**[이 파일의 내용은 코드스테이츠 DOB 교육 과정 중에 활용 가능한 자료(유어클래스 내용이 일부 포함됨)로 본인이 학습하는 용도로만 사용 가능합니다. 외부 유출은 허용되지 않습니다.]**

아래 내용 외에 각자 필요하다고 생각하는 용어 및 내용을 추가해서 정리한 후, 자주 반복해서 읽으면 실력향상에 많은 도움이 될 수 있습니다. 각자 편리한 방법(다른 문서 편집 도구들)을 찾아서 정리해보기 바랍니다.

\* GNU를 사용하는 방법

GNU는 저작권이 인정되는(not public domain) 소프트웨어다. 누구나 GNU를 개작하고 배포할 수 있지만 어떤 이도 GNU가 보다 널리 배포되는 것을 제한할 수 없다. 즉, 변경한 내용을 독점할 수 없다는 것이다. 나는 모든 버전의 GNU가 공개된 채로 남아 있기를 보장받고 싶은 것이다.

\* DevOps: 개발 + 운영

DevOps는 새로운 소프트웨어 기능, 개선 요청 또는 버그 수정 등 하나의 아이디어가 개발에서 배포에 이르는 프로세스의 속도를 높임으로써 더 빨리 프로덕션 환경에 전달되어 사용자에게 가치를 전달하도록 합니다. 이러한 접근 방식을 적용하려면 개발 팀과 운영 팀이 자주 커뮤니케이션하고 팀원들과 공감하면서 업무에 접근해야 합니다. 확장성과 유연한 프로비저닝도 필요합니다. DevOps를 확립하면 셀프 서비스와 자동화를 통해 다양한 이점과 경쟁력을 얻을 수 있습니다. 대개 표준 개발 환경에서 코딩 작업을 하는 개발자는 IT 운영 담당자와 긴밀하게 협력하여 소프트웨어 빌드, 테스트, 출시 속도를 가속화할 수 있습니다.

<https://www.redhat.com/ko/topics/devops>

\* 프로비저닝(provisioning)

프로비저닝은 사용자의 요구에 맞게 시스템 자원을 할당, 배치, 배포해 두었다가 필요 시 시스템을 즉시 사용할 수 있는 상태로 미리 준비해 두는 것을 말한다. 서버 자원 프로비저닝, OS 프로비저닝, 소프트웨어 프로비저닝, 스토리지 프로비저닝, 계정 프로비저닝 등이 있다.

<https://www.redhat.com/ko/topics/automation/what-is-provisioning>

\* Kubernetics

- 컨테이너: 앱이 구동되는 환경까지 감싸서 실행할 수 있도록 하는 격리 기술

- 컨테이너 런타임: 컨테이너를 다루는 도구, 가장 유명한 것은 Docker

- 쿠버네티스: 컨테이너 런타임을 통해 컨테이너를 오케스트레이션 하는 도구

- 오케스트레이션: 여러 서버에 걸친 컨테이너 및 사용하는 환경 설정을 관리하는 행위

\* <https://www.samsungsds.com/kr/insights/220222_kubernetes1.html>

CLI: Command line interface

SSH: secure shell

\* 리눅스 디렉토리 구조

/home: 사용자 디렉토리

/usr: 사용자 유틸리티 & 어플리케이션

/root:super user의 home 디렉토리

/dev: 장치관련 파일

/bin: single-user 모드에서 사용가능한 필수 command binaries, cat, ls, cp, terminal commands 등

/etc: system 전역에 걸친 configuration 파일들(즉, 모든 사용자들에 대한 system의 행위에 영향을 끼치는)

/sbin: superuser만 사용하는 운영 관련 중요 commands

/var: variable files, 시스템의 정상적인 운영도중 지속적으로 변견되는 내용 파일들, logs, spool files, temporary e-maile files

/dev/tty:

(원격) 사용자와 시스템 사이를 중계하는 역할을 합니다. 다리 역할을 한다고 할까요? tty를 통해서 사용자는 시스템과 통신을 하게 됩니다. 사용자가 입력하는 키보드의 입력은 tty를 통해서 시스템에 전달되고 시스템에서 내보내는 출력 역시 tty을 통과해서 사용자에게 전달되죠. 예전에는 UNIX 시스템에 터미널이라는 장치를 통해 접근했고 그 터미널의 종류가 모두 달랐기 때문에 그 터미널을 추상화하기 위해서도 사용됩니다.

