1

산술 연산자

- + 더하기 `expr \$a + \$b` 로 사용한다 = 값 할당 a = \$b 로 사용하면 b의 값이 a에 할당된다
 - 빼기 `expr \$a - \$b` 로 사용한다 [\$a = \$b] 값이 같은 경우에 true, 값 비교
 - * 곱하기 `expr \$a * \$b` 로 사용한다 [\$a != \$b] 값이 다른경우에 true
 - / 나누기 `expr \$a / \$b` 로 사용한다 ※ 이때 반드시 [] 대괄호와 값 사이에 space 가 있어야함.
 - % 나머지 연산자 `expr \$a % \$b` 로 사용한다

05 기본 연산자 -2

관계 연산자

-eq 두 값이 같은 경우 true

-ne 두 값이 다른 경우 true

-gt 왼쪽 값이 오른쪽 값보다 큰 경우 true 반환

-lt 왼쪽 값이 오른쪽 값보다 작은 경우 true 반환

-ge 왼쪽 값이 오른쪽 값보다 크거나 같은 경우 true 반환

-le 왼쪽 값이 오른쪽 값보다 작거나 같은 경우 true 반환

05 기본 연산자 -3

부울 연산자

! 논리적인 부정

-o OR 연산자

-a AND 연산자

06 반복문 - 1

`while ... do ... done` 을 이용해 반복문을 선언할 수 있다.

ex)

`a = 0`

`while [“a” -lt 10]`

`do`

`echo -n “$b”`

`a = `expr $a + 1``

`done`

07 Quoting Mechanism

Metacharacters

- ? [] ` “ \\$; () | ^ < > 는 metacharacters 로 shell 에서 특수한 목적으로 사용된다.
- ‘ ’ 을 이용하면 위의 특수 목적 문자를 출력할 수 있다.
- 내부적으로 ‘ ’ 를 사용해야 하는경우 “ ” 를 이용하여 ‘ ’ 을 출력할 수 있다.
- `` 를 이용하여 `` 내부의 shell 명령어를 실행할 수 있다.

예를 들어 echo <-\$1500.**>; (update?) [y|n] 을 표현하는 방법으로 아래 두가지를 사용할 수 있다.

- echo \<-\$1500.**\>\; \\(update\\?\\) \\[y\\|n\\]
- echo '<-\$1500.**>; (update?) [y|n]'

08 Shell Substitution

echo command 와 함께 사용되는 escape sequence

echo -e backslash escape 을 적용할 수 있다

\\" backslash

\a alert

\n 줄 바꿈

\b backspace

\r 캐리지 리턴 (맨 앞으로 이동)

\c 줄바꿈 삭제

\t 가로 방향으로 tab 만큼 띄움

\f form 양식

\v 세로 방향으로 tab 만큼 띄움

09 function

function_name () {} 함수 선언

function_name 함수 사용

function_name param1 param2 함수에 파라미터 param1 param2 전달

함수 정의 및 사용 예시

```
Hello () {  
    echo "Hello World $1"  
    return 10  
}  
Hello FastCampus
```