

리눅스 기본교육

2024.11.11

다원컴퓨팅(주)
윤민수

DAWON
COMPUTING

CONTENTS

1. 리눅스 소개
2. 리눅스 기본명령어
3. 터미널 세션관리
4. 쉘 설정 및 사용방법

1. 리눅스 소개

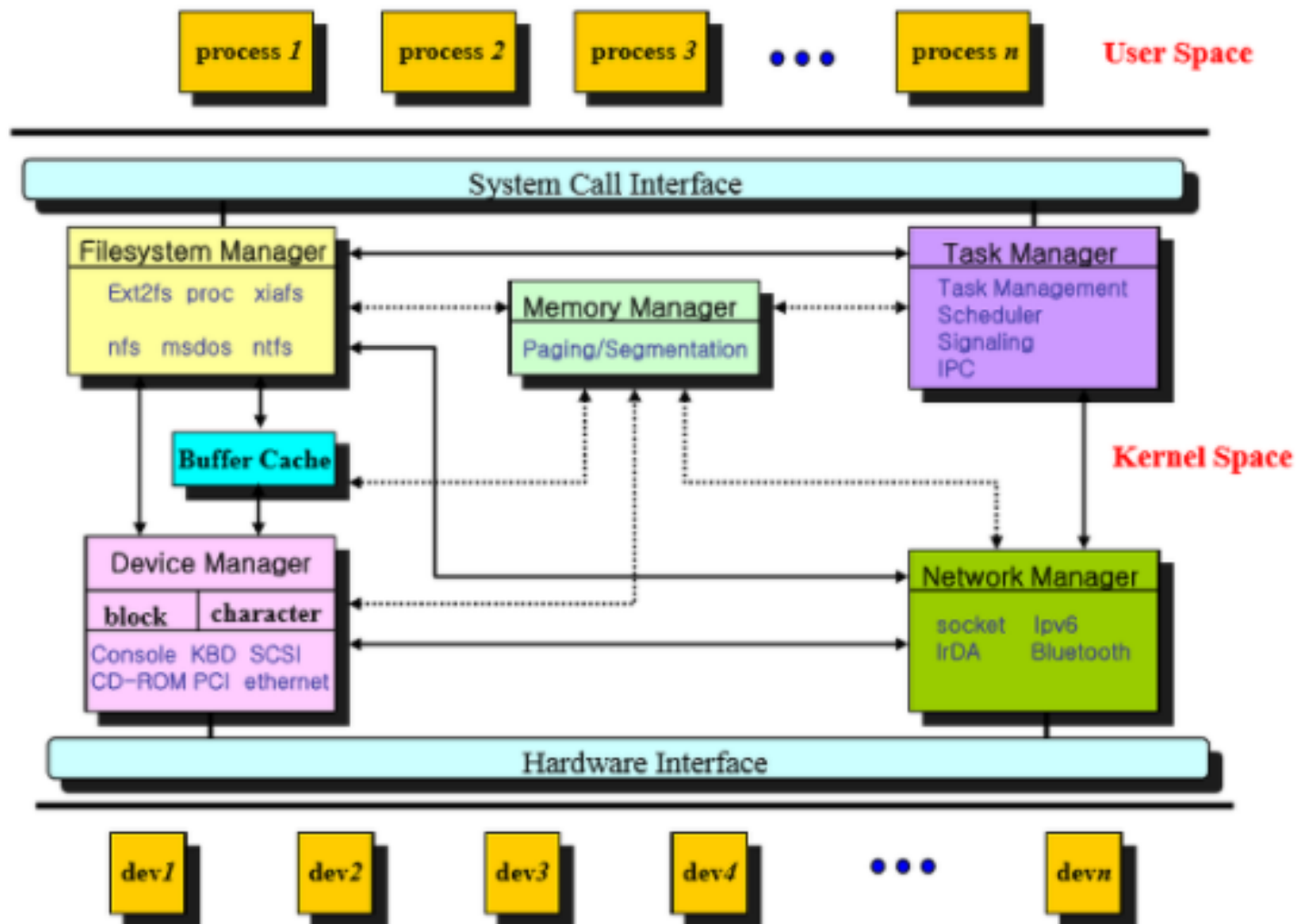
1. 리눅스 소개

리눅스 특징

- UNIX 를 모델로 하여 만들어진 오픈소스 기반 운영체제
- 다중 사용자, 다중 처리 시스템
- 다양한 하드웨어 지원
- 여러 파일 시스템을 동시에 지원
- 안정적인 네트워크 기능 제공
- 명령어로 모든 기능을 수행

1. 리눅스 소개

리눅스 시스템 아키텍처

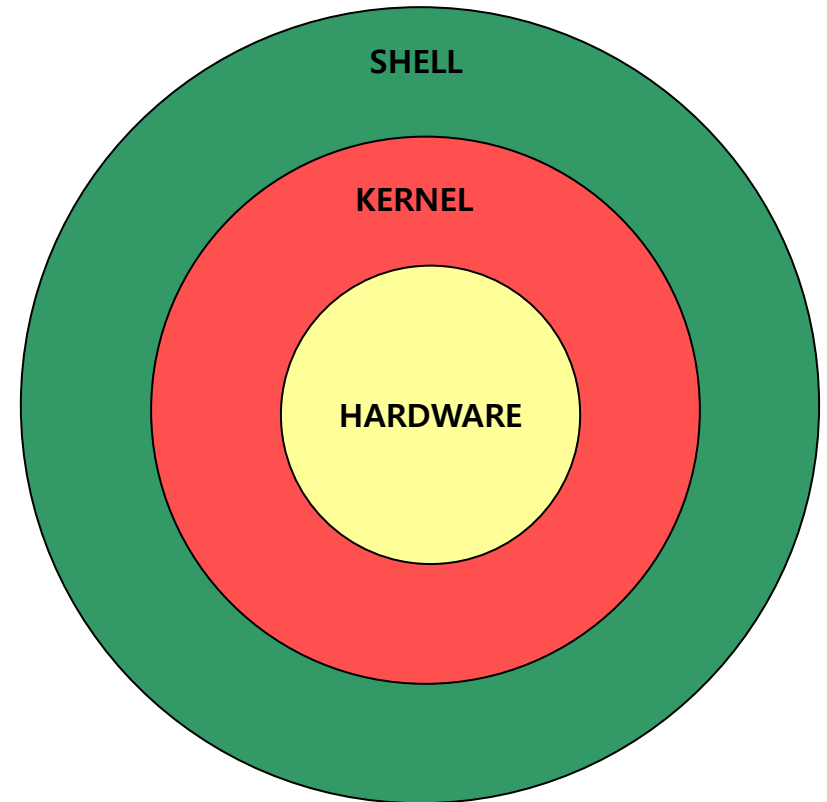


(Source : Linux Kernel Internals)

1. 리눅스 소개

리눅스 구조

- Kernel
 - 운영체제의 핵심
 - 시스템 자원들을 관리하는 부분
- Shell
 - 명령어 해석기
 - 사용자와 시스템 간의 명령을 해석/전달
 - sh, bash, ksh, csh 등이 있음
- 여러 유저가 하나의 시스템을 공유하여 사용할 수 있도록 설계



