== REPORT ==

<ROS 1일차>

1. 리눅스 & git 보고서 작성

	robit 18 기 인턴
학번	2023741059
이름	홍지현

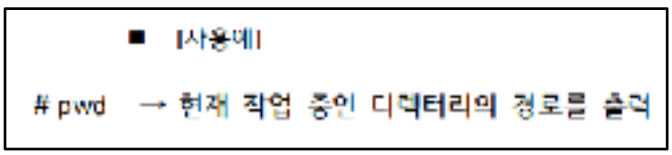
Linux 명령어 정리

#파일 시스템 탐색

1. pwd

-Print Work Directory

-현재 작업 중이거나 사용자가 있는 디렉터리의 경로를 볼 수 있습니다.

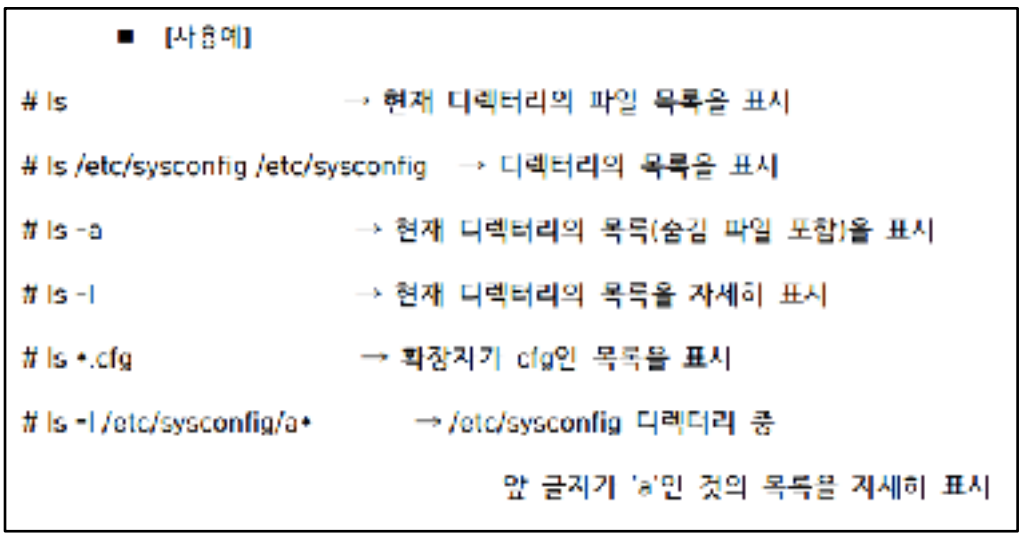


2. ls

-List Segments

-파일과 디렉터리의 모든 정보를 제공하며 특정 디렉터리와 특정 파일의 내용도 제공합니다. 그리고 옵션을 사용하여 탐색이 가능합니다.

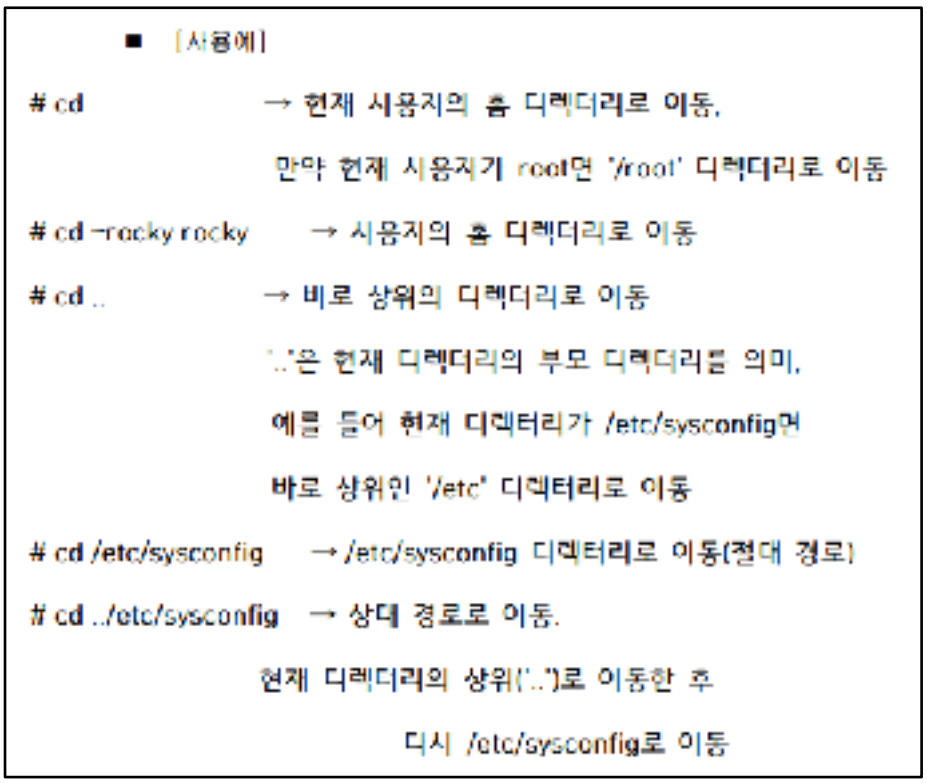
- a : "."으로 시작하는 숨긴 파일들을 포함하여 모든 디렉토리와 파일들을 보여준다.
- l : 디렉토리와 파일의 권한을 보여주는 긴 형식의 목록을 출력한다.
- s : 파일의 크기를 출력한다. 디렉토리는 0으로 출력된다.
- S : 파일들을 크기의 내림차순으로 출력한다. (제일 큰 파일이 맨 위)
- t : 파일과 디렉토리의 시간 내림차순으로 출력한다. (가장 최근 수정된 파일이 맨 위)



3. cd

-Change Directory

-인자값 없이 사용하게 된다면 홈 디텍터리(~)로 이동하는 명령을 수행합니다. cd 뒤에 상대 경로(.., ./tmp)나 절대 경로(/Uers/Documents)를 붙여주면, 해당 경로로 이동하는 명령을 수행합니다.



