cat(전체)/head/tail/more/less

grep 원하는 단어 파일명

grep "원하는 문장" 파일명

cat 파일명 | grep "원하는 문장"

grep "원하는 문장" 파일명 | head/tail -n 3

ls -al | grep "파일명" -> ls -al 결과중 "파일명"을 찾아라

ls -al | grep 파일명 -> ls -al 결과(출력)를 파일명한테 보내줘(입력)

붙여넣기: 드래그(클립보드같은 역할) 후 shift+insert키

자동완성: 글자+tab키

. : 현재 디렉토리

.. : 현재 디렉토리의 상위 디렉토리

find /etc -name "\*.conf" -print (절대경로)

: etc를 포함한 하위 디렉토리에서 .conf를 가진 모든 파일을 찾아 출력해라

find -name "\*.conf" -print (상대경로): 현재 디렉토리를 포함한...

find . -name "\*.conf" -print (상대경로): 현재 디렉토리를 포함한...

.. /파일명 : 현재 디렉토리의 상위 디렉토리의 하위 디렉토리의 파일명을 찾을 때 사용

find | grep "conf": 현재 디렉토리의 하위 디렉토리에서 conf를 포함한 모든 파일 찾기

find | grep conf

압축파일: .zip / .gz / .tar.bz / .tar.gz / .tgz

.gz -> gzip 툴 사용하여 압축

파일 생성

find > filelist: 왼쪽 커맨드에 나오는 출력문을 다음에 나오는 filelist라는 파일명에 저장해라

gzip 파일명: 원하는 파일명 압축하는 법 -> 파일명.gz

file 파일명.확장자: 파일이 어떤 파일인지 설명해줌

ex) file filelist.gz -> 결과: gzip으로 압축된 데이터라고 설명이 나옴

압축해제: gunzip

gunzip 파일명.확장자: 압축된 파일을 해제함

리눅스는 사실 확장자라는 개념이 없음: 어떤종류라는 것만 알려주는 역할

따라서 압축된 파일이라도 파일명을 바꾸면 .xx없이 그냥 바꾼 파일명으로 바뀜

단, 속성은 바뀌지 않음

ex) mv filelist.gz test -> filelist.gz였던 파일이 test로 파일명이 바뀜

하지만 file test 실행시 여전히 압축된 데이터라고 설명이 나옴

tar 툴: 압축하는 용도가 아니라 파일을 연결해주는 역할

.tar.gz : tar툴을 이용하여 여러 파일을 연결한 후 gzip툴을 이용하여 압축한 파일 형식

tar -cf 파일명.tar 파일명1 파일명2: 파일명1과 파일명2를 연결하여 하나의 파일명으로 만듦

tar -czf 파일명.tar.gz 파일명1 파일명2: 파일명1과 파일명2를 연결하여 하나의 파일명으로 만들고 gzip으로 압축해라

tar -tvf 파일명.tar: 연결된 파일명들의 리스트를 나타냄

tar -xf 파일명.tar: 연결을 해제함 -> 결과: 파일명1, 파일명2

mkdir 디렉토리명: 디렉토리 생성

tar -zxf ../test.tar.gz: 현재 디렉토리의 상위 디렉토리에 있는 test.tar.gz파일을 압축해제하고 연결을 해제한다

삭제

rm -rf 디렉토리명: 지정된 디렉토리 이하의 모든 파일 삭제

rm 파일명.확장자 파일명.확장자: 해당 파일명을 가진 파일들 삭제

현재 시간을 출력하는 커맨드: date

date +%y: 현재연도의 뒷자리 두자리만 출력

date +%Y: 현재연도 출력

date +%m: 현재 월 출력

date +%d: 현재 일 출력

date +%Y-%m-%d: 현재 년 월 일 출력

history: 지금까지 썼던 명령어들 출력

!(히스토리에 나온 번호): 해당 번호에 해당하는 명령어 실행

!! : 직전에 실행한 명령어 실행