ps. 중간에 터미널이 있기 때문에 console 접속이 아니면 C 표준 라이브러리 함수인 getc() getchar()등이 예상한대로 동작하지 않게되죠. ;)

https://jinwoo1990.github.io/dev-wiki/linux-directory/

\* File descriptor:

파일 디스크립터(File Descriptor)란 리눅스 혹은 유닉스 계열의 시스템에서 프로세스(process)가 파일(file)을 다룰 때 사용하는 개념으로, 프로세스에서 특정 파일에 접근할 때 사용하는 추상적인 값이다. 파일 디스크럽터는 일반적으로 0이 아닌 정수값을 갖는다.

흔히 유닉스 시스템에서 모든 것을 파일이라고 한다. 일반적인 정규파일부터 디렉토리, 소켓, 파이프, 블록 디바이스, 케릭터 디바이스 등 모든 객체들을 파일로 관리한다. 유닉스 시스템에서 프로세스가 이 파일들을 접근할 때 파일 디스크립터라는 개념일 이용한다. 프로세스가 실행 중에 파일을 Open하면 커널은 해당 프로세스의 파일 디스크립터 숫자 중 사용하지 않는 가장 작은 값을 할당해준다. 그 다음 프로세스가 열려있는 파일에 시스템 콜을 이용해서 접근할 때, 파일 디스크립터(FD)값을 이용해서 파일을 지칭할 수 있다.

프로그램이 프로세스로 메모리에서 실행될 때, 기본적으로 할당되는 파일디스크립터는 표준입력(Standard Input), 표준 출력(Standard Output), 표준에러(Standard Error)이며 이들에게 각각 0, 1, 2라는 정수가 할당된다.

<https://twofootdog.tistory.com/51>

\* 데이터 분석을 위해 사용할 수 있는 도구

awk

사람 이름: Aho + Weinberger(바인베거) + Kernighan(케나인). (A:Alfred V. Aho, W:Peter J. Weinberger, K:Brian W. Kernighan)

텍스트 파일의 전체 내용 출력.

파일의 특정 필드만 출력.

특정 필드에 문자열을 추가해서 출력.

패턴이 포함된 레코드 출력.

특정 필드에 연산 수행 결과 출력.

필드 값 비교에 따라 레코드 출력.

<https://recipes4dev.tistory.com/171>

sed(stream editor): grep과 같은 필터, 라인들을 하나씩 읽고 수정하고, 출력하기 때문에 기억장치 안의 버퍼를 사용하지 않음. 버퍼를 사용하지 않으면 파일의 크기에 제한없이 작업 가능. 따라서, sed는 큰 파일 처리 가능, <https://spadework-blog.tistory.com/94>

grep

특정 파일에서 지정한 문자열이나 정규 표현식을 포함한 행을 출력해주는 명령어입니다. 특히 tail이나 ls 등 다양한 명령어와 조합하여 응용되는 경우가 많아서 이 grep 명령어는 리눅스에서 능숙하게 사용할 줄 알아야 하는 기본 명령어입니다.

tail -f a.log | grep 192.168.0.55

<https://coding-factory.tistory.com/802>

sort

<https://jaynamm.tistory.com/entry/%EB%A6%AC%EB%88%85%EC%8A%A4-sort-%EB%AA%85%EB%A0%B9%EC%96%B4>

uniq

<https://www.lesstif.com/lpt/linux-uniq-95879394.html>

tail –f

<https://blog.naver.com/comlo1/221241134005>

\* 파일복사

cp

-i : 복사될 파일이 이름이 이미 존재할 경우, 사용자에게 덮어 쓰기 여부를 묻습니다.

-b : 복사될 파일이 이름이 이미 존재할 경우, 백업파일을 생성합니다.

-f : 복사 될 파일이 이름이 이미 존재 할 경우, 강제로 덮어쓰기 합니다.

-r : 하위 디렉토리 까지 모두 복사합니다.

-a : 원본 파일의 속성, 링크 정보까지 모두 복사합니다.

-p : 원본 파일의 소유자, 그룹, 권한 등의 정보까지 모두 복사합니다.

-v : 복사 진행 상태를 출력합니다.

