**2025 로빛 인턴 Linux&Git 보고서**

2025407012/로봇학부/송연우

목차

1. Linux

1) 리눅스의 특징

2) 리눅스의 전체적 구조

3) 리눅스 터미널 사용법

4) 리눅스 파일 시스템 구조

2. git

1) git 영역

2) git 파일의 상태

3) git 명령어

**1. Linux**

리눅스는 운영 체제(OS)중 하나로, 유료 소프트웨어인 윈도우와 다르게 무료로 제공되며 오픈 소스라는 특징이 있습니다. 윈도우가 GUI(Graphic User Interface)를 중심으로 사용자에게 사용 환경을 제공하는 반면 리눅스는 CUI(Command Line Interface)와 GUI모두 사용하며 이미 완성되어 제공되는 환경인 윈도우와 다르게 사용자가 조작하고 새로운 체제를 만들어 배포할 수 있는 자유도를 제공합니다. 오픈 소스 운영 체제이기 때문에 다른 개발자들의 소스 코드의 열람과 수정이 자유로워 개발 환경에서 많이 쓰입니다. 리눅스는 또한 다양한 배포판이 존재합니다. 대표적으로 Ubuntu, CentOS, Debian, Fedora 등이 있으며 각자가 고유한 특징과 시스템을 지닙니다. 개발자는 다양한 배포판 중 자신이 쓰기에 알맞은 버전을 선택해 사용할 수 있습니다.

**1)리눅스의 특징**

- 오픈 소스: 리눅스는 오픈 소스 운영 체제로 소스 코드가 공개되어 누구나 코드를 확인, 수정할 수 있습니다.

- 멀티 유저, 멀티태스킹: 리눅스는 다중 사용자 및 다중 작업 환경을 지원합니다. 여러 사용자가 동시에 컴퓨터를 사용할 수 있고, 하나의 컴퓨터에서 여러 작업을 동시에 처리할 수 있습니다.

- 안정성: 리눅스는 운영 체제의 안정성이 높고 대규모 서버 환경에서 안정적으로 운영되어 왔습니다.

- 보안성: 커뮤니티와 개발자들은 보안 업데이트를 제공해 시스템을 보호하며, 리눅스 자체에서 보안 기능을 제공합니다.

- 커뮤니티/지원: 리눅스는 활발한 커뮤니티가 있고 사용자들 간 지식을 자유롭게 공유할 수 있습니다.

-유연성: 다양한 하드웨어, 플랫폼에서 사용 가능합니다.

- 무료 사용: 유료 소프트웨어 체제와 다르게 무료로 사용 가능하며 라이선스 비용/사용료가 없어 자유롭게 사용할 수 있습니다.

**2)리눅스의 전체적 구조:**

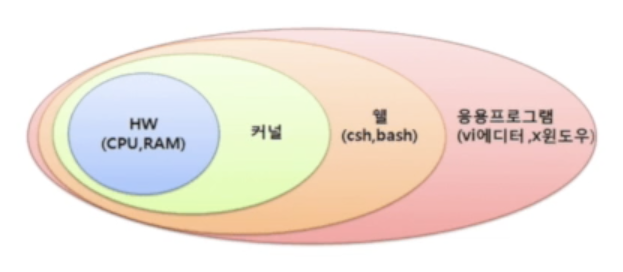
- 하드웨어: 컴퓨터의 물리적 부품을 가리킵니다.

- 커널: 소프트웨어의 핵심 부분으로 하드웨어를 운영/관리합니다. 컴퓨터의 기본적 장치를 관리하고 제어합니다.

- 쉘: 사용자와 운영체제 사이의 인터페이스를 제공하는 특수 프로그램입니다. 명령어를 입력받아 수행하는 해석기입니다.

- 응용 프로그램: 운영 체제에서 실행되는 모든 소프트웨어를 뜻합니다.

리눅스의 전체적인 구조를 나타낸 그림은 다음과 같습니다.



**3) 리눅스 터미널 사용법**

리눅스 터미널에서 사용되는 명령어를 설명합니다.

**ls**: 디렉터리 내 파일 목록을 확인할 수 있습니다.

+ -l: 자세한 파일 정보까지 표시합니다.

-a: 숨겨진 파일/디렉터리까지 포함해 표시합니다.

-lh: 파일 크기를 보기 쉬운 형식으로 표시합니다.

-r: 파일의 정렬 순서를 거꾸로 표시합니다.

-t: 파일을 작성 시간 순으로 표시합니다.

-lha: 이렇게 여러 옵션을 함께 사용할 수 있습니다.

**cd**: 디렉터리로 이동할 수 있습니다.

+ cd 단독/ cd ~으로 쓰면 home 디렉터리로 이동하며, cd ..는 상위 디렉터리고, cd -는 이전 디렉터리로 이동합니다. ‘cd 디렉터리명’ 을 쓰면 해당 디렉터리로 이동합니다.