exit : 쉘에서 빠져나오는 명령어, 결과적으로 쉘을 실행하고 있던 터미널 끝내는 명령어

bash: 또다른 bash 쉘 생성

echo 쓰고 싶은 글자: 글자 출력

ex) echo aaa -> 결과: aaa

echo 변수: 변수 출력

echo $PWD(환경변수) : PWD라는 변수의 값을 출력 : 현재 디렉토리 출력

echo $PATH(환경변수) : :콜론으로 연결된 디렉토리의 목록

env: 어떤 변수에 어떤 값이 들어가 있는지 보여줌

which 툴: 툴의 위치를 알려주는 명령어

관리자 권한으로 명령어(커맨드)를 실행할 수 있는 명령어: sudo

ex) sudo apt install hello - hello 패키지 설치

sudo apt remove hello - hello 패키지 삭제

apt list --installed: 내 pc에 설치 되어있는 패키지 리스트 조회

우분투 패키지 매니저: apt

텍스트에디터: 나노 / vim(vi) / ...

나노 이용법: 터미널 창에 nano + enter

나노 나가기: ctrl(^)+x

실행취소: alt(M) + U

블럭: shift + 방향키

----------------------------------------------------------------------

파일은 이름을 가지고 있음 - 경로와 이름이 공존 - 위치를 구분

하나의 파일엔 데이터를 가지고 있음(데이터를 담는 그릇): 기본적으로 디스크에 존재

변수(데이터를 담는 그릇): 메모리에 존재

하나의 시스템에는 엄청 많은 파일이 존재한다

운영체제는 하드웨어가 있으면 그 위에 바로 올라가는 최초의 소프트웨어

운영체제는 수많은 파일들을 관리해줘야한다

파일시스템: 파일을 관리하기 위한 시스템

리눅스는 tree형태의 계층 구조를 가지고 있음

윈도우 파일시스템: NTFS / FAT16, 32

리눅스 파일시스템: EXT2, 3, 4 / XFS

파일시스템마다 지원하는 기능이 다르다

운영체제의 시점으로 보는 파일의 종류

일반파일: 말 그대로 일반 파일, 텍스트 파일이나 동영상 파일 등이 일반파일에 속함

디렉토리: 윈도우의 폴더와 같은 개념인 디렉터리도 파일이라고 함

심볼릭 링크 파일: 어떤 다른 파일을 가리키는 파일. 윈도우의 바로가기와 같은 개념

블록 디바이스 파일: 블록 디바이스를 제어하기 위한 파일

문자 디바이스 파일: 문자 디바이스를 제어하기 위한 파일

하드웨어를 위한 소프트웨어 = 디바이스 드라이버(운영체제와 하드웨어의 중간): 디바이스를 구동/제어 함

디바이스 드라이버는 운영체제와 어플리케이션(디바이스 파일을 이용?)과 상호작용함

APP - DIVCEFILE(char/block) - DEVICE DRIVER(OS) - DEVICE(H/W)

파이프 파일: 파이프를 나타내는 파일, 프로세스간 통신에 사용됨

파이프(pipe): 구동중인 프로세스와 다른 프로세스가 통신을 하는 방식

두 프로세스가 파이프를 이용하여 통신을 하면 파이프 파일이 생성됨

소켓: 소켓을 나타내는 파일, 프로세스간 통신에 사용됨

프로페스와 또다른 프로세스가 통신을 할 때 소켓을 이용하여 할 수도 있음(파이프와 비슷한 기능을 가지고 있음)

디렉터리

/etc: 여러가지 설정 파일 - 일반 권한으로는 /etc 밑 파일을 건드릴 수 없음 sudo명령어 필요

파일구조: name - inode - data

아이노드(inode): 파일마다 유지되어야 하는 데이터 중 하나

하드링크(hard link):

(파일이름 - inode (원본과) 공유 - filedata (원본과) 공유)

소프트링크(softlink = symbolic link = symlink ): 윈도우의 바로가기 같은 역할

(파일이름 - inode - linkdata -> filename(원본))