##상대 경로, 절대 경로##

#절대 경로#

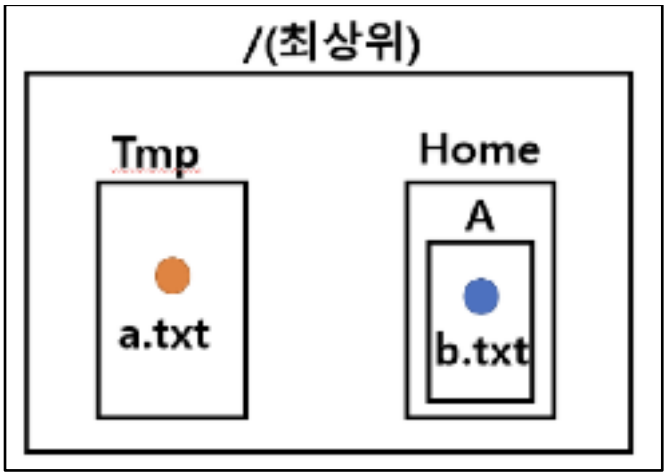
-리눅스의 디텍터리(파일 저장소)는 계층구조

-디렉터리에 있는 최상위 디텍터리는 /이며, 최상위 디렉터리부터 파일명에 이르는 경로를 절대 경로라고 합니다.

#상대 경로#

-현재 자신이 위치한 디렉터리를 기준으로 하는 경로를 상대경로라고 합니다. ./는 현재 디렉터리 ../는 상위 디렉터리를 의미합니다.

#example#



1. a.txt와 b.txt의 절대 경로

-절대 경로는 최상위 디렉터리부터 나타내는 것으로,

a.txt -> /Tmp/a.txt b.txt -> /Home/A/b.txt

2. a.txt와 b.txt의 상대 경로

-파란색 동그라미가 현재 위치하고 있는 디렉터리라고 가정한다면

./../../Tmp/a.txt

-주황색 동그라미가 현재 위치하고 있는 디렉터리라고 가정한다면

./../Home/A/b.txt

** 상대 경로 작성 시 현재 디렉터리를 의미하는 ./는 생략이 가능합니다.

4. mkdir

-Make Directory

-새로운 디렉터리를 생성하는 명령를 수행합니다.

■ [사용예]

mkdir abc → 현재 디렉터리 아래에 /abc 이름의 디렉터리 생성

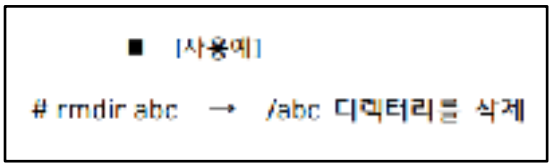
mkdir -p /def/fgh → /def/fgh 디렉터리를 생성. 만약 /fgh 디렉터리의 부모 디렉터리인

/def 디렉터리가 없다면 자동 생성(p는 Parents의 약자)

5. rmdir

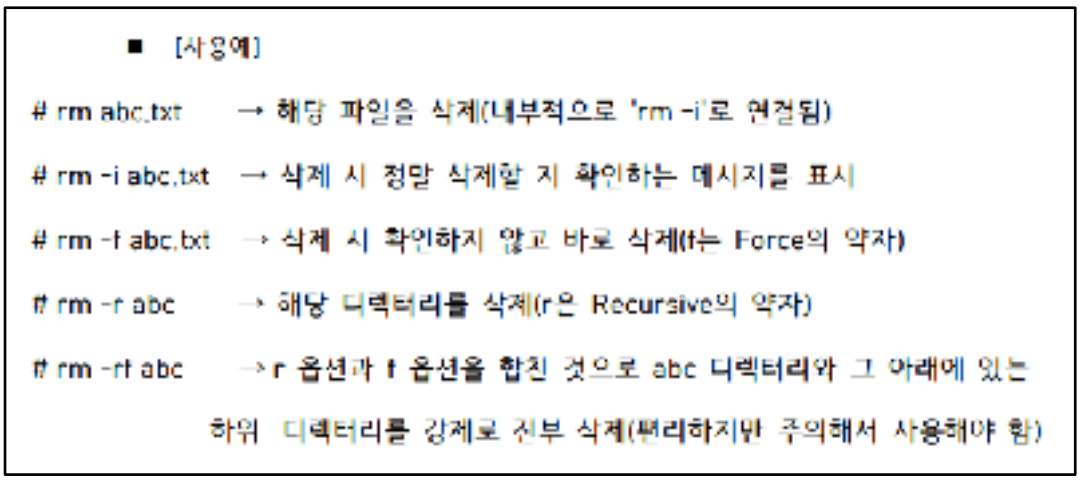
-Remove Directory

-빈 디렉터리를 삭제하는 명령을 수행합니다. rmdir로 삭제하려는 디렉터리가 비어있지 않을 경우에는 사용할 수 없습니다.



** 파일이나 디렉터리가 담긴 디렉터리를 삭제할 때는 rm -rf 명령어를 사용해야 합니다.

6. rm



-위 내용 참고

#시스템 조작

1. uname

-Unix Name

-이름,버전 및 기타 시스템 세부 사항과 같은 시스템 정보를 얻기 위한 기본 linux 명령어입니다. 이 명령으로 OS 및 커널 버전을 빠르게 확인할 수 있으며, 시스템의 명령 길이를 확인할 수 있습니다.