2. 리눅스 기본명령어

2. 리눅스 기본명령어

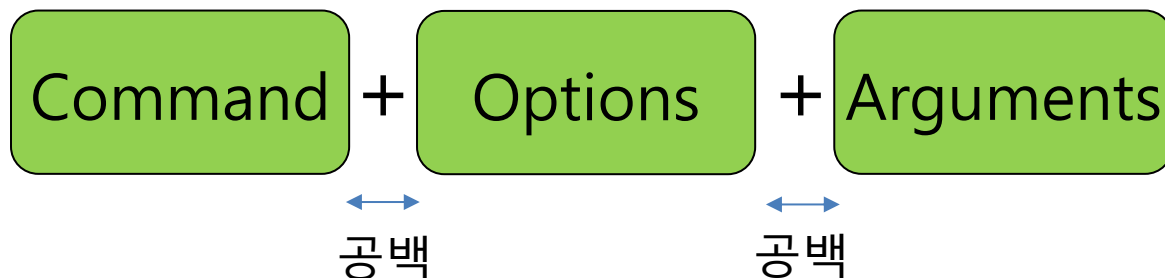
리눅스 명령어 구조

- Example

\$ ls

\$ ls -al

\$ ls -al /home



- Command

- 일반적으로 소문자
- 무엇을 할 것인가

- Option

- 명령어 출력 향상
- 필요에 맞춰 결과를 출력
- 여러 옵션 조합 가능 (공백없이)

- Argument

- 하나 이상의 argument를 가질 수 있다.
- argument가 두개 이상인 경우 공백으로 구분

2. 리눅스 기본명령어

Man Page

- 시스템에서 제공하는 도움말 (man page)
- 기본적으로 command 마다 man page 가지고 있음
 - q: 종료
 - b: 이전 페이지
 - f / space: 다음페이지

```
$ man ls
```

```
LS(1)
```

```
User Commands
```

```
LS(1)
```

```
NAME
```

```
ls - list directory contents
```

```
SYNOPSIS
```

```
ls [OPTION]... [FILE]...
```

```
DESCRIPTION
```

```
List information about the FILES (the current directory by default). Sort entries
```

```
alphabetically if none of -cftuvSUX nor --sort is specified.
```

2. 리눅스 기본명령어

기본 명령어

Command	Description
ls	파일의 목록 \$ ls \$ ls -alh ./
cd	작업 디렉토리 위치 이동 \$ cd /home \$ cd tmp \$ cd ..
pwd	현재 작업 디렉토리 위치 확인 \$ pwd
which	PATH에 등록된 영역에 존재하는 특정 명령어의 위치 확인 \$ which ls
id	자신의 UID/GID 확인 \$ id
whoami	자신의 계정명 확인 \$ whoami

2. 리눅스 기본명령어

기본 명령어

Command	Description
du	디렉토리의 디스크 사용량 확인 \$ du -sh ./
df	파일시스템 사용량 확인 \$ df -h
date	시스템에 설정된 시간 확인 \$ date \$ date -d "yesterday" \$ date +%Y%m%d%H%M
cal	달력 \$ cal \$ cal -3
who	시스템 사용중인 사용자 리스트 확인 \$ who

2. 리눅스 기본명령어

기본 명령어

Command	Description
whereis	which와 유사, man page 위치, 소스 등의 위치도 확인 가능 \$ whereis ls
mkdir	새로운 디렉토리 생성 \$ mkdir idh \$ mkdir -p ./idh/src
rm	디렉토리/파일 삭제 (주의필요) \$ rm ./file \$ rm -rf directory
cp	디렉토리/파일 복사 \$ cp file1 file2 \$ cp -r dir1 dir2

2. 리눅스 기본명령어

기본 명령어

Command	Description
ln	링크파일 생성 \$ ln file1(원본파일) file2(링크파일 명) \$ ln -s file1(원본파일) file2(링크파일 명)
find	디렉토리/파일 찾기 \$ find ./ -name test.txt \$ find /home -type f -atime -3 \$ find /home -inum xxxxx -exec rm -rf {} \;
grep	화면 출력 내용 또는 파일에서 문자열 찾기 \$ grep test * \$ grep -i test *
history	기존에 사용한 명령어 이력 조회 \$ history

2. 리눅스 기본명령어

기본 명령어

Command	Description
cat	파일의 전체내용 출력 \$ cat test.txt
touch	빈 파일의 생성, 파일 액세스 시간 변경 \$ touch test.txt
more	화면에 출력되는 내용을 페이지 단위로 출력 \$ more test.txt
less	more 보다 기능이 더 많음 \$ less test.txt
ps	프로세스 확인 \$ ps \$ ps aux \$ ps -ef grep -i python
pidof	특정 프로세스의 PID 확인 \$ pidof ssh

2. 리눅스 기본명령어

기본 명령어

Command	Description
kill	프로세스 종료 \$ kill <PID> \$ kill -9 <PID>
clear	터미널 내용 정리
mv	디렉토리/파일 이름 변경 또는 다른 위치로 이동 \$ mv file1 file2 \$ mv dir1/file1 dir2
echo	텍스트를 화면에 출력 \$ echo "Hello" \$ echo \$HOME
diff	2개의 텍스트 파일 내용을 비교 \$ diff file1 file2

2. 리눅스 기본명령어

기본 명령어

Command	Description
head	화면 출력결과에서 처음부분을 보여줌 (기본 10줄) \$ head file1 \$ head -n 30 file1
tail	화면 출력결과에서 끝부분을 보여줌 (기본 10줄) \$ tail -n 30 file1 \$ tail -f /var/log/message
wc	파일의 라인 수 또는 글자 수 출력 \$ wc -l test.txt \$ wc test.txt
sleep	지정한 시간(초) 동안 대기 \$ sleep 100
watch	명령어를 반복해서 실행 (기본 2초 간격) \$ watch who
file	파일의 종류를 확인 \$ file test.txt

2. 리눅스 기본명령어

기본 명령어

Command	Description
Ctrl+Z (^Z)	실행중인 명령어를 잠시 멈춤 \$ sleep 1000 ^Z \$ fg % [num]
Ctrl+C (^C)	실행중인 프로세스를 종료
jobs	백그라운드로 실행중인 작업의 목록 \$ jobs
&	실행하고자 하는 명령어를 백그라운드로 실행 \$ sleep 1000 &
fg	명령어를 포그라운드로 실행 \$ fg \$ fg %1

2. 리눅스 기본명령어

파일관리 (Permission)

- 기본적으로 파일은 소유권을 가지고 있는 계정만 접근 가능(root 제외)
- 파일의 permission을 변경하여 다른 사람과 파일 공유 가능

Command	Description
chmod	퍼미션 변경 \$ chmod 755 <directory/file> \$ chmod u+rw,g+rw,o+rw <directory/file>
chown	소유자 변경 \$ chown root:root <directory/file>
chgrp	그룹 변경 \$ chgrp sys <directory/file>

2. 리눅스 기본명령어

파일관리 (Permission)

- 퍼미션

File	User			Group			Other		
	4	2	1	4	2	1	4	2	1
-	r	w	x	r	w	-	r	-	-

- : 파일의 종류
- rwX : 사용자는 읽기, 쓰기, 실행 가능
- rw- : 그룹은 읽기, 쓰기 가능
- r-- : 기타 사용자는 읽기 가능

- r : 읽기 (4)
- w : 쓰기 (2)
- x : 실행 (1)

2. 리눅스 기본명령어

파일관리 (Permission)

- chmod 명령어 사용 예

Command	Description
chmod u+rwx test.txt	소유자에게 읽기/쓰기/실행 권한 부여
chmod 700 test.txt	소유자에게 읽기/쓰기/실행 권한 부여, 그외는 접근 금지
chmod u+rwx,g-x,o-rwx test.txt	소유자는 읽기/쓰기, 그룹은 실행 권한 제외, 기타 접근 금지
chmod 777 test.txt	모든 사용자 읽기/쓰기/실행 권한 부여
chmod a+rwx, test.txt	chmod 777 test.txt 와 동일