현재 위치에 있는 org\_folder 디렉토리를 new\_folder 디렉토리라는 이름으로 복사

ex) cp -r org\_folder new\_folder

cp [복사 대상1] [복사 대상2] [복사 대상3] ..... [복사 될 경로]

ex) cp test1.txt test2.txt test3.txt /user/jtaewu

<https://coding-factory.tistory.com/750>

\* 스케줄러

cron

The cron command-line utility is a job scheduler on Unix-like operating system.

<https://zamezzz.tistory.com/197>

\* 리눅스 배포본 비교

Linux 배포본은 종류도 많고 겉보기엔 상당히 다른 것처럼 보이지만, 자세히 살펴보면 그렇게 크게 다르지 않습니다. 패키지 관리 프로그램이 무엇이냐, 그래피컬 인터페이스가 어떤 것이냐, Red Hat 계열이냐/Debian 계열이냐, 배포본 솔루션업체가 추구하는 성향이 어떠냐 등에 따라 조금씩 다르지만, 결국 같은 커널을 사용하고 같은 핵심 라이브러리를 사용하는 보통의 Linux일 뿐입니다.

<https://www.samsungsds.com/kr/insights/linux_distribution.html>

\* RISC (Reduced Instruction Set Computer)는 CPU명령어의 개수를 줄여 하드웨어 구조를 좀 더 간단하게 만드는 방식

\* 터미널

tty: 표준 출력에 해당하는 터미널의 장치 파일 이름을 출력해주는 명령어. 보통 현재 사용 중인 터미널의 파일명을 확인할 때 사용. 내가 현재 몇 번째 가상 콘솔에 있는지 확인하는 명령어

/dev/tty

현재 콘솔 장치

현재 콘솔 세션

로그인한 사용자의 터미널 장치

fopen()으로 열어 터미널 장치로부터 데이터를 읽거나 쓸 수 있다.

[root@zetawiki ~]# tty

/dev/pts/2

[root@zetawiki ~]# echo hello > /dev/tty

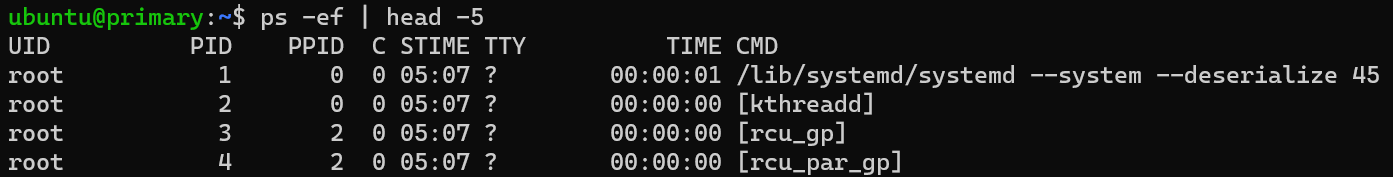
hello

\* 파일 위치

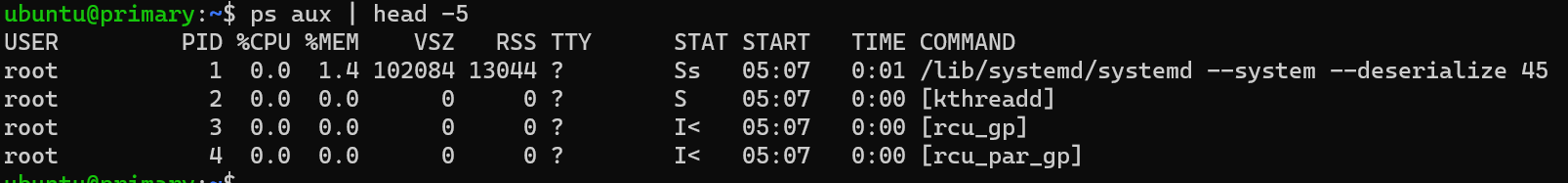
which ls

\* 프로세스 목록

ps –ef | head -5



ps aux | head -5



ps 항목 값

<https://jhnyang.tistory.com/268>

\* 파일 및 디렉토리 찾기

sudo find / -name "grep“

sudo find / -size +100M

find . -empty

find . -newer .bashrc

sudo apt-get install mlocate

sudo updatedb

locate -n 10 sys\*

\* 패턴으로 찾기 (grep)

$ grep "문자열" [filename]