- uname -a : 시스템의 모든 정보를 출력
- uname -m : 시스템 하드웨어 타입 정보
- uname -n : 사용중인 네트워크 호스트 이름 확인
- uname -p : 프로세서 정보 확인
- uname -r : 커널 릴리즈 확인 (운영체제 배포 버전)

- uname -s : 커널명 확인
- uname -v : 커널 버전 확인

2. ps

-Process Status

-현재 시스템에서 실행 중인 프로세스를 시각화 해줍니다. 시스템 리소스를 분석하는데 사용되는 수단이며, 터미널을 통해 기본적으로 시스템 프로세스를 조작할 수 있습니다. ps는 기본 및 최상위 리눅스 모니터링 도구 중 하나이며, ps외에 실시간으로 프로세스의 상태롤 볼려면 top 명령어를 쓰는 방법도 있습니다.

- ps : 명령어를 입력한 순간의 프로세스 정보 출력. 현재의 Shell에 의해서 수행된 프로세서들을 조회할 수 있습니다.
- ps -f : Full Listing. 프로세스 정보에 대해 상세하게 출력합니다. (uid, pid, parent pid, CPU 사용량, 시작 시간 등등)
- ps -l : Long Listing. 프로세스의 기본 정보 및 프로세스가 사용하고 있는 OS 자원(CPU, Memory)의 활용 규모, OS의 리소스 활성화 상태 등을 출력합니다.
- ps -o : Optional Listing. 프로세스의 상태값 중 출력을 원하는 컬럼값을 지정하여 출력합니다.

-List에서 각 칼럼의 정보

- UID : User ID. 일반적으로 컴퓨터의 최초 사용자를 가리키는 UID 501을 출력합니다.
- PID : Process ID. 동일한 프로그램이지만 다른 PID를 부여 받을 수 있습니다.
- PPID : Parent Process ID. 해당 프로세스를 실행시킨 부모 프로세스의 PID
- TTY : The Controlling Terminal For the Process, 터미널 번호
- Time : 시작 시간
- CPU : 해당 프로세스가 사용한 CPU 시간의 양
- CMD : 실행 중인 명령 커맨드

-현재 Shell 뿐만 아니라 모든 프로세스를 출력하고 싶을 때는 -e 옵션 또는 -aux 옵션을 사용한다. 리스팅 수가 너무 많을 수 있으니 "| less" 명령을 포함시키면 좋습니다.

-유닉스 옵션 : -a, -e, -u. "man ps" 명령어로 내장 매뉴얼 확인

-BSD 옵션 : a, u, x.

- a : 터미널에서 실행한 프로세스의 정보 출력

- u : 프로세스 소유자 이름, CPU, 메모리 사용량 등 상세정보 출력
- x : 시스템에서 실행 중인 모든 프로세스의 정보 출력

3. kill

-자원으로 제한으로 인해 멈춘 프로세스를 중지하는 명령어 입니다. ps로 프로세스 목록을 확인한 후 멈추고자 하는 프로세스의 PID를 사용하면 됩니다.

$ kill [PID}

$ killall [Process Name]

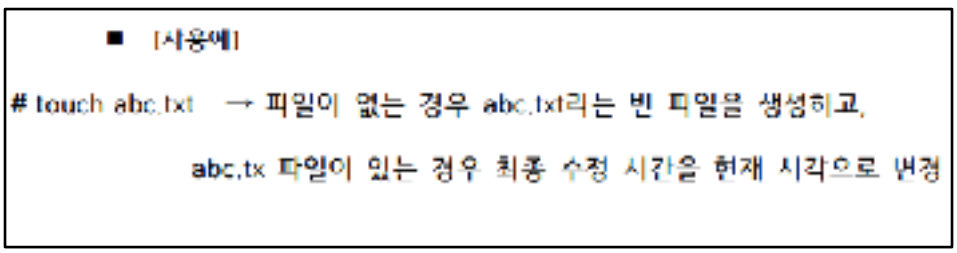
4. shutdown

-시스템을 종료하는 명령어 입니다. 현재 접속 중인 모든 사용자에게 시스템이 종료된다는 메시지를 보낼 수 있습니다.

#파일 관리

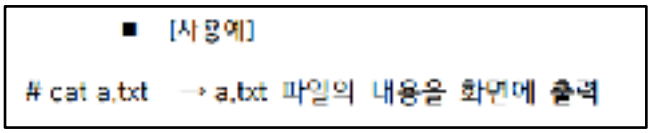
1. touch

-유효한 빈 파일을 작성하기 위한 명령어 입니다. 이동 중에 파일을 생성하고 요구 사항에 따라 나중에 또는 실시간으로 파일을 채울 수 있습니다. 타임 스템프를 변경하기 위한 명령이기도 합니다.



2. cat

-새 파일을 작성하고 터미널에서 파일 내영을 보고 출력을 다른 명령 도구나 파일로 리디렉션을 하는 데 사용됩니다.



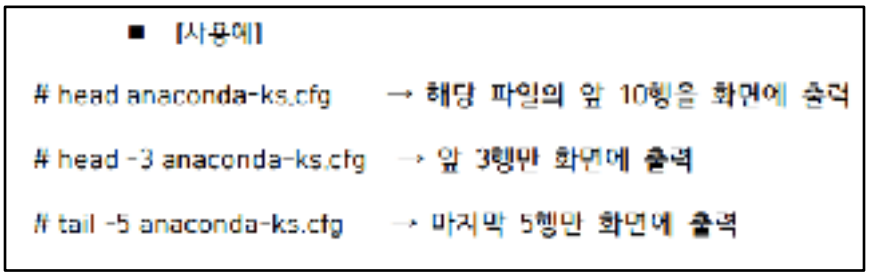
3. head

-터미널에서 직접 파일 또는 파이프 된 데이터의 시작을 볼 수 있습니다. 텍스트 처리를 많이 사-

용하는 리눅스 명령어 중 하나입니다. 기본적으로 파일의 맨 위부터 10줄을 출력합니다.