2. 리눅스 기본명령어

실 습

1) 현재 계정의 UID/GID 확인

```
$ id
```

2) 새로운 파일 생성

```
$ touch my_test.txt
```

3) 파일 생성 확인

```
$ ls -l
```

4) 임시 디렉토리 생성

```
$ cd
```

```
$ mkdir -p ./temp_dir/test_dir
```

5) 현재 작업 위치 확인

```
$ pwd
```

6) 파일 복사

```
$ cp ./my_test.txt ./temp_dir/test_dir
```

7) 파일 복사결과 확인

```
$ cd ./temp_dir/test_dir
```

```
$ ls -l
```

8) 파일 이름 변경

```
$ mv my_test.txt my_test.sh
```

9) 파일 권한 변경 및 확인

```
$ chmod 700 my_test.sh
```

```
$ ls -l my_test.sh
```

```
$ chmod +x my_test.sh
```

```
$ ls -l my_test.sh
```

1) 현재 작업위치 확인

```
$ pwd
```

2) 상위 디렉터리 이동

```
$ cd ..
```

```
$ pwd
```

```
$ ls -l
```

3) 홈디렉터리로 이동

```
$ cd
```

```
$ cd ~
```

4) 파일 검색

```
$ find . -name "my_test.*"
```

5) 파일 삭제

```
$ rm ./my_test.txt
```

6) 링크파일 생성

```
$ ln -s ./temp_dir/test_dir/test.sh ./my_test.sh
```

```
$ ls -l
```

7) 파일 정보 확인

```
$ file ./my_test.sh
```

2. 리눅스 기본명령어

파이프라인 / 리다이렉션

- 파이프 (pipe) : 프로세스와 명령어를 연결해 주는 기능
 - `command1 | command2 | command3`
 - `ps -ef | grep nobody`
 - `ps` 실행한 결과를 `grep` 명령어의 입력으로 전달
- 리다이렉션 : 출력의 방향을 재 지향하는 기능
 - `>` : 명령어 출력을 파일로 전달, 파일이 있으면 덮어쓰기, 없으면 생성
 - `ls -al > list.txt`
 - `>>` : 명령어 출력을 파일로 전달, 파일이 존재하면 덧붙여 쓰기
 - `ls -al >> list.txt`
 - `<` : 파일을 명령어 입력으로 전달
 - `grep word < filename`

2. 리눅스 기본명령어

파일 압축 / 압축 해제

Command	Description
tar	여러 파일들을 하나의 파일로 묶기/풀기 (확장자 : tar) \$ tar cvf test.tar a.txt b.txt \$ tar xvf test.tar
zip / unzip	파일 압축 명령어 (확장자 : zip) \$ zip test.tar
gzip / gunzip	파일 압축 명령어 (확장자 : gz) \$ gzip test.tar \$ gzip -d test.tar.gz
bzip2 / bunzip2	gzip 보다 압축 효율이 좀더 좋음 (확장자 : bz) \$ bzip2 test.tar

2. 리눅스 기본명령어

파일 압축 / 압축 해제 (tar 옵션)

Option	Description
-c	tar 파일 생성 (묶기)
-x	tar 해제 (풀기)
-d	tar 파일과 해당 파일 시스템간의 차이점 확인하고자 할 때
-r	tar 파일에 다른 파일들을 추가할 때
-t	tar 파일에 있는 파일들을 확인할 때
-f	tar 파일을 지정
-p	원래 permission 유지
-v	Verbose
-z	gzip 으로 압축/압축 해제
-j	bzip2으로 압축/압축 해제
-J	xz 으로 압축/압축 해제
-h	링크가 가리키는 파일을 묶기 (파일크기 커짐)

2. 리눅스 기본명령어

파일 압축 / 압축 해제 (예)

Command	Description
tar cvf test.tar a.txt b.txt ...	a.txt, b.txt 파일을 하나의 test.tar 파일로 묶으면서 목록 출력
tar zcf test.tgz a.txt b.txt	a.txt, b.txt 파일을 하나의 tar 파일로 묶은 후 gzip 으로 압축 .tgz는tar로 묶은 후 gzip 으로 압축되었음을 표시
tar xzf test.tgz a.txt b.txt - remove-files	파일 압축 후 a.txt, b.txt 파일 삭제 (공간 확보)
tar xzf test.tgz	test.tgz 파일을 gunzip 으로 압축 해제 및 tar 해제
tar tvf test.tar	test.tar 안에 있는 파일의 목록만 출력 (압축/해제 안함)

2. 리눅스 기본명령어

파일관리 (vi)

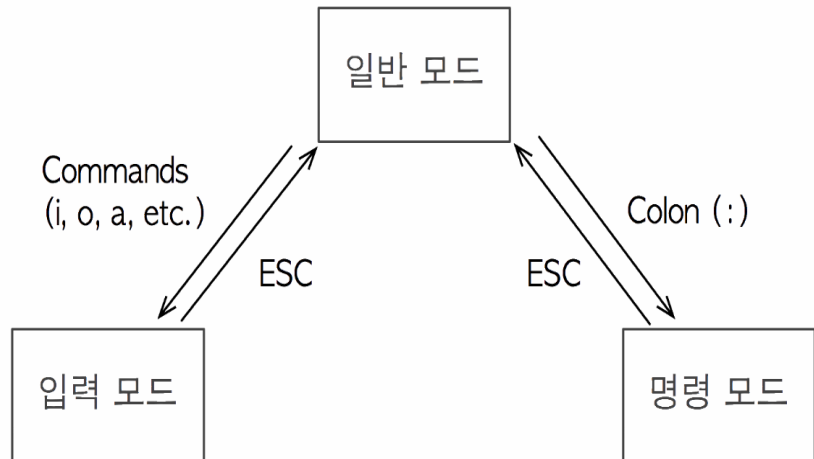
- 파일 열기 / 수정

\$ vi file

\$ vim file

Mode

- normal mode (일반 모드)
 - vi 안에서 삭제, 편집 등을 위한 명령어 사용
 - insert mode (입력 모드)
 - 글자 입력 모드 (i)
 - 일반 모드로 변경시 "[ESC]키"
 - command mode (명령 모드)
 - 일반 모드에서 : (콜론) 입력
 - vim의 종료, 파일 저장 등
- vimtutor
 - vim (vi) 관련 도움말



2. 리눅스 기본명령어

파일관리 (vi)

Command	Description
h / ←	커서를 왼쪽으로 한 글자 이동
j / ↑	커서를 위로 한 줄 이동
k / ↓	커서를 아래로 한 줄 이동
l / →	커서를 오른쪽으로 한 글자 이동
Enter	커서를 다음 줄의 첫번째 칸으로 이동
Esc :q! Enter	파일을 저장하지 않고 vim 종료
Esc :wq! Enter	파일을 저장하고 종료
Esc :! Enter	잠시 셸 환경으로 돌아감, 셸 환경에서 exit 실행하면 다시 vim 환경으로 돌아옴
x	커서 위치의 글자 지우기
Esc (숫자+)yy	(현재부터 숫자만큼 라인 수+) 커서 위치의 줄을 복사
Esc (숫자+)dd	(현재부터 숫자만큼 라인 수+) 커서 위치의 줄을 잘라내기
Esc p	커서 위치의 아래 줄에 붙여넣기
i text Esc	커서 앞에 text 입력

2. 리눅스 기본명령어

파일관리 (vi)

- 기본 단위 커서 이동

Key	Description
h / ←	커서를 왼쪽으로 한 글자 이동
j / ↑	커서를 위로 한 줄 이동
k / ↓	커서를 아래로 한 줄 이동
l / →	커서를 오른쪽으로 한 글자 이동
Enter	커서를 다음 줄의 첫번째 칸으로 이동

- 화면 내 커서 이동

Key	Description
H	커서를 화면의 맨 위쪽으로 이동
M	커서를 화면의 중간 줄로 이동
L	커서를 맨 아래 줄로 이동

2. 리눅스 기본명령어

파일관리 (vi)