빈 파일을 만드는 명령어: touch

touch hahahah -> hahaha라는 파일이 생김

rm hahaha -> hahaha파일 삭제

데이터가 들어간 파일 만드는 명령어: nano

nano testfile -> 내용작성 -> 저장 -> testfile생성

testfile 이름을 apple이라고 바꿈(사실상 경로를 바꾸는 것이라 볼 수 있음)

mv testfile apple -> tetfile 이름 사라지고 apple이라는 파일이 생김

mv apple /tmp/pineapple -> apple 파일 위치는 tmp 밑, 이름은 pineapple로 바뀜

mkdir 디렉토리명 -> 새로운 디렉토리 생성

rmdir 디렉토리명 -> 해당 디렉토리 삭제

단, 안에 파일이 없어야 삭제 가능

만약 비어있는 파일이나 다른 파일이 존재한다면

그 파일들을 다 지운 후 명령어를 실행해야 해당 디렉토리를 지울 수 있다.

rm -r(디렉토리 포함 안에 있는 콘텐츠)f(강제로) 디렉토리명 -> 강제로 파일이 들어있는 디렉토리 지울 때 사용

mv명령어는 파일과 디렉토리 똑같이 이용가능

파일의 inode 찾는 방법: ls -i, ls -ali

하드링크를 생성하는 명령어: ln

ex) ln pineapple hello: pineapple 파일의 inode를 hello(하드링크)라는 파일을 만들어 공유함

-> 이때 inode를 공유하기 때문에 hello를 변경하면 pineapple데이터도 같이 변경됨

파일의 속성을 확인하는 명령어: stat + 파일명

->(결과화면) modify(수정): 파일의 컨텐츠(내용)를 변경한 시간

change(변경): inode가 변경(링크의 개수의 변화 등)된 시간

소프트링크를 생성하는 명령어: ln

ex) ln -s pineapple hello

-----------------

사용자와 그룹

유저a 유저b 하나의 리눅스 시스템을 동시에 사용가능

파일 속성: 권한+소유권 -> permissions(mode) + Owner(user), Group

파일권한: 8진표기법, 의미표기법

cat /etc/passwd : 시스템에 등록되어있는 유저의 목록과 설정을 출력함

(한 줄당 하나의 계정)

cat /etc/group: 이 시스템에 존재하는 그룹에 대한 정보

uid: 앞에 나오는 정보가 계정명 (UID)

gid: 뒤에 나오는 정보가 그룹명 (GID)

사용자를 추가하는 명령어: sudo adduser(root권한 필요) 유저이름(추가할 계정 이름)

useradd

whoami: 내가 어떤 계정으로 로그인 했는지 알아보는 명령어

다른계정으로 실행되는 쉘 띄우는 명령어: su - 바꾸고싶은 다른 계정이름

이후 exit 명령어를 이용하면 쉘에서 로그아웃 하게됨 = 원래 계정으로 돌아감

계정 지우는 명령어: sudo deluser 삭제하고자 하는 계정 이름 + (--remove-home)

->주의: 삭제하고자하는 계정이 사용중이던 터미널 종료한 후 로그아웃이 된 뒤에 삭제 가능

그룹추가하는 명령어: adduser --group 그룹명

addgroup 그룹명

그룹 삭제하는 명령어: sudo groupdel 그룹명

그룹 조회 명령어: cat /etc/group

sudo adduser 추가할 계정명 --ingroup 추가한 계정이 속할 그룹명

권한 부여 명령어: chmod 664(바꾸고자하는 파일권한: 8진법으로 작성) 파일명

chmod g+w testfile(의미표기법)

스크립트: 순차적으로 실행되는 명령의 모음

bash script: 첫줄은 무조건 샤뱅(#!/bin/bash) 작성해야함

ex) #!/bin/bash

ls

pwd

whoami

bash script 실행하기: $home/pig/firstscript -실행안됨

1. ls -al 명령어를 사용하여 권한 확인 - excute 권한 있는지 확인

2. 없다면 권한 부여: chmod u+x (744) firstscript

chmod a(나, other, 그룹 모두에게 권한 부여)+x (755) firstscript

3. $home/pig/firstscript (절대경로)

(= ./ firstscript (상대경로): 현재 디렉토리에 있는 firstscript파일 지정)

4. firstscript에 적었던 명령어 순으로 결과값 출력

------------------------------

프로세스와 시그널

프로세스(process): 처리과정, 공정.../실행중인 프로그램 / 메모리 상에 있는 객체

프로그램: 저장장치(하드디스크, ssd...)에 저장되어 있는 실행이 가능한 파일 (디스크에 저장되어 있는 상태)