4. tail

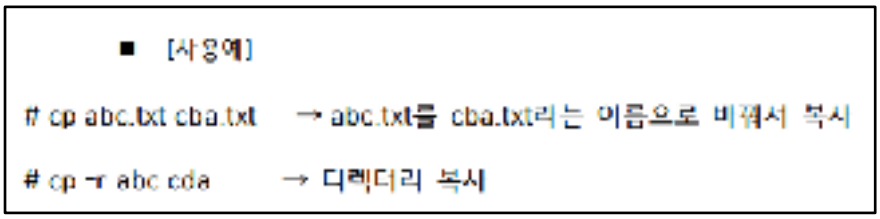
-파일의 마지막 행을 기준을 지정한 행까지의 파일 내용 일부를 출력합니다. 기본적으로 마지막 10줄을 출력합니다.



5. cp

-Copy

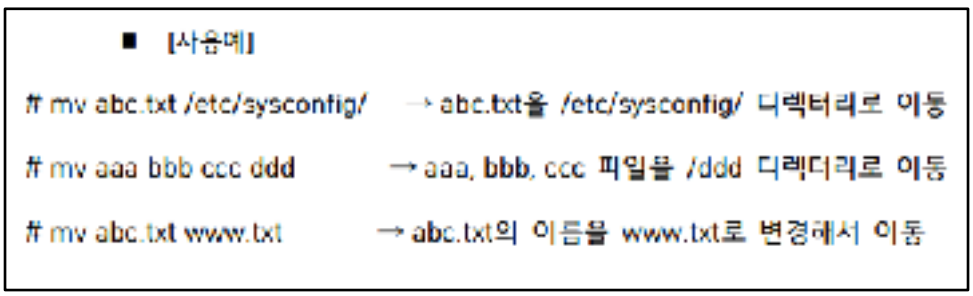
-시스템에서 파일이나 디렉터리를 한 폴더에서 다른 폴더로 복사하도록 지시하는 명령어 입니다. 해당 명령어를 사용하여 터미널에서 바로 여려 파일을 디렉터리로 복시할 수 있습니다.



6. mv

-Move

-GUI에서 수행하는 절단 작업("잘라내기")를 보완합니다. cp는 기존 위치에 파일이 그대도 있으면서, 지정한 다른 위치로 파일을 복사하는 개념입니다. 반면 mv는 기존 위치에 있는 파일을 없에고. 지정한 다른 위치에 파일을 옮기는 작업입니다. -f 옵션을 사용하여 큰 파일을 전송할 수 있습니다.



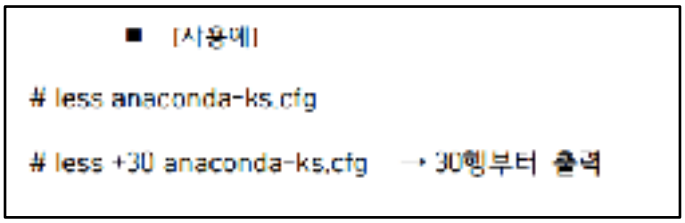
7. comm

-두 개의 파일을 공통 행과 구별되는 행으로 비교할 수 있습니다.

8. less

-터미널 세션을 방해하지 않으면서 파일 내에서 양방향으로 탐색할 수 있습니다.

- 위, 아래 방향키 : 한줄 위, 아래 이동
- 스페이스바 : 한페이지 아래로 이동
- /[keyword] : 파일 내에서 [keyword] 찾기
- q : 나가기



9. ln

-Link

-특정 파일에 대한 심볼릭 링크를 만들기 위한 명령어

10. cmp

-Compare

-두 파일을 비교하그 그 결과를 표준 출력 스트림에 인쇄하는 명령어 입니다. 해당 명령어는 comm과 함께 대량의 텍스트 파일을 정기적으로 처리하는 사용자들이 가장 많이 사용하는 명령어 입니다.

11. dd

-파일을 한 유형에서 다른 유형으로 복사 및 변환하기 위해 사용하는 명령어 입니다.

12. alias

-직접 파일의 다른 문자열로 단어를 바꿀 수 있기에 많이 사용되는 명령어로, 쉘을 사용자 정의하고 환경 변수를 조작할 수 있는 최상위 터미널 명령어입니다.

#리눅스 명령어 검색 및 정규 표현식

1. find

-파일을 검색하는데 가장 많이 사용되는 명령어 입니다. 해당 명령어를 통해 파일 권한, 소유권, 수정 날짜, 크기 등과 같은 특정 기준에 따라 파일을 검색할 수 있습니다.

2. which

-해당 명령어는 검색하려는 모든 파일이 실행 파일일 경우 유용합니다.

3. locate

-특정 파일의 위치를 찾는데 사용되는 명령어. Linux 시스템에서 특정 파일의 위치를 모를 때 활용 할 수 있는 가장 간단한 터미널 명령입니다.

4. grep

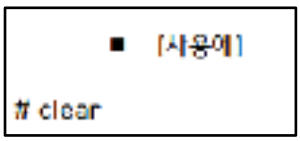
-대량의 텍스트 파일에서 패턴을 검색할 때 사용할 수 있는 가장 명령어 입니다. 찾고자 하는 패턴을 입력으로 받아 특정 패턴에 지정된 파일을 검색합니다.

5. sed

-지정된 부분을 교체하여 파일 또는 스트림의 각 줄을 조작하는데 가장 많이 사용되는 명령어 중 하나이다. 많은 양의 텍스트 데이터를 다루고 이동 중에도 변경해야 하는 사용자들이 많이 사용한다.

6. clear

-터미널 화면을 지우는 데 사용하는 명령어 입니다.



7. echo

-터미널 콘솔에 특정 텍스트를 출력할 수 있는 명령 줄 유틸리티 입니다. echo의 출력을 다른 터미널 명령으로 파이프 할 수도 있습니다.

8. sort

-정렬 명령어. 사전 순 또는 역순으로 파일을 정렬해야 할 때 사용합니다.