Key	Description
<code>[Esc] :1,\$s/ab/cd/g [Enter]</code>	파일의 첫번째 줄부터 끝까지 ab를 찾아서 cd로 변경
<code>[Esc] :1,\$s/^ //g [Enter]</code>	문장의 첫글자 공백 삭제
<code>[Shift] v [↑]/[↓]</code>	커서 위치부터 줄 단위 선택
<code>[Ctrl] v [↑]/[↓]/[←]/[→]</code>	커서 위치부터 글자 단위 선택
<code>[Esc] :!ls [Enter]</code>	ls 명령어 실행, Enter 누르면 vim 화면으로 되돌아옴
<code>[Esc] :r !command [Enter]</code>	커서 위치부터 command의 결과를 삽입
<code>[Esc] set num [Enter]</code>	줄 번호 표시
<code>[Esc] set nonum [Enter]</code>	줄 번호 표시 해제
<code>[Esc] u</code>	실행 취소(undo)
<code>[Esc] Ctrl+r</code>	다시 실행 (redo)

2. 리눅스 기본명령어

실 습

1) 홈디렉토리 이동

```
$ cd ~
```

2) 파일 내용 수정

```
$ vi my_test.sh
```

i (입력모드)

```
-----  
#!/bin/bash
```

```
hostname
```

```
echo "hostname=$HOSTNAME"
```

```
-----  
[esc] (입력모드 종료)
```

```
:wq! (저장 후 vi 종료)
```

3) 파일 실행 및 결과확인

```
$ chmod u+rx ./my_test.sh
```

```
$ ./my_test.sh
```

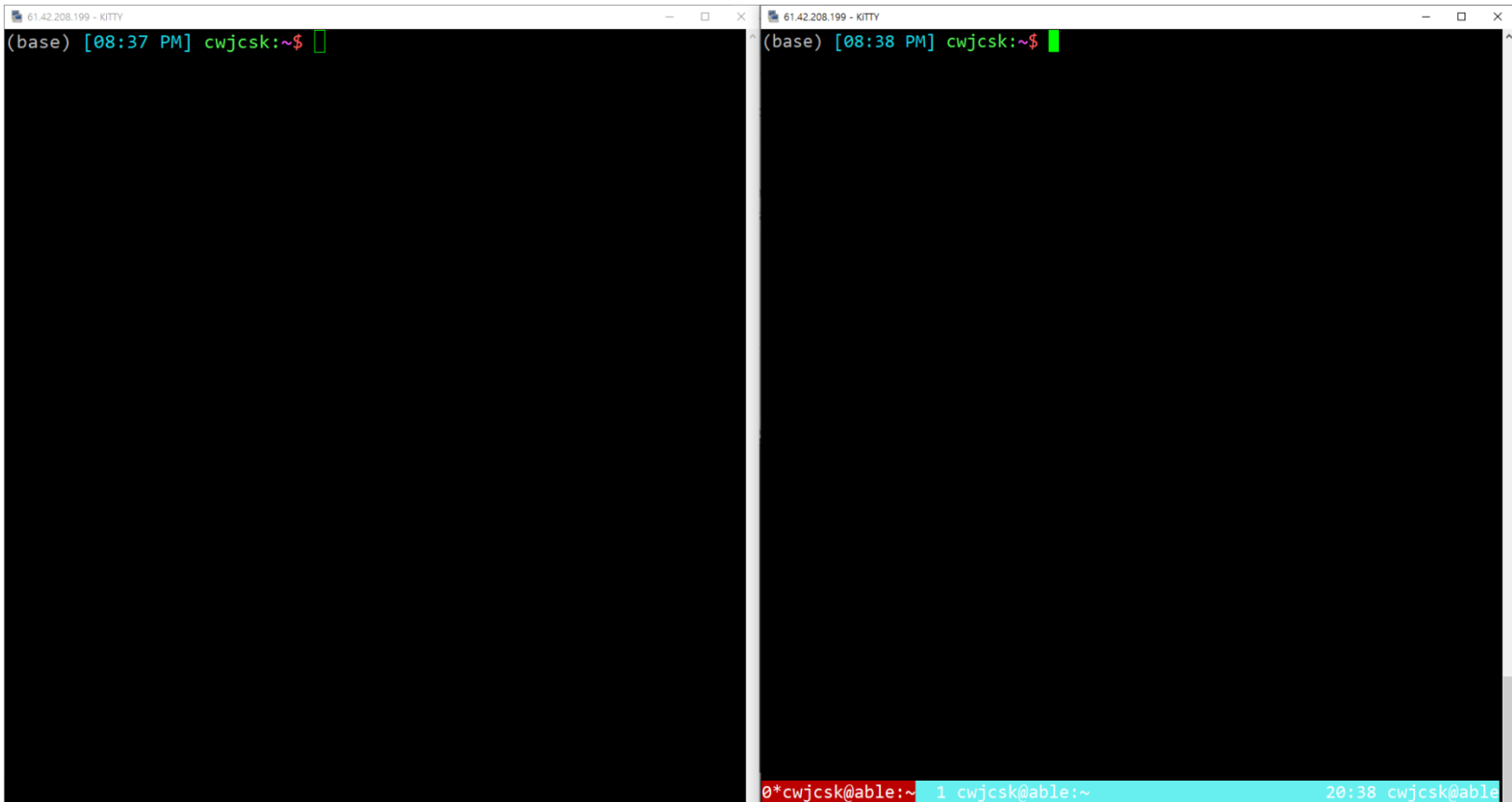
3. 터미널 접속관리

3. 터미널 접속관리

screen

독립적으로 동작하는 가상 터미널을 띄워주는 것을 의미합니다. 즉, 백그라운드로 동작하는 가상 터미널

장점: 가상터미널에서 명령어를 실행시키고 터미널을 꺼도, 명령어가 백그라운드로 계속 실행상태로 유지
screen 명령을 이용해서 다시 접속하면 해당 터미널 그대로 작업을 이어갈 수 있음



3. 터미널 접속관리

screen

Option	Description
screen -S [스크린 명]	screen에 이름을 지정하며 진입
screen -R [스크린 명]	screen이 존재한다면 다시 진입(Reattach)하고, screen이 없다면 해당 스크린 이름을 만들며 진입.
screen -ls (screen -list)	현재 존재하는 스크린 리스트 출력
screen -x [스크린 명]	실행 중인 스크린에 다시 진입(Reattach). 서로 다른 사용자가 같은 session 연결시, 화면 공유
screen -S [스크린 명] -X quit	해당 스크린 종료 (해당 스크린 삭제됨)

내부명령어	Description
Ctrl+a, d	현재 스크린으로부터 탈출(Deattach). (스크린은 꺼지지 않고 여전히 동작 중)
Ctrl+a, c	스크린에서 새창 띄우기
Ctrl+a, 숫자	해당 번호의 스크린으로 이동
Ctrl+a, n	다음 창으로 이동 (Ctrl+a, space와 동일)
Ctrl+a, p	이전 창으로 이동 (Ctrl+a, Backspace와 동일)