운영체제의 가장 중요한 역할 중 하나: 프로세스 관리

프로세스 스케쥴링

cpu의 관점에서 할 일들을 정해주는 것 -> 해야되는 일: (여러) 프로세스를 실행해 주는 일

<프로세스 상태>

프로세스 생성

프로세스 대기

프로세스 실행

프로세스 블로킹(응답이 올때까지 기다려야 되는 상황/프로세스가 외부에 있는 자원에 오퍼레이션을 하고 끝날때까지 기다리는 상황)

프로세스 종료

<프로세스 계층구조>

pid(process id): 유니크한 id 즉, 겹치지 않음

부모 프로세스와 자식 프로세스: 프로세스가 다른 프로세스를 만들면 만들어진 프로세스를 자식 프로세스라고 함(자식 프로세스는 여러 개일 수 있음)

init 프로세스: init프로세는 부모 프로세스가 없다. 그 이외의 모든 프로세스는 부모 프로세스가 존재함

처음 리눅스 월드 시작하면 프로세스와 관련한 초기화를 하게 됨 이때 init프로세스를 만들게 됨.

즉 아무 프로세스가 없는 단계에서 init프로세스를 만들게 됨

따라서 pid가 1임

프로세스 종료 처리: 자식 프로세스가 종료가 되었을 때 종료 처리를 해야되는 것

(자신이 사용하던 메모리 영역들을 깔끔하게 정리하지 못함)을 부모 프로세스가 그 영역을 처리 해줌

고아 프로세스와 좀비 프로세스: 부모 프로세스가 종료 처리를 잘 하지 못했을 때 발생함

고아 프로세스: 부모 프로세스가 먼저 종료가 되어 버려서 부모가 없는 프로세스가 된 경우

4441 문제: 종료 될 때, 종료 처리를 해줄 부모 프로세스가 없어짐...

이때, 고아프로세스가 생기면 입양을 해줌 부모프로세스를 init 프로세스로 바꿔줌

좀비 프로세스: 부모프로세스와 자식프로세스가 있을 때 자식프로세스가 종료될때까지 기다리거나 자기 일을 하다가 자식 프로세스가 죽었다는 시그널을 받으면 그때 자식의 종료처리를 해줄 수 있음

이때, 자식 프로세스가 종료가 되었을 때 위에서 말한 두가지 경우를 부모 프로세스가 안하고 있고 다른 일을 하고 있을 때 발생함.

즉, 부모 프로세스가 다른 일을 하고 있어서 제때 자식 프로세스의 종료처리를 하지 못했을 때 발생한다.

자식 프로세스는 부모 프로세스가 정리 해줄 때까지 기다리는 상태-종료되지 않아서 프로세스는 제대로 동작하지 않지만 리눅스 시스템에 남아있게 됨.

따라서 리눅스 시스템의 여러 자원을 끌어안고 있어 장기적으로 pid와 메모리를 계속해서 사용하기 때문에 리눅스 시스템에 가용할 자원들이 줄어들게 된다.

<프로세스 종료 상태>

프로세스 종료 상태란?

프로세스가 종료될 때 넘겨주는 정보

부모 프로세스에게 정보를 넘기게 됨 -> 종료처리가 성공인지 실패인지 부모 프로세스가 판단할 수 있게 됨

상태 정보

정상 종료 여부

시그널을 받아 종료되었는지 여부 및 시그널의 종류

코어 덤프를 생성했는지 여부(0은 성공을 의미, 다른 값은 실패를 의미)

표준 스트림

>>모든 프로세스가 생성될 때 가지고 있는 표준적인 스트림

표준입력 / 표준출력 / 표준에러

#0: stdin / #1: stdout / #2 stderr

파일 디스크립터

파일을 대변하는 객체(프로세스가 파일을 열어서 데이터를 저장/읽기를 하기를 원할 때 필요한 객체)

파일에 대한 작업(읽기/쓰기)을 하기 위해서는 파일 디스크립터를 열어야 함

프로세스로 열린 파일 디스크립터 목록을 관리

포어그라운드 프로세스와 백그라운드 프로세스

<포어그라운드 프로세스>

쉘의 표준 입력이 연결된 프로세스

표준 출력과 표준 에러도 쉘과 연결되어 있음

<백그라운드 프로세스>

쉘의 표준 입력이 연결되지 않은 프로세스

표준 출력과 표준 에러는 쉘과 연결되어 있음

데몬 프로세스 >> 악마 프로세스?