9. sudo

-권한이 없는 사용자는 낮은 수준의 권한이 필요한 파일에만 액세스하고 수정할 수 있습니다. 해당 명령어를 사용하면 일반 사용자 계정에서 root에 액세스할 수 있다.

10. chmod

-시스템 파일 또는 객체의 액세스 권한을 변경하거나 수정하는데 사용하는 명령어입니다. 해당 명

령어는 사용자로부터 다양한 매개 변수 세트를 취할 수 있으며, 파일 권한 변경에 따라 다릅니다

11. chown

-chown 명령은 chmod 명령과 매우 유사합니다. 그러나 액세스 권한을 변경하는 것이 아닌, 파일 또는 디렉터리의 소유권을 변경하는 명령어입니다. chmod와 chown 명령어는 모두 root 권한이 필요합니다.

git 명령어 정리

Github란?

- 소프트웨어 개발 플랫폼 및 소스코드 호스팅 서비스
- Git을 통한 저장소 호스팅을 지원
- GUI 제공
- 영리적인 서비스와 오픈소스를 위한 무상 서비스 모두 제공

Git이란?

- 오픈 소스 버전 관리 시스템(VCS: Version Control System)

  ->버전을 관리할 수 있는 수단

  ->업데이트 사항 등을 그때, 그때 바로 반영 할 수 있는 시스템
- 분산 버전 관리 시스템(DVCS)

  ->소프트웨어를 개발하는 소스 코드를 효과적으로 관리할 수 있게 도와주는 무료 오픈 소프트웨어
- 로컬에서 코드 및 버전 관리
- 텍스트 명령어 입력 방식

버전 관리 시스템이란?

버전 관리 시스템(이하 VCS)은 파일 변화를 시간에 따라 기록했다가 나중에 특정 시점의 버전을 다시 꺼내 올 수 있는 시스템이다. 많은 개발자들이 소스 코드를 관리할 때 VCS를 사용하지만, 거의 모든 컴퓨터 파일의 버전을 관리할 수 있다.

VCS를 사용하면 각 파일을 이전 상태로 되돌릴 수 있고, 프로젝트를 통째로 이전 상태로 되돌리거나 시간에 따라 수정 내용을 비교해 볼 수도 있다. 누가 문제를 일으켰는지 추적할 수 있고 누가 언제 만들어낸 이슈 인지도 알 수 있다. 또한 파일을 잃어버리거나 잘못 고쳤을 때도 쉽게 복구할 수 있다.

분산 버전 관리 시스템이란? Distributed VCS

분산 버전 관리는 단순히 파일의 마지막 스냅숏을 사용(checkout) 하지 않는다. 저장소를 히스토리와 더불어 통째로 복제한다. 서버에 문제가 생기면 이 복제물로 다시 작업을 시작할 수 있다. 클라이언트 중에서 아무거나 골라도 서버를 복원할 수 있다. 즉, 복제물은 모든 데이터를 가진 진정한 백업이다.

---

git init 명령어란?

Git은 소스 코드의 버전 관리를 도와주는 도구입니다. 일반적으로 프로젝트 단위로 Git 저장소를 만들어 사용하며, 소스 코드 파일을 Git으로 관리하기 위해서는 먼저 Git 저장소를 초기화해야합니다. 이 때 저장소를 초기화하기 위해 사용하는 명령어가 git init입니다.

git init 명령어로 Git 코드 저장소 새로 만들기

프로젝트를 시작하고 소스 코드를 Git으로 관리하고자 한다면 먼저 git init으로 저장소를 초기화해야합니다. 기본적인 사용법은 간단합니다. 저장소로 사용하고자 하는 디렉터리로 이동한 다음 git init을 인자없이 실행하면 됩니다.

```
git init
```

중요한 것은 먼저 저장소로 사용할 디렉터리로 이동해야한다는 점입니다. 이 때 디렉터리에 파일의 존재 유무는 중요하지 않습니다. Git 저장소를 초기화하더라도 디렉터리에 있는 내용을 자동으로 Git 저장소에 추가하지는 않기 때문에 빈 저장소로 초기화되는 것은 같습니다. 여기서는 /tmp 아래에 빈 디렉터리를 하나 만들고, git status 명령어로 Git 저장소 디렉터리가 아닌 것을 확인해 보겠습니다.

```
$ mkdir -p /tmp/git-init

$ cd /tmp/git-init

$ git status

fatal: not a git repository (or any of the parent directories): .git
```

아직 git init을 실행하지 않았기 때문에, Git 저장소가 아니라는 에러 메시지가 출력됩니다. 이 메시지에서 중요한 힌트를 얻을 수 있는데, Git 저장소는 .git 디렉터리로 관리됩니다.

```
$ ls -al

total 0

drwxr-xr-x 3 lainyzine staff 96 5 15 15:14 .

drwxr-xr-x 24 lainyzine staff 768 5 15 15:13 ..
```

먼저 디렉터리가 비어있는 것을 확인하기 위해 ls로 파일 목록을 출력해 봅니다.

아무 파일도 없습니다. 이 상태에서 git init을 실행합니다.

```
$ git init

Initialized empty Git repository in /tmp/git-init/.git/
```

현재 디렉터리 아래에 .git 디렉터리를 만들고 여기에 Git 저장소를 초기화한다는 것을 알 수 있습니다.

첫 커밋 이전의 Git 저장소 상태 확인하기

저장소가 초기화되면 어떤 변화가 있을까요? 먼저 한 번 더 ls를 실행해보겠습니다.

```
$ ls -al

total 0

drwxr-xr-x 3 lainyzine staff 96 5 15 15:16 .

drwxr-xr-x 24 lainyzine staff 768 5 15 15:13 ..

drwxr-xr-x 10 lainyzine staff 320 5 15 15:16 .git
```

.git 디렉터리가 생성된 것을 알 수 있습니다. 바로 여기서 Git 저장소가 관리됩니다.