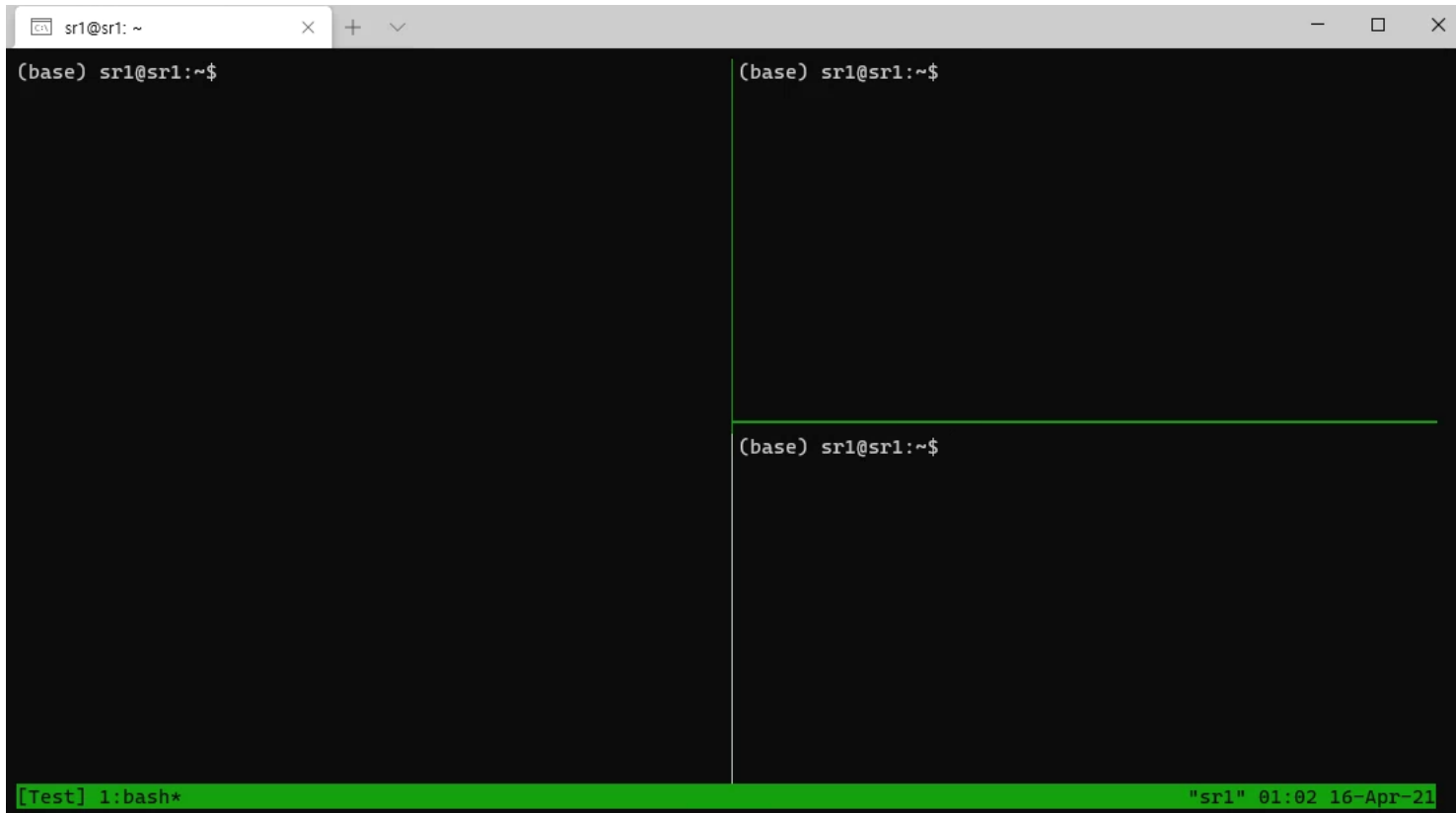
3. 터미널 접속관리

TUMX (Terminal Multiplexer)

하나의 터미널 콘솔에서 여러 개의 터미널을 생성하고 관리할 수 있는 기능을 제공 (screen 과 유사한 기능)

장점: * 터미널에서 명령어를 실행시키고 터미널을 꺼도, 명령어가 백그라운드로 계속 실행상태로 유지
tmux 명령을 이용해서 다시 접속하면 해당 터미널 그대로 작업을 이어갈 수 있음

* 생성된 세션은 여러 유저가 동시에 접속하여 함께 작업의 진행이 가능



```
sr1@sr1: ~
(base) sr1@sr1:~$
(base) sr1@sr1:~$
(base) sr1@sr1:~$
[Test] 1: bash*
sr1 01:02 16-Apr-21
```

3. 터미널 접속관리

TUMX (Terminal Multiplexer)

Option	Description
tmux new -s 세션명	새로운 세션을 생성
tmux ls	tmux 세션 리스트 확인
tmux a -t 세션명	특정 세션으로 들어가기 서로 다른 사용자가 같은 session 연결시, 화면 공유
tmux kill-session -t 세션명	세션 종료

내부명령어	Description
Ctrl+b, d	현재 세션에서 빠져나오기
Ctrl+b, %	현재 화면 세로 분할
Ctrl+b, "	현재 화면 가로 분할
Ctrl+b, 방향키	방향키가 가리키는 화면으로 1회 이동
Ctrl+b+방향키	현재 화면 크기 조절
Ctrl+b, ?	단축키 목록 보기

▣ TMUX 실행

```
$ tmux new -s term_name
```

```
$ [ctrl 키] + b 후 %      ## 현재 화면(Pane)을 세로분할  
$ [ctrl 키] + b 후 "      ## 현재 화면(Pane)을 가로분할  
- 다른 화면(Pane) 이동: ctrl + b , 방향키
```

▣ 현재 화면 종료

```
$ exit
```

▣ TMUX 세션 빠져나오기

```
$ [ctrl 키] + b 후 d
```

▣ TMUX 세션 조회

```
$ tmux ls
```

▣ TMUX 세션으로 다시 들어가기

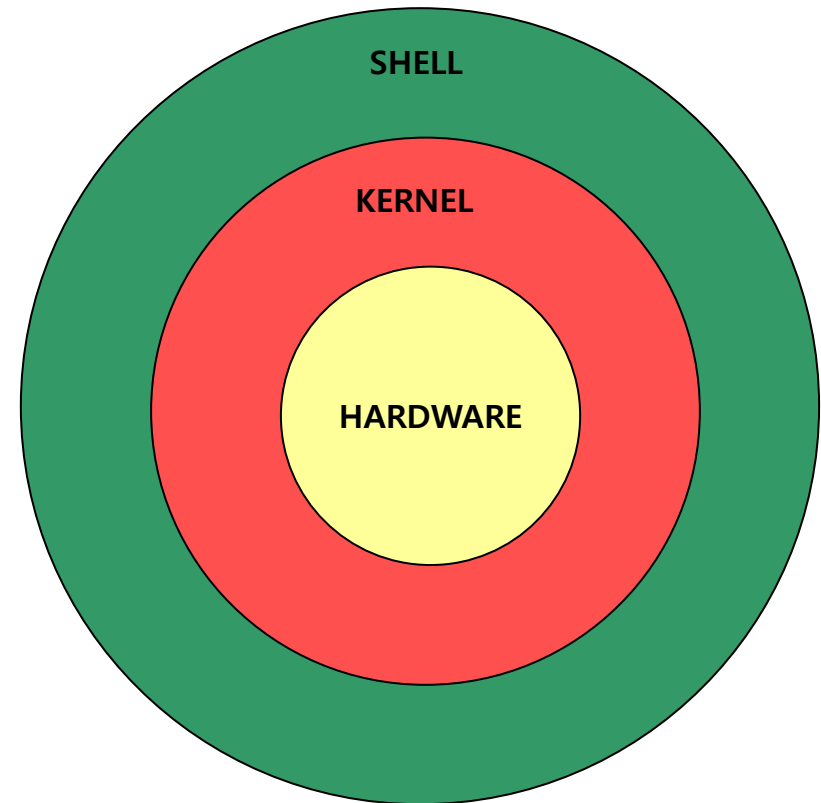
```
$ tmux a -t term_name
```

4. 웹 설정 및 사용법

4. 셸 설정 및 사용법

Shell

- Linux의 주요 Shell
 - sh (Borne Shell)
 - bash (Bourne-Again Shell)
 - ksh (Korn Shell)
 - csh (C Shell)
- 현재 사용 중인 셸 확인
 - echo \$SHELL
- 시스템에서 제공하는 shell 종류
 - chsh -l
 - cat /etc/shells
- Shell Script Programming
 - UNIX/Linux 시스템의 기본 명령어를 반복적으로 사용할 때 유용
 - shell script는 shell에서 제공하는 기능을 순차적으로 사용하기 위해서 프로그램처럼 동작하도록 작성
 - 필요에 따라 shell용 연산자와 옵션들을 사용할 수 있음