$ grep "Gatsby" gatsby.txt # gatsby.txt 파일에서 Gatsby 문자열 찾기

대소문자 구분 않함

$ grep -i "문자열" [filename]

$ grep -i "he" gatsby.txt # gatsby.txt 파일에서 he 문자 대소문자 구분없이 찾기

매칭되는 패턴이 존재하지 않는 라인 선택

$ grep -v "문자열" [filename]

$ grep -v "he" gatsby.txt # gatsby.txt 파일에서 he가 없는 라인 나타내기

다음의 명령을 실행할 수 있는 grep 명령어의 옵션 찾기

단어 단위로 문자열 검색

검색된 문자열이 포함된 라인 번호 출력

하위 디렉토리를 포함한 모든 파일에서 문자열 검색

최대 검색 결과 갯수 제한

문자열 A로 시작해서 문자열 B로 끝나는 패턴 찾기

<https://rootkey.tistory.com/104>

\* su

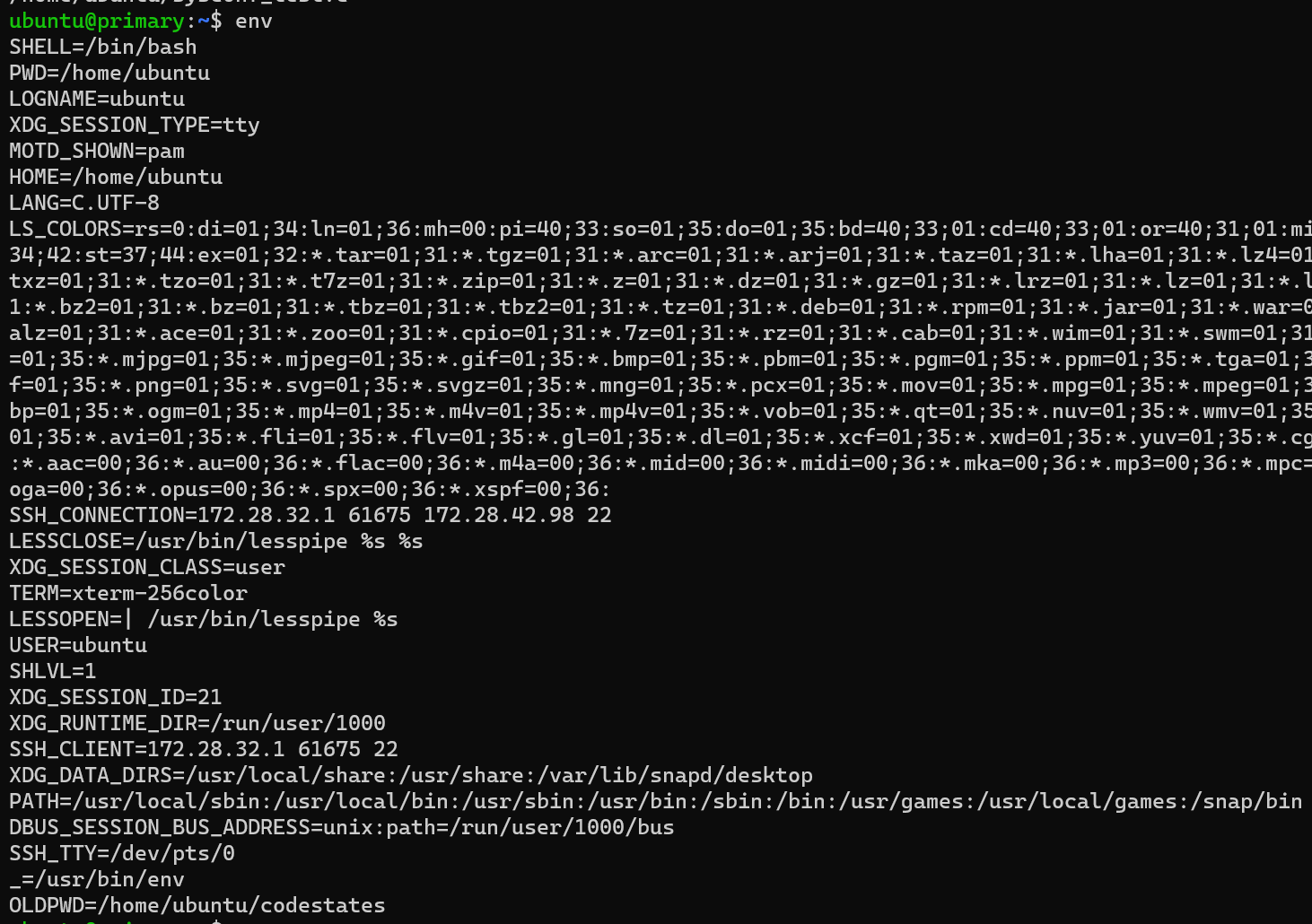
su는 substitute user identity의 줄임말로, "사용자의 대리자"라는 의미