이번에는 Git 저장소의 상태를 확인하는 git status와 커밋 로그를 보여주는 git log를 실행해보겠습니다

```
$ git status

On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

$ git log

fatal: your current branch 'master' does not have any commits yet
```

Git 저장소가 초기화 되었기 때문에 저장소 정보가 출력됩니다만, 메시지를 요약해보면 아직 저장소에 커밋이 없다는 의미입니다.

git clone 명령어 기초와 GitHub 저장소 복제

혼자서 개발하는 경우에는 로컬에서 Git 저장소를 초기화해서 사용하면 됩니다만, 협업을 하는 경우나 인터넷에 소스 코드를 공개하는 경우 GitHub에 저장소를 만들고 이 저장소를 클론해서 작업하는 방식을 주로 사용합니다. GitHub 계정이 있는 경우 아래 페이지에서 저장소를 생성할 수 있습니다.

클론을 하려면 Git 저장소의 주소를 알아야합니다. 화면에서 Code 버튼을 클릭하면 Git 저장소 주소가 나타납니다. Git 저장소의 주소는 HTTPS와 SSH 방식으로 제공되며 다음과 같은 형식을 따릅니다. (메뉴에서 알 수 있지만, ZIP 압축 파일로 다운로드 받을 수도 있습니다.)

HTTPS와 SSH 형식의 GitHub 저장소 주소는 아래와 같습니다.

```
# HTTPS 형식

https://github.com/[USERNAME]/[REPOSITORY_NAME].git

# SSH 형식

git@github.com:[USERNAME]/[REPOSITORY_NAME].git
```

[USERNAME]은 GitHub 사용자 이름으로, [REPOSITORY_NAME]은 저장소를 생성할 때 지정한 저장소 이름이됩니다.

git clone 명령어 형식

저장소를 복제해오기에 앞서 git clone 명령어의 기본적인 사용법을 소개합니다.

```
git clone [REPO_URL] [DIR]
```

[REPO_URL]에는 클론해올 저장소의 주소를 지정해줍니다. [DIR] 인자는 저장소를 로컬에 복제할 위치를 지정합니다. [DIR] 생략 가능하며, 특별한 이유가 없다면 보통 생략합니다.

HTTPS 프로토콜로 Git 저장소 clone

적절한 디렉터리로 이동한 후에 다음 명령어를 실행해줍니다.

```
$ git clone https://github.com/lainyzine/git-clone.git

Cloning into 'git-clone'...

remote: Enumerating objects: 3, done.

remote: Counting objects: 100% (3/3), done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), done.
```

저장소 복제가 성공적으로 진행되었습니다. 디렉터리에 들어가보면 Git 저장소를 관리하는 .git 디렉터리와 README.md 파일이 생성된 것을 확인할 수 있습니다.

```
$ cd git-clone

$ ls -al

total 8

drwxr-xr-x 4 lainyzine staff 128 Apr 25 14:41 .

drwxr-xr-x 3 lainyzine staff 96 Apr 25 14:41 ..

drwxr-xr-x 13 lainyzine staff 416 Apr 25 14:42 .git

-rw-r--r-- 1 lainyzine staff 11 Apr 25 14:41 README.md
```

git add 명령어 사용법

Git은 변경사항을 커밋 단위로 관리합니다. 커밋에는 사용자가 원하는만큼만 딱 변경사항을 반영

할 수 있는데, 이를 위해서 사용하는 특별한 공간이 바로 스테이징 영역입니다.

Git에 익숙하지 않더라도 커밋을 만들 때 git add . 명령어를 사용하곤 합니다. git add는 Git 저장소에 새로운 파일이나 수정사항을 커밋하기 전에 스테이징 영역에 변경 사항을 추가하기 위한 명령어입니다. git add는 커밋을 만들기 위한 필수적이 과정인 동시에, 커밋을 의도한대로 만들기 위해서는 스테이징 영역 개념과 git add 명령어의 역할을 정확히 이해해야합니다.

git add 사용법: 저장소 스테이징 영역에 파일 추가하기

Git에서는 저장소의 변경사항을 바로 커밋하지 않습니다. Git 저장소에는 커밋에 반영할 변경사항을 스테이징 영역이라는 개념이 있습니다. 사용자는 Git 저장소 전체 변경사항 중에서 다음 커밋을 생성할 때 반영할 변경사항을 골라서 스테이징 영역에 반영합니다. 이 때 사용하는 명령어가 바로 git add입니다. 그리고 스테이징 영역의 변경사항들을 커밋으로 만드는 명령어가 git commit 입니다.

파일을 직접 지정하는 방식: git add <FILE>

앞에서 살펴보았듯이 git add의 기본적인 사용은 인자로 추가하고자 하는 파일을 지정하는 방법입니다.

```
git add <FILE>
```

git add는 스테이징 영역에 추가하고자 하는 여러 파일을 한꺼번에 추가하는 것도 가능합니다.

```
git add <FILE1> <FILE2> ...
```

git add .: 현재 디렉터리 내의 모든 변경사항을 스테이징 영역에 추가

가장 많이 사용하는 경로 중 하나가 바로 .(점)입니다. .은 현재 디렉터리를 가리키는 특별한 이름입니다. 따라서 git add .을 입력하면 현재 디렉터리 아래의 모든 변경사항이 스테이징 영역에 추가됩니다.

```
git add .
```

-A 옵션: Git 저장소의 모든 변경사항을 스테이징 영역에 추가

Git 저장소의 모든 변경사항(추가, 수정, 삭제)을 스테이징 영역에 추가하고 싶다면 git add . 대신 -A 옵션을 사용할 수 있습니다.

```
git add -A
```

git status 명령어란?