4. 셸 설정 및 사용법

Shell 환경설정 (bash)

사용자 접속 후 셸 프롬프트를 구성하는 환경설정 파일을 읽어들이는 과정



/etc/profile

- 1)시스템 공통으로 적용되는 환경 변수 설정
- 2)기본 접근 권한 설정
- 3) /etc/profile.d/*.sh를 실행

/etc/bashrc

- 1)시스템 공통으로 적용되는 함수와 alias 등 설정
- 2)기본 프롬프트 설정
- 3)서브 셸을 위한 명령 경로 설정
- 4)서브 셸을 위한 기본 접근 권한 설정

\$HOME/.bash_profile

- 1) 사용자의 SHELL 환경변수 설정
- 2) \$HOME/.bashrc 를 읽어 옴

\$HOME/.bashrc

- 1) 사용자에게 환경에 적용되는 함수와 alias 등 설정
- 2) 다른 사용자에게 영향을 미치지 않음

4. 쉘 설정 및 사용법

Shell 환경설정 (bash)

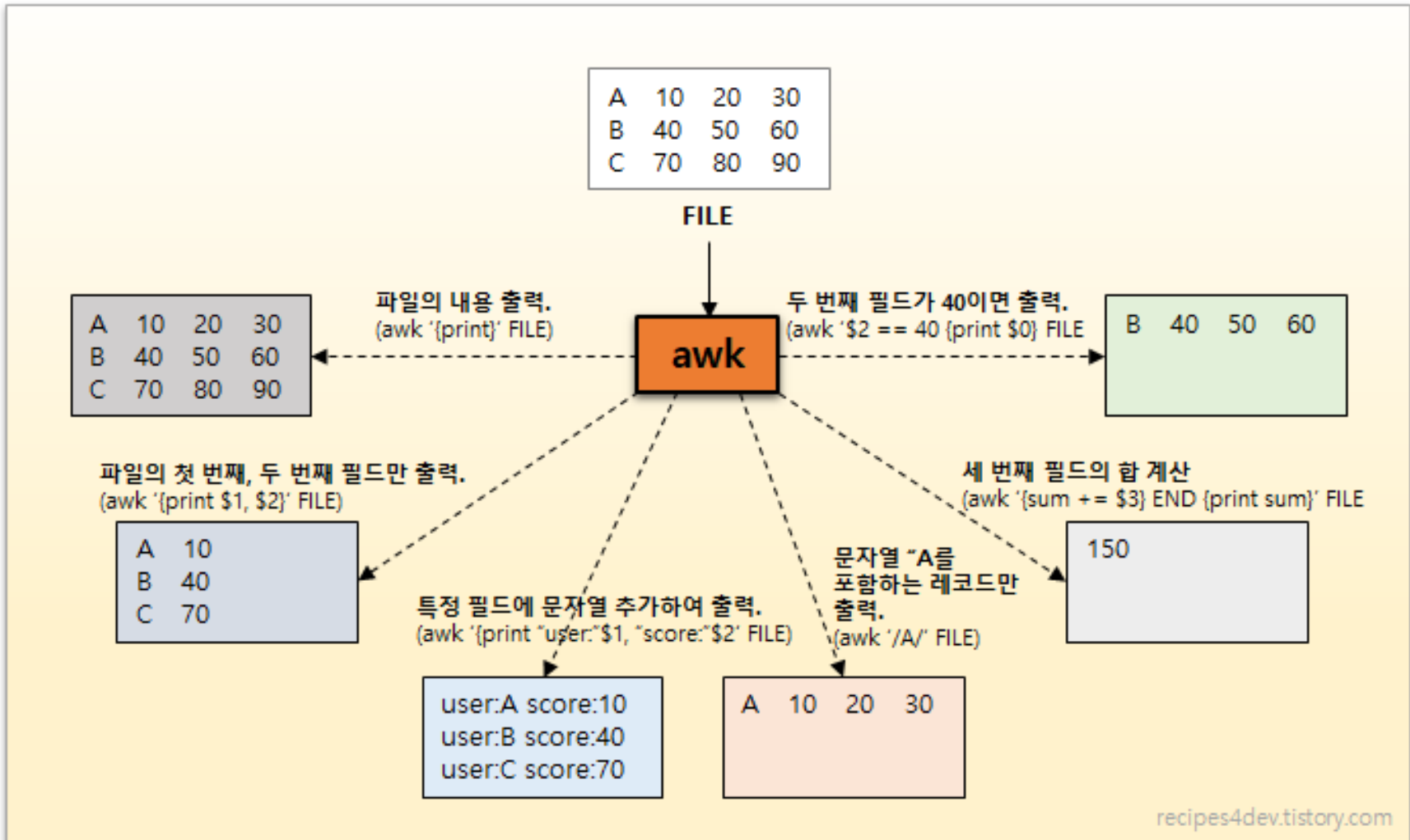
현재 프롬프트 상태에서 임시로 쉘 환경설정을 위한 명령들

Command	Description
env	설정된 환경변수 확인 \$ env
source	환경변수 파일을 다시 설정 \$source ~/.bashrc \$source ~/.bash_profile
export	환경변수 설정 \$ export 변수명=값 \$ export PATH=~/.bin:\$PATH
unset	설정된 환경변수 삭제 \$ unset 변수명
echo	설정된 환경변수 중 특정 변수값 확인 \$ echo \$PATH \$ echo \$SHELL
alias	길이가 긴 명령어를 간단한 문자로 변경하여 등록 \$ alias go-netconf='cd /etc/sysconfig/network-scripts/'

4. 쉘 설정 및 사용법

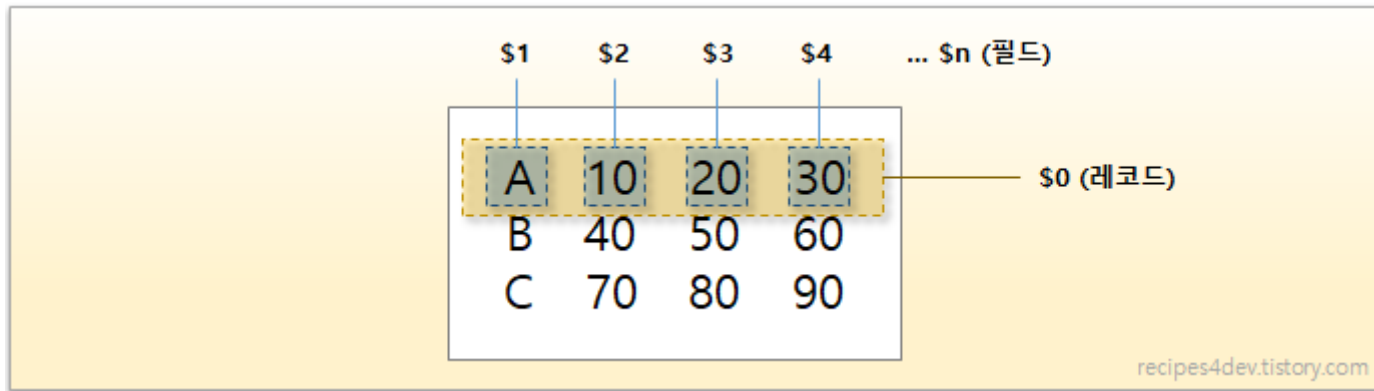
awk

파일이나 데이터로부터 레코드(record)를 선택하고, 선택된 레코드에 포함된 값을 조작하거나 데이터화 하는 것을 목적으로 사용하는 명령어



4. 쉘 설정 및 사용법

awk



- 테스트 파일 생성

```
$ vi file.txt
```

```
A 10 20 30
```

```
B 40 50 60
```

```
C 70 80 90
```