>> MIT 대학의 프로그래머가 만든 용어

>> ‘보이지 않는 곳에서 어떤 일을 하는 유령’에서 영감을 받음

데몬프로세스는 백그라운드 프로세스로 동작하기 위해 만들어진 프로세스

표준 스트림을 갖고 시작하지만 모두 닫아 버림 -> 쉘과의 입출력 교환 불가

부모 프로세스를 init프로세스로 변경

시그널: 커널이 어떤 사건 발생을 프로세스에 알리는 신호이다.

>>비동기 이벤트를 처리하기 위한 프로세스간 통신(IPC)

동기? >> 클라이언트쪽이 서버에 요청을 보내고 응답이 올 때까지 기다림

비동기? >> 요청을 보내고 응답이 올 때까지 다른 일을 함(아무 일도 안할 수도 있음)



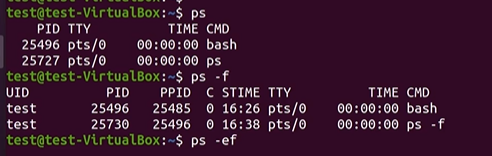
코어파일: 비정상적으로 종료될 때 프로세스가 가지고 있는 정보를 저장한 파일

코어덤프: 파일을 만드는 행위

프로세스 목록 조회 및 구조 파악

프로세스 목록 조회: ps -> 현재 터미널에서 실행된 프로세스의 목록 보여주는 명령어

ps –f : 위의 경우보다 더 많은 정보 보여줌



ps –e : 모든 프로세스 목록 조회

ps –ef: 위의 경우보다 더 많은 정보 보여줌

echo $$: 현재 프로세스의 id를 출력하는 명령어

cf)새로실행하면 같은 파일이라도 pid가 계속해서 달라짐

프로세스 종료 상태

직전에 수행한 명령어의 종료 상태를 받아오는 명령어: echo $?

>> 결과ex): 0 -> 성공

>> 결과ex): 2 -> 실패

시그널 전송

프로세스를 죽이는 시그널을 보내는 명령어: kill

SIGKILL(9): 강제종료할 때 쓰임

ex) kill -9 25916(pid) -> 죽었음-> pid가 25916인 프로세스(실행중인) 파일을 강제종료 시킴 -> echo $? -> 137

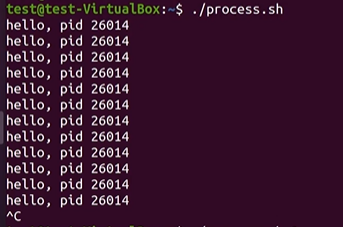
kill –TERM 25916 -> 종료됨 -> echo $? -> 143

ctrl+c(bash가 sigint를 보내게 됨) -> echo $? -> 130

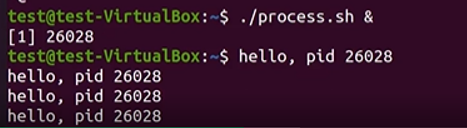
kill –ABRT 25916-> 정지됨(코어덤프됨) -> 코어파일(core) 생김 -

포어그라운드 프로세스와 백그라운드 프로세스

기본 실행 방법(포어그라운드: 사용자의 입력과 연결이 되어있음)