Git은 분산 버전 관리 도구로 Git 저장소 디렉터리의 변경사항을 추적하고 커밋 단위로 관리할 수

있도록 도와주는 도구입니다. Git은 다양한 서브커맨드들을 가지고 있습니다만, 이중에서도 가장 많이 사용하는 명령어가 바로 git status입니다.

git status 서브커맨드 이름에서 유추할 수 있듯이 Git 저장소의 상태를 출력하는 명령어입니다. 이 때 상태라는 것은 좀 더 정확히 표현하면 Git 저장소의 HEAD, 워킹트리(Git 저장소의 디렉터리), 그리고 스테이징 영역을 비교한다는 것을 의미합니다.

HEAD와 워킹트리의 상태를 비교하고, 커밋하고 싶은 내용을 스테이징 영역으로 옮기고(git add), 이를 커밋(git commit)하는 게 Git의 가장 중요한 기능입니다. 이 글에서는 git status 명령어의 사용법을 해설합니다.

git status 명령어: 기본 형식

앞에서 살펴보았듯이 git status 명령어는 아래와 같이 대부분의 경우 옵션이나 인자 없이 실행하는 경우가 대부분입니다.

```
git status
```

자주 사용되는 형식은 아니지만 git add와 마찬가지로 인자로 디렉터리나 파일을 지정하는 것도 가능합니다. 이 경우 해당 파일에 대해서만 결과를 출력해줍니다.

```
git status <FILE1> <FILE2> ...
```

예를 들어 Git 저장소에서 README.md 파일의 상태만 보고 싶다면 아래와 같이 git status를 실행합니다.

```
git status README.md

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: README.md
```

README.md 파일의 현재 상태만 보여줍니다.

로컬에서 기본값으로 사용할 Git 사용자 이름과 이메일 설정

(global 옵션)

현재 시스템의 모든 Git 작업에 사용할 사용자 이름(user.name)과 이메일을 설정하고자 한다면, global옵션을 사용해 git config 명령어를 실행해줍니다.

git config --global user.name "Your Name"

git config --global user.email you@example.com

Git 저장소 초기화와 Git 저장소의 스테이징 영역에 대해 이해

[1] 먼저 git init

[2] 커밋을 만들기 전에 반드시 반드시 사용자 이름과 이메일을 지정해야합니다. 이 정보는 누가 이 변경사항을 만들었는지를 기록하한 정보로, 커밋에 포함됩니다. 여기서는 현재 Git 저장소에만 적용되도록 설정했습니다.

[3] 다음으로 중요한 부분은 git add 입니다. Git은 현재 디렉터리의 모든 변경사항을 자동적으로 커밋하지 않습니다. 사용자가 커밋하고 싶은 변경사항을 선택할 수 있는데 이 때 사용하는 명령어가 바로 git add 입니다. 이 명령어로 커밋하고 싶은 변경사항만 스테이징 영역에 추가할 수 있습니다.

[4] 이 과정이 모두 준비되고 나서야, 이제 커밋을 만들 차례입니다. 아래는 커밋을 생성할 때 가장 많이 사용하는 형식입니다.

git commit -m 'Initialize repository'

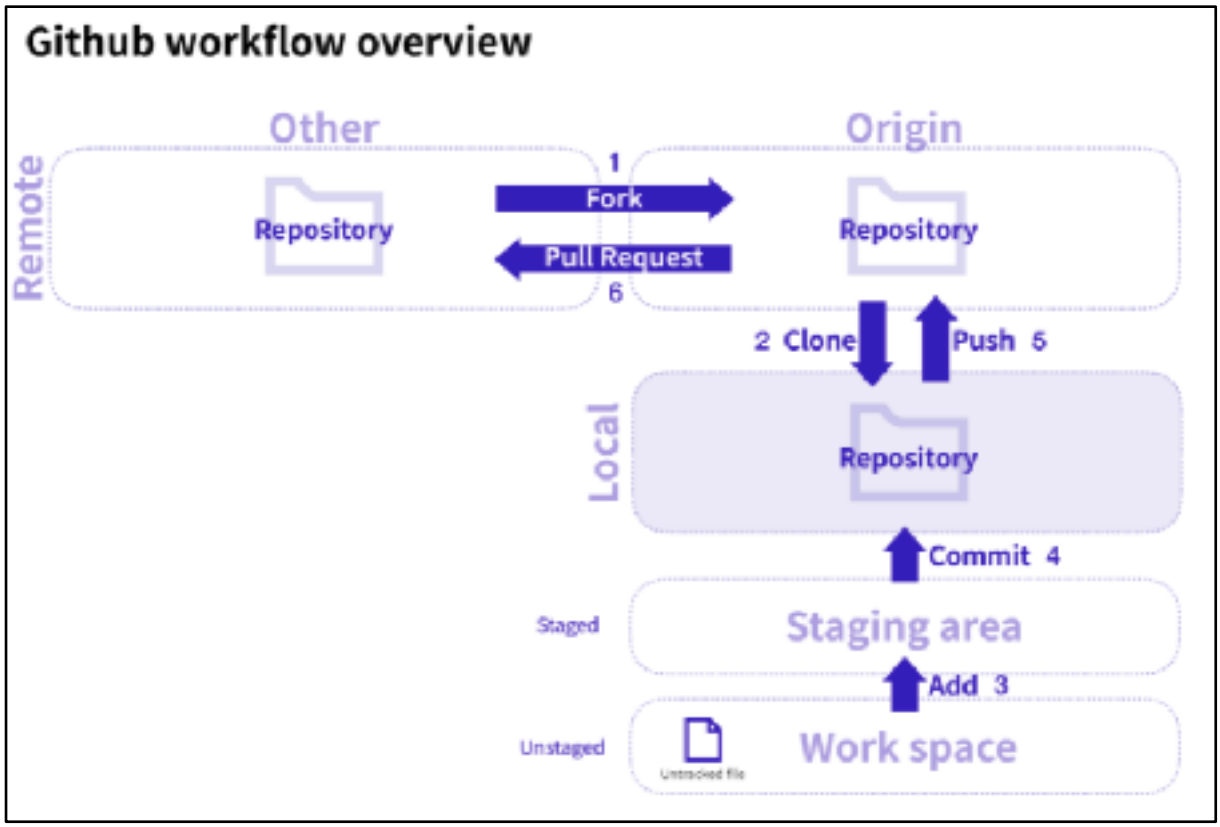
git commit 다음에 -m 옵션을 사용합니다. git commit 명령어는 현재 스테이징 영역에 있는 모든 변경사항을 하나의 커밋으로 만드는 명령어입니다. -m 옵션은 커밋의 변경사항을 요약해서 메모할 수 있는 옵션입니다.