- 레코드의 길이가 10 이상인 경우, 첫 번째(\$1), 세 번째(\$3), 네 번째(\$4) 필드 출력 예

```
$ awk 'length($0) >=10 {print $1, $3, $4}' ./file.txt
```

```
A 20 30
```

```
B 50 60
```

```
C 80 90
```

4. 쉘 설정 및 사용법

awk

- 테스트 파일 생성

```
$ vi file2.txt
```

```
1 ppotta 30 40 50
```

```
2 soft    60 70 80
```

```
3 prog    90 10 20
```

- 필드 값에 임의의 문자열을 같이 출력 (예)

```
$ awk '{print "no:"$1, "user:"$2}' ./file2.txt
```

```
no:1 user:ppotta
```

```
no:2 user:soft
```

```
no:3 user:prog
```

- 레코드의 길이가 10 이상인 경우, 첫 번째(\$1), 세 번째(\$3), 네 번째(\$4) 필드 출력 (예)

```
$ awk 'BEGIN { print "TITLE : Field value 1,2" } {print $1, $2} END {print "Finished"}' file.txt
```

```
TITLE : Field value 1,2
```

```
A 10
```

```
B 40
```

```
C 70
```

```
Finished
```

4. 쉘 설정 및 사용법

awk

```
$ cat ./file2.txt
```

```
-----  
1 ppotta 30 40 50  
2 soft    60 70 80  
3 prog    90 10 20  
-----
```

- 파일에서 " pp " 문자가 포함된 레코드만 출력

```
$ awk ' /pp/ ' ./file2.txt  
1 ppotta 30 40 50
```

- 파일에서 20, 30 이 포함된 레코드만 출력

```
$ awk '/[2-3]0/' ./file2.txt  
1 ppotta 30 40 50  
3 prog    90 10 20
```

4. 쉘 설정 및 사용법

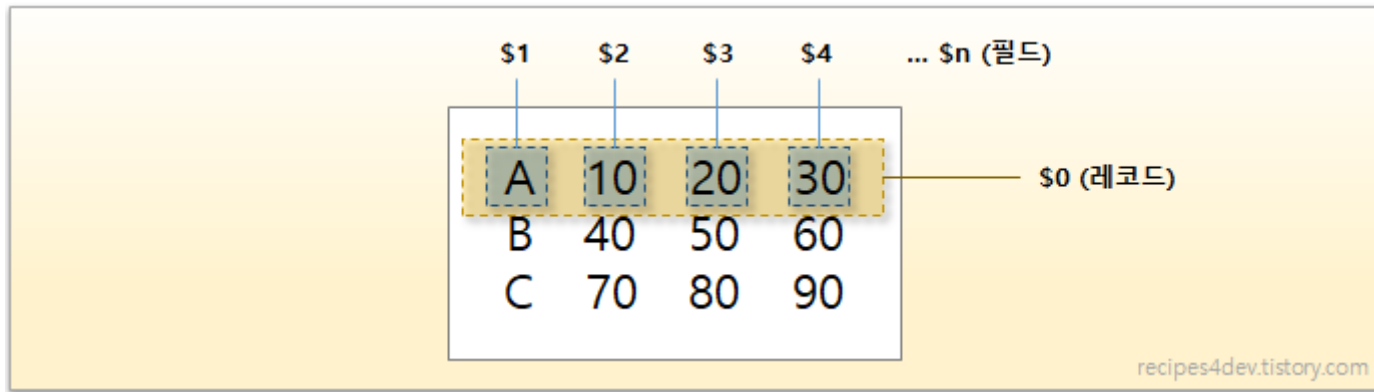
awk

```
$ ls -l
total 3
drwxr-xr-x  2 dawon MIT 4096 Nov  7  15:43 backup
-rw-r--r--  1 dawon MIT  54 Nov  7  17:12 file2.txt
-rw-r--r--  1 dawon MIT  36 Nov  7  17:04 file.txt
-rw-r--r--  1 dawon MIT 276 Nov  7  15:43 test
-rw-r--r--  1 dawon MIT 208 Nov  7  18:02 test.sh
```

```
$ ls -l | awk '{ print $1, $3, $9 }'
total
drwxr-xr-x dawon backup
-rw-r--r-- dawon file2.txt
-rw-r--r-- dawon file.txt
-rw-r--r-- dawon test
-rw-r--r-- dawon test.sh
```

4. 쉘 설정 및 사용법

실습



- 테스트 파일 생성

```
$ vi file.txt
```

```
A 10 20 30
```

```
B 40 50 60
```

```
C 70 80 90
```

- 레코드의 길이가 10 이상인 경우, 첫 번째(\$1), 세 번째(\$3), 네 번째(\$4) 필드 출력

```
$ awk 'length($0) >=10 { print $1, $3, $4} ' ./file.txt
```

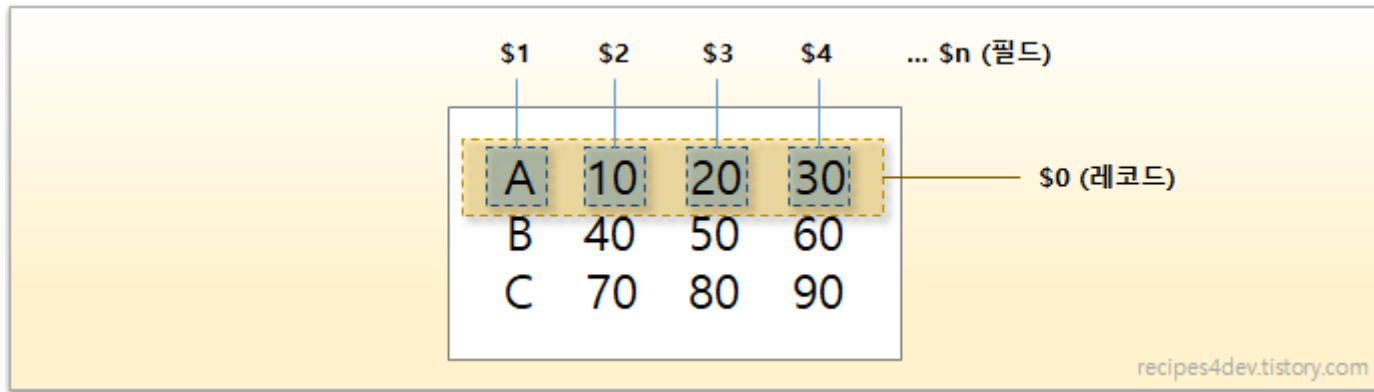
```
A 20 30
```

```
B 50 60
```

```
C 80 90
```

4. 쉘 설정 및 사용법

실습



- 파일에서 "A"가 포함된 레코드만 출력
`$ awk '/A/' ./file.txt`
- ls 명령 실행결과 중 1, 3, 9번 필드만 출력
`$ ls -l`
`$ ls -l | awk '{ print $1, $3, $9 }'`