백그라운드로 실행하는 방법

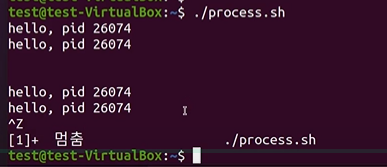


이때 모든 사용자의 입력값은 process.sh과 연결되어 있는 것이 아니라 프롬포트(bash)와 연결되어 있음

따라서 ctrl+c 혹은 엔터키와 같이 입력값을 넣어도 영향을 줄 수 없고

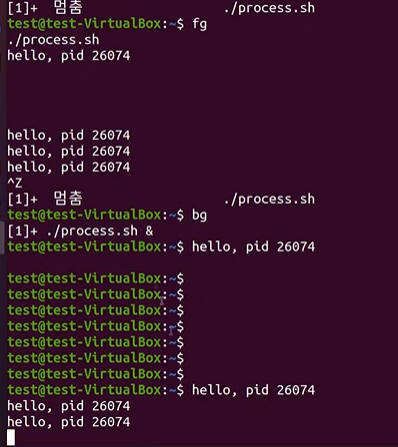
계속해서 hello, pid 26028이 반복해서 출력됨 -> 포어그라운드로 바꾸기: fg + enter

./process.sh -> 실행 -> ctrl+z -> 멈춤



fg: 다시 계속해서 실행

bg: 백그라운드 실행으로 바뀜



jobs: 백그라운드 프로세스들을 조회할 수 있는 명령어



[2] 파일만 포어그라운드 실행으로 바꾸는 명령어: fg %2 + enter

-> ctrl+c 누르면 [2]는 종료됨 -> [1], [3]만 계속해서 실행됨

-------------------------------

리디렉션과 파이프라인

리디렉션

- 화면상에 출력될 무언가를 파일로 저장하거나

- 파일 디스크립터로 보내거나

- 파이프라인을 통해 다른 프로세스에 파일 디스크립터로 연결

출력 리디렉션(>): 화면에 출력되는 것을 다른 곳(파일)으로 보냄

[ n ]>[ | ] word(파일명)

ex)ls > filename / ls >| filename / ls 2> filename / ls 2>| filename

표준 출력 스트림 번호: 1

표준 에러 스트림 번호: 2

noclobber 옵션 설정 시 덮어쓰기 시도 시 에러 발생

단 >| 사용시 noclobber 옵션 설정 여부와 관계없이 덮어쓰기 성공

추가 모드 출력 리디렉션: >>

[n] >> 파일명

표준 출력 스트림 번호: 1

표준 에러 스트림 번호: 2

파일 디스크립터로 리디렉션: >&

[n] >& [FD]

>과 동일하지만 대상 파일 대신 대상 파일 디스크립터 지정

표준 출력과 표준 에러를 한꺼번에 출력하고 싶을 때 자주 사용

표준 출력 및 표준 에러 동시 리디렉션: &>

&> 파일명

파일 디스크립터 리디렉션 문법이 귀찮고 어렵다

좀 더 편하게 사용 가능

입력 리디렉션: <

[n] < 파일명

파일의 내용이 지정된 스트림(n)으로 리디렉션

n이 생략되면 표준 입력(fd 0)을 입력

wc: (툴)워드카운트 / wc 친 다음 입력함

맨 마지막 숫자 3개: wc가 출력한 결과(라인 수, 단어 수(띄어쓰기 기준?), 캐릭터의 수)

wc 다음에 표준입력 +ctrl+d

here documents(입력 리디렉션의 일종): <<

[commend] << [-]DELIM

DELIM

프로그램의 표준 입력으로 multi-line string 전달

코드 블록의 내용이 임시 파일로 저장됐다가 프로그램의 표준 입력으로 리디렉션

DELIM은 다른 단어로 변경 가능(의미상 EOF, END 등)

<<- 사용되면 라인 앞쪽의 tab 문자가 제거됨

cat > 파일명 << DELIM(EOF): 입력한 내용이 파일명으로 저장됨

Here strings: <<<

[commend] <<< word

here document의 한 줄 버전

ex) cat > hellotext <<< "hello world there"

다른 방법(echo이용)

echo "hello world there" > hellotext

파이프라인: 프로그램 간 상호작용

파이프라인: |

command1 [ | command2]

command1의 표준 출력이 command2의 표준 입력으로 연결됨

command1 [ |& command2 ]

command1의 표준 출력과 표준 에러가 command2의 표준 입력으로 연결됨

각 command는 sub-shell에서 실행됨

전체 파이프라인의 exit status는 마지막 실행 커맨드의 exit status임

pipefail 옵션이 설정되어 있다면 실패한 커맨드가 존재하면 실패로 처리됨