super user는 초기에 사용

sudo – substitute user do, super user do

\* 환경 변수 목록

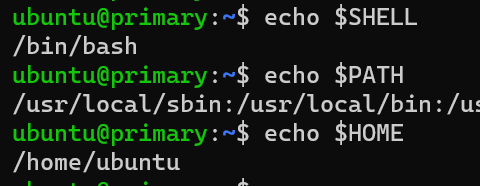
env



SHELL: 어떤 종류의 쉘(대화형 인터페이스)을 사용할지 선택하는 환경 변수

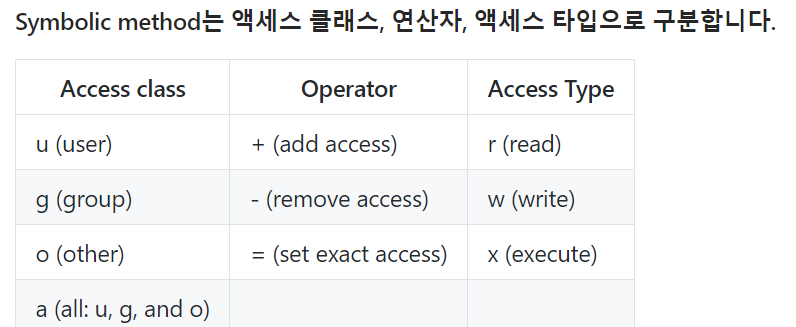
HOME: 홈 디렉토리가 무엇인지 알려주는 환경 변수

PATH: 어디서든 해당 디렉토리에 있는 실행 파일을 실행할 수 있게 만들어주는 환경 변수



\* 파일 권한 부여 - Read, Write, Execute 권한

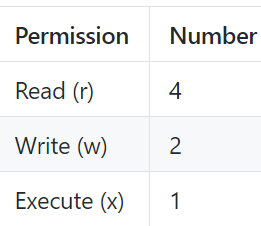
방법1





방법2

Absolute form: 3bits의 숫자. 0~7





\* 프로세스와 쓰레드

gcc

<https://m.blog.naver.com/kore2758_/221259202488>

gcc multithread.c -o multithread –pthread

./multithread

gcc thread.c -o thread –pthread

./thread

\* 프로세스 관리 – ps, pstree

# 포그라운드 방식

$ command

$ ping google.com

# 백그라운드 방식 -- &추가

$ command &

$ ping google.com &

-> 중지하려면 jobs 입력 후 []안의 번호 확인 -> fg 번호 [엔터] -> ctrl+c

# nohup 커맨드(no hang up)

$ nohup command &

$ nohup ping google.com &

UNIX 방식

ps -e : 시스템에서 실행 중인 모든 프로세스의 정보를 출력한다.

ps -f : 프로세스의 자세한 정보를 출력합니다.

ps -u uid : uid로 특정한 사용자에 대한 모든 프로세스의 정보를 출력한다.

ps -p pid : pid로 지정한 특정 프로세스의 정보를 출력한다.

BSD 방식

ps a : 터미널에서 실행한 프로세스의 정보를 출력한다.

ps u : 프로세스 소유자의 이름, CPU 사용량, 메모리 사용량 등 상세 정보를 출력한다.

ps x : 시스템에서 실행 중인 모든 프로세스의 정보를 출력한다.

이러한 옵션은 방식 별로 섞어서 사용할 수 있으며, 보통 프로세스의 관계를 보기 위해서 UNIX 방식인 ps -ef를, 프로세스의 상태를 보는 용도로는 BSD 방식인 ps aux를 가장 많이 사용합니다.