4. 쉘 설정 및 사용법

awk

awk 사용 예시 (참고용)

awk 사용 예	명령어 옵션
파일의 전체 내용 출력	<code>awk '{ print }' [FILE]</code>
필드 값 출력	<code>awk '{ print \$1 }' [FILE]</code>
필드 값에 임의 문자열을 같이 출력	<code>awk '{print "STR"\$1, "STR"\$2}' [FILE]</code>
지정된 문자열을 포함하 는 레코드만 출력	<code>awk '/STR/' [FILE]</code>
특정 필드 값 비교를 통해 선택된 레코드만 출력	<code>awk '\$1 == 10 { print \$2 }' [FILE]</code>
특정 필드들의 합 구하기	<code>awk '{sum += \$3} END { print sum }' [FILE]</code>
여러 필드들의 합 구하기	<code>awk '{ for (i=2; i<=NF; i++) total += \$i }; END { print "TOTAL : "total }' [FILE]</code>
레코드 단위로 필드 합 및 평균 값 구하기	<code>awk '{ sum = 0 } {sum += (\$3+\$4+\$5) } { print \$0, sum, sum/3 }' [FILE]</code>

4. 쉘 설정 및 사용법

awk

awk 사용 예시 (참고용)

awk 사용 예	명령어 옵션
필드에 연산을 수행한 결과 출력하기	awk '{print \$1, \$2, \$3+2, \$4, \$5}' [FILE]
레코드 또는 필드의 문자열 길이 검사	awk 'length(\$0) > 20' [FILE]
파일에 저장된 awk program 실행	awk -f [AWK FILE] [FILE]
필드 구분 문자 변경하기	awk -F ':' '{ print \$1 }' [FILE]
awk 실행 결과 레코드 정렬하기	awk '{ print \$0 }' [FILE] sort -r
특정 레코드만 출력하기	awk 'NR == 2 { print \$0; exit }' [FILE]
출력 필드 너비 지정하기	awk '{ printf "%-3s %-8s %-4s %-4s %-4s\n", \$1, \$2, \$3, \$4, \$5}' [FILE]
필드 중 최대 값 출력	awk '{max = 0; for (i=3; i<NF; i++) max = (\$i > max) ? \$i : max ; print max}' [FILE]

4. 쉘 설정 및 사용법

awk

awk 정의 변수 (참고용)

정의 변수	설명
ARGV	command line argument 배열
ARGC	ARGV 배열 요소의 갯수
CONVFMT	포매팅 형식 (example: 숫자 포매팅 형식)
ENVIRON	환경변수 배열
FILENAME	파일 이름 (경로포함)
FS	필드 구분자 (default: space)
NF	필드의 갯수
NR	현재 레코드의 순서
OFMT	문자열 출력 형식
OFS	필드 구분자 (default: space)
ORS	레코드 구분자 (default: newline)
RLENGTH	match 함수에 의해 매칭된 문자열의 길이
RS	레코드 구분 문자 (default: newline)

4. 쉘 설정 및 사용법

Shell 스크립트

명령어들을 자동으로 실행하기 위해 작성하는 실행파일 입니다.
한꺼번에 여러 명령어를 실행하거나 반복적인 작업을 자동화할 때 유용합니다.

- `#!/bin/sh` 또는 `#!/bin/bash` 로 시작, 명령어 해석기를 지정
- `#` 으로 시작하면 그 라인은 주석, 즉 실행되지 않음
- 명령어 해석기가 스크립트의 첫번째 줄부터 명령어들을 실행

```
#!/bin/bash
# This is sample script
cd /var/tmp
rm -f ./*
echo "파일을 정리했습니다."
```

- 명령어 해석기 지정

```
#!/bin/sh
#!/bin/bash
#!/bin/csh
#!/bin/ksh
#!/bin/awk
#!/usr/bin/perl
```

4. 쉘 설정 및 사용법

Shell script를 사용하면 안되는 경우

- 속도가 중요한 작업인 경우
 - 강력한 산술 연산 작업, 정밀도 연산 또는 복소수 써야 할 때
 - 플랫폼간 이식성이 필요할 때
 - 구조적 프로그래밍이 필요한 복잡한 어플리케이션
 - 중요한 어플리케이션
 - 보안상 중요한 어플리케이션
 - 과도한 파일 연산이 필요할 때
 - 다차원 배열이 필요할 때
 - 링크드 리스트나 트리 같은 데이터 구조가 필요할 때
 - 그래픽, GUI 등을 만들 때
 - 시스템 하드웨어에 직접 접근해야 할 때
 - 포트, 소켓 I/O
 - 예전에 쓰던 코드를 사용하는 라이브러리나 인터페이스를 써야 할 필요가 있을 때
 - 독점적으로 소스 공개를 하지 않을 어플리케이션 작성시 (셸 스크립트는 필연적 오픈소스)
- 위 경우에 하나라도 해당된다면 Perl, Python, TCL, C, C++, Fortran, Java 등을 사용할 것

4. 셸 설정 및 사용법

Shell 스크립트 (실행)

- vi test.sh

#!/bin/bash

#명령어 해석기 지정(bash)

echo my_home=\$HOME

echo workdir=\$PWD

:wq!

- chmod 555 scriptname - 아무나 읽기/실행 가능
- chmod +rx scriptname - 아무나 읽기/실행 가능
- chmod u+rx scriptname - 소유자만 읽고/실행 가능
- ./test.sh - 셸 스크립트 실행
- sh -x ./test.sh - 각 명령어 실행 결과를 출력, 디버깅용

4. 쉘 설정 및 사용법

Shell 스크립트 (특수문자)

- #
 - 주석
 - #! 은 제외
 - # echo "주석"
 - echo " # 주석"
 - echo "# 주석 " # 주석은 여기에
- ; (세미콜론)
 - 명령어 구분
 - 보통 라인으로 명령어를 구분
- . (점)
 - source 명령어와 동일
 - 정규표현식 (Regular Expression)에서는 한 개의 문자

4. 셸 설정 및 사용법

Shell 스크립트 (특수문자)

- " (partial quoting)
 - 셸이 "문자열"에서 **거의 대부분의 특수문자** 부분을 미리 해석하지 못하게 하여 일반문자로 해석 (일부 특수문자 제외)
- ' (full quoting)
 - 셸이 '문자열'에서 **모든 특수문자** 부분을 미리 해석하지 못하게 하여 일반문자로 해석
- \ (back slash, escape)
 - 특수문자를 일반문자로 해석되도록 할 때 사용
 - \X : X 문자를 이스케이프 시키고, 'X' 라고 쿼우팅 하는 것과 동일한 효과,
- / (slash)
 - 파일명의 각 요소들을 구분 (/home/idh/bin/color.sh)
 - 나누기 산술연산
- \$
 - 변수 치환
 - \${} : 매개변수 치환
 - \$*, @\$: 위치 매개변수

4. 쉘 설정 및 사용법

Shell 스크립트 (변수)

- 변수=상수
= 앞뒤로 공백이 있으면 안됨
상수로 \$(...) 사용 가능

```
#!/bin/bash

a=23          # 평범한 경우
echo $a
b=$a
echo $b

a=`echo Hello!` # 'echo' 명령어의 결과를 'a' 로 할당
echo $a

a=`pwd`        # 'pwd' 명령어의 결과를 'a' 로 할당
echo $a

date=$(date)   # 'date' 명령어 결과를 'date'로 할당
echo $date

exit 0
```

4. 쉘 설정 및 사용법

실 습

```
$ vi test.sh
#!/bin/bash

a=23; echo a=$a
b=$a && echo b=$b

a=`echo Hello!`
echo $a

a="$pwd"
echo mypath=$a > echo.txt
cat echo.txt

date=$(date)
echo $date

exit 0
```

```
$ chmod 700 ./test.sh
```

```
$ ./test.sh
$ bash -x ./test.sh
```