git commit 사용법

앞에서 소개한대로 git commit의 가장 기본이 되는 사용법은 git commit -m'<MESSAGE>' 형식입니다. 하지만 git commit은 옵션 없이도 사용할 수 있습니다.

git commit

git 과정 정리



1. 로컬 Git 리포지토리 만들기

1) 코드를 저장할 디렉토리를 만들고 해당 디렉토리에 로컬 Git repository를 생성합니다.

   (git init)

2) 코드를 작성하고 저장하는 공간, 작업 공간(work space)의 파일 및 디렉토리를 git의 관리 하에 있는 상태로 올려줄 수 있습니다.

   (git add)이 영역이 staging area입니다.

3) staging area의 파일은 commit이 가능합니다.
   commit으로 local Git repository에 내 코드를 기록할 수 있습니다.

   (git commit)

1. git init

작업 공간에 local Git repository를 생성합니다.
일반적인 폴더에 Git repository를 추가하면 개발자는 Git을 이용하여 자신의 프로그램 버전 관리를 할 수 있습니다.

2. Staging area

Commit 하기 전에 내용을 기록하는 장소 입니다. git add <파일명> 명령어로 원하는 파일 혹은 디렉토리를 staging area로 옮길 수 있습니다. 즉, 내 Local의 Untracked files를 Git의 관리 하인 Staging area로 추가할 수 있습니다. git status로 staging area의 내용을 확인할 수 있습니다. git add 명령어 이후 git status 명령어를 실행하면 work space에 있는 untracked 파일, staging area에 있는 tracked 파일 목록을 확인할 수 있습니다.

3. git commit

staging area에 있는 코드 내역을 그대로 스냅샷을 찍어서 기록하는 행위입니다. 즉, 수정 작업이 끝났을 때 변경 사항을 저장합니다. 저장 후에는 staging area는 비워집니다.

---

1. 원격 Git 리포지토리

- git repository란 파일이나 폴더를 저장해두는 곳을 말하는데
- local repository는 내 컴퓨터의 저장소를 의미하고,
- remote repository는 원격 온라인 서버 상의 저장소를 의미합니다.

원격 git 리포지토리를 다루는 방법은 다음과 같습니다.

1) Github에서 원격 리포지토리를 생성합니다.
2) 로컬 리포지토리에 원격 리포지토리 git url을 등록합니다.(git remote add)
3) 로컬 Git repository에 기록한 내역을 원격 Git repository에 push합니다.(git push)

2. git remote

원격 리포지토리를 다루기 위한 git 명령어입니다.
git remote add 는 로컬 리포지토리에 원격 리포지토리 주소를 등록하는 명령어입니다.

git remote add <name><URL>

- <name> : 앞으로 로컬 리포지토리에서 원격 리포지토리 주소를 대신할 이름. 대부분 origin 으로 사용합니다.
- <URL> : 원격 리포지토리 주소(ex: git@github.com:StarryPro/my-first-github-repository.git)

3. git push

로컬 리포지토리에서 기록한 내역을 원격 리포지토리로 옮기는 작업입니다.
git push는 새롭게 생성한 원격 리포지토리에 기존 커밋 기록을 옮기거나, 기존 원격 리포지토리에서 일부 변경된 내용을 옮기는데 사용할 수 있습니다.

- <remote> : 원격 리포지토리의 이름. origin
- <branch> : 브랜치의 이름을 입력합니다. 기본적으로 main이나 master으로 등록되어있습니다.

로컬 리포지토리의 기록을 원격 리포지토리 origin의 main브랜치(master 브랜치)로 Push하려면git push origin main 또는 master를 입력합니다.
git push <remote><branch>

4. git pull

원격 리포지토리의 기록을 로컬 리포지토리로 옮기는 작업입니다.
git pull은 새롭게 생성한 로컬 리포지토리에 기존 커밋 기록을 옮기거나, 원격 리포지토리와 로컬 리포지토리에서 일부 변경된 내용을 옮기는데 사용할 수 있습니다.

git pull <remote><branch>

Github 명령어 모음 및 개념

1) git init
2) git remote add origin 깃주소
3) git add .
4) git commit -m "message"
5) git push origin master

처음에 commit 하여 push한 후에 파일 및 폴더가 추가 된 경우, 보통은 그냥 commit 하고 Push 하면 업데이트되어 깃허브에 적용 될 것이다. 하지만 파일 및 폴더가 추가된 경우라면 pull 한 다음 병합하여 다시 commit 하고 Push 해줘야한다.

1) git clone {원격 저장소 주소}
   - github 원격 저장소 주소와 연결
2) git push origin HEAD
   - 원격으로 내용을 업로드 하는 것을 PUSH 라고 한다

3) git fetch
   - 협업하다 보면 다른 사용자가 먼저 PUSH 했을 수도 있다. 이럴 땐 저장소의 최신 버전을 가져와야 한다. fetch를 사용하자

4) git pull origin master
   - 최신 업데이트 된 내용을 가져오고 병합하려면 PULL을 이용한다

5) git add .

6) git commit -m "finish"
   - finish라는 메세지와 함께 commit 전송한다

7) git push --set-upstream origin master
   - 새로운 set으로 origin master에 업로드

8) git push
   - PUSH 업로드가 완료된다

== End ==