위의 명령에 따른 결과로 다양한 항목이 출력됩니다. 위에서 살펴본 항목 외에도 무엇이 있는지 아래서 살펴보겠습니다.

ps -f의 출력 정보

UID : 프로세스를 실행한 사용자 ID

PPID : 부모 프로세스 번호

C : CPU 사용량(%)

STIME : 프로세스의 시작 날짜나 시간

TTY : 프로세스가 실행된 터미널의 종류와 번호

STAT : 프로세스 상태

ps a, u, x의 출력 정보

USER : 사용자 계정 이름

%CPU : CPU 사용량(%)

%MEM : 물리적 메모리 사용량(%)

RSS : 사용하고 있는 물리적 메모리의 크기(kb)

VSZ : 사용하고 있는 가상 메모리의 크기(kb)

START : 프로세스 시작 시간

\* 프로세스 관리 – kill

kill –l

SIGHUP : 재시작할 때 사용

SIGINT : 실행 중지 시그널, Ctrl + c

SIGKILL : 프로세스 강제 종료

SIGTERM : 프로세스 정상종료 (기본 명령)

SIGCONT : 정지된 프로세스 실행

SIGSTOP : 터미널에서 입력되는 정지 시그널

SIGTSTP : 실행 정지 후 재실행 대기, Ctrl+ z

kill -시그널번호(시그널명) %작업번호

kill –시그널번호(시그널명) PID

위에서 언급한 대로 아무런 시그널을 지정하지 않고 kill PID 이라고만 하면 SIGTERM (9)이 전달됨.

kill `ps -ef | grep 프로세스이름 | grep -v grep | awk '{print $2}'`

파이프와 grep, awk, 역따옴표(backticks : 백틱)을 조합하면 특정 이름의 프로세스를 모두 찾아서 종료시킬 수 있습니다.

파이프와 grep은 리눅스에서 가장 많이 사용되는 명령어 조합 중에 하나로 이전 스트림의 출력값에 원하는 형식으로 필터링을 할 수 있습니다.

위 스크립트에서 프로세스이름 부분에 와일드카드 문자나 정규표현식도 사용 가능합니다.

grep -v 은 지정된 패턴과 일치하는 항목을 제외할 때 사용합니다.

여기서는 "grep 프로세스이름" 도 하나의 명령어기 때문에 "ps -ef | grep 프로세스이름" 의 결과에 포함되는데 kill 명령어에서 이를 제외시키기 위함입니다.

awk 명령어는 입력 값(record : 레코드)을 공백 문자로 분리하여 필드(field) 단위로 처리 할 수 있게 해주는 명령어입니다.

여기서는 ps -ef 명령어의 결과에서 PID를 추출하기 위해 사용됩니다.

마지막으로 `명령어` 백틱(backticks)은 치환 명령어라고도 하는데 ` ` 로 감싼 부분의 실행 결과로 치환됩니다.

결과적으로 kill `ps -ef | grep 프로세스이름 | grep -v grep | awk '{print $2}'` 에서 `ps -ef | grep 프로세스이름 | grep -v grep | awk '{print $2}'` 부분이 실행된 결과로 치환되어 kill PID 형태로 실행이 되는 것입니다.

\* 프로세스 관리

jobs

+ 기호로 표시된 프로세스는 현재 처리되고 있는 작업을 의미하며, - 기호는 다음에 작업할 프로세스를 의미합니

프로세스의 상태

Running : 실행 중

Stopped : 일시 중단(Ctrl + Z 입력)

Terminated : 강제 종료(kill 명령 종료)

Done : 정상 종료

\* 데몬과 서비스

데몬은, 리눅스 환경에서 시스템이 초기화가 끝난 후, init 이라는 특별한 프로그램에 의해 자발적으로 백그라운드에서 실행하는 프로그램을 의미합니다. 그러나 일반적으로 “백그라운드에서 작동하는 프로세스”의 의미로 통용됩니다. 데몬은 보통 syslogd, httpd 처럼 프로그램 이름 끝에 d가 붙습니다.이러한 데몬은 상시 실행되며 주로 다음과 같은 일을 합니다.

하드웨어 장치 관리

보조 기억장치와 디렉토리의 연결(마운트)

네트워크를 통한 프로세스간 통신(소켓)

애플리케이션 수준의 서비스 제공

\* 대표적인 데몬: cron

\* 서비스 관리 (systemctl)

\* bash 스크립트

....