**pwd**: 현재 작업 중인 디렉터리를 확인할 수 있습니다.

**mkdir**: 새 디렉터리를 생성할 수 있습니다.

+ -p를 붙이면 하위 디렉터리를 포함하여 생성할 수 있습니다.

**touch:** 빈 파일을 생성하거나 파일의 시간을 최근으로 업데이트할 수 있습니다.

**rm**: 파일 또는 디렉터리를 삭제할 수 있습니다.

+ -r, -f등 추가 명령어를 붙여 하위 디렉터리를 포함해 삭제하거나, 강제로 삭제할 수 있습니다.

**cp**: 파일을 복사합니다.

+ ‘cp 복사될 파일 복사할 대상 위치’ 순으로 씁니다.

**mv**: 파일을 이동하거나 이름을 변경할 수 있습니다.

+ ‘mv 파일명 디렉터리명’ 순으로 씁니다.

**paste**: 파일을 열 단위로 나란히 붙입니다.

+ ‘paste 파일명1 파일명2 > 새로운파일명’ 순으로 씁니다.

**vi:** 텍스트를 편집합니다.

+ ‘vi 파일명’ 으로 파일을 열고, 수정한 뒤 esc키를 누르고 wq입력 후 엔터를 누르면 저장됩니다.

**wq**: vi에서 저장 후 종료합니다.

+:wq 를 입력하면 파일 저장 후 vi에디터를 종료합니다.

**q!**: vi에서 저장하지 않고 종료합니다.

+ :q! 를 입력하면 파일을 저장하지 않고 vi에디터를 종료합니다.

**nano**: 간단한 텍스트 편집기입니다.

**cat**: 파일 내용을 출력합니다. ‘”cat> 파일명.txt”로 실행할 시 파일을 생성하고 편집할 수 있습니다.

**echo**: 문자열을 출력합니다.

+ ‘echo “문자열”’ 순으로 문자열을 출력합니다.

**file**: 파일 유형을 확인합니다.

**more**: 파일 내용을 페이지 단위로 출력합니다

**less**: 파일 내용을 페이지 단위로 출력합니다.

+less에서는 스크롤 기능을 제공해 사용자가 파일의 내용을 탐색할 수 있게 합니다.

**grep**: 파일 내에서 특정 문자열을 검색합니다.

+’grep “문자열”’ 파일명 순으로 사용할 수 있습니다.

**wc**: 파일 내 단어, 줄 문자 수를 셉니다.

**sort**: 파일 내용을 정렬합니다.

**head**: 파일의 상위내용 일부를 출력합니다.(기본적으로 10줄 출력)

**tail**: 파일의 하위 내용 일부를 출력합니다.(기본적으로 10줄 출력)

**chmod**: 파일의 권한을 변경합니다.

**chown**: 파일의 소유자를 변경합니다.

**chgrp**: 파일의 그룹을 변경합니다.

**find**: 파일 시스템에서 파일을 검색합니다.

**du**: 디스크 사용량을 확인합니다.

**df**: 디스크의 여유 공간을 확인합니다.

**mount**: 파일 시스템의 마운트를 확인합니다.

**umount**: 파일 시스템 언마운트

**passwd**: 사용자 암호를 변경합니다.

**useradd**: 새로운 사용자를 추가합니다.

**userdel**: 사용자를 삭제합니다.

**groupadd**: 새 그룹을 추가합니다.

**groupdel**: 그룹을 삭제합니다.

**sed**: 텍스트를 편집하고 치환합니다.

**awk**: 텍스트를 처리하고 보고서를 생성합니다.

**cut**: 파일에서 필드를 추출합니다.

**clear**: 터미널 화면을 지웁니다.

**4) 리눅스 파일 시스템 구조**

주요 디렉터리

-**/**(루트 디렉터리): 파일 시스템의 최상위 디렉터리입니다.

-**/bin**: 기본적 실행 파일이 저장되며, ls, cp, mv, rm등 기본 명령어를 사용합니다.

-**/etc**: 시스템 전체의 설정과 관련한 파일이 저장됩니다.

-**/home**: 일반 사용자들의 홈 디렉터리입니다.

-**/var**: 변화하는 데이터를 저장하는 디렉터리입니다.

-**/usr**: 사용자가 설치한 프로그램/데이터를 저장합니다.

-**/tmp**: 임시 파일이 저장되어 재부팅 시에는 내용이 삭제될 수 있습니다.

-**/proc**: 가상 파일 시스템으로 실행 중인 프로세스/시스템 정보를 파일 형태로 제공합니다.

**2. Git**

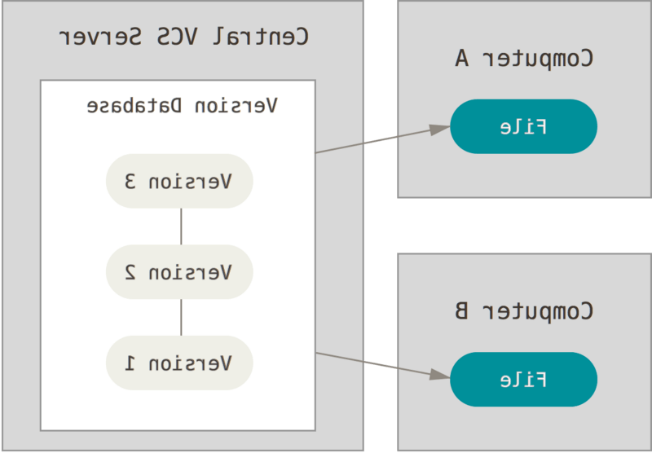
Git은 컴퓨터 파일의 변경사항을 추적/여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 스냅샷 스트림 기반의 분산 버전 관리 시스템 또는 그러한 명령어를 가리킵니다. 버전 관리 시스템은 특정 문서/소프트웨어 등 버전을 안전하게 관리하는 시스템으로, 3가지 형태(로컬형, 중앙형, 분산형)이 있습니다.

로컬형은 사용자의 컴퓨터에 관리할 버전 내용을 저장하는 형태로, 사용이 간단하지만 다른 사람들과의 공유가 어렵고 사용자 컴퓨터에 문제가 발생했을 때 복구가 어렵습니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

중앙형은 사용자 컴퓨터가 아니라 여러 사용자의 버전 관리 내용을 중앙 서버에서 관리하는 형태입니다. 하나의 중앙 서버에서 여러 팀원을 관리할 수 있다는 편리성이 장점이지만 팀원이 사용할 수 있는 버전은 가장 최신 버전 뿐이며 중앙 서버에 문제가 생길 경우 팀원들의 작업이 불가능해집니다.



분산형은 사용자의 컴퓨터에서는 로컬형처럼 개발이 가능하고, 이를 중앙형처럼 중앙 서버로 보내거나 중앙 서버의 진행상황을 가져와 사용자의 컴퓨터를 갱신시킬 수도 있습니다. Git은 이 중에서도 분산형 버전 관리 시스템에 속합니다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

Git의 장점은 협업에 뛰어난 특성을 지닌다는 점인데, 대표적인 장점으로는 아래 세 가지가 있습니다.

-동시성: 다수의 개발자가 동일한 기능을 동시에 개발할 수 있습니다.

-가역성: 기능을 작성하다 오류가 발생하거나 하는 경우에 이전 코드로 되돌릴 수 있습니다.

-기록성: 이전 코드의 내용이 저장되어 오류 발생 시 추적이 용이합니다.

이 외에 git과 github의 차이점은, git은 분산 버전 관리 시스템이고 github는 git을 사용해 만든 프로젝트들을 공유하는 저장소라는 것입니다.

**1) git 영역**

(1) 작업 디렉터리: 작업을 하는 프로젝트 디렉토리를 뜻합니다.

(2) 스테이징 영역: git add를 한 파일들이 존재하는 영역입니다.

(3) 저장소: 작업 디렉터리에서의 변경 이력이 저장되어 있는 영역으로, 커밋들이 저장되는 영역입니다.

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

사용자가 작업 영역에서 A.txt 파일과 B.txt 파일을 작성하고, git add A.txt와 git add B.txt를 실행해서 스테이징 영역에 올라가게 합니다. 이후 git commit -m "1st commit"을 실행해서 스테이징 영역에 있던 파일들이 스냅샷처럼 하나의 버전으로 저장소에 저장되게 됩니다. 여기서 스테이징 영역에 올린 파일만이 저장소에 반영됩니다.

**2) git 파일의 상태**

git으로 관리되는 파일은 4가지 상태로 분류됩니다.

1. Untracked: 파일의 변동사항이 추적되고 있지 않는 상태를 나타내며, 해당 파일에 git add를 한 번도 하지 않은 상태를 뜻합니다.
2. Tracked: Untracked의 반대 개념으로 3가지 상태로 다시 갈라집니다.

-Staged: 스테이징 영역에 올라온 상태입니다.

-Unmodified: 현재 파일의 내용이 최신 커밋의 모습과 비교했을 때 바뀐 것이 없는 상태입니다. 주로 커밋을 하고 난 직후 작업 영역 안의 파일들의 상태입니다.

-Modified: Unmodified와 반대 상태로 커밋을 한 뒤 변경된 사항이 있을 때의 파일 상태입니다.

**3) git 명령어 종류**

**Config:** git의 환경 설정 시에 쓰이는 명령어입니다. 운영체제 단위의 환경 설정을 할 때 사용합니다.

git config –global user.name”Your name” 으로 전역 사용자명을 설정할 수 있므며,

git config –global user.name”Your email address”로 이메일을 설정할 수 있습니다.

git config user.name”Your ~~” 으로 쓰면 저장소별 사용자명/이메일 설정이 가능합니다. 이때는 해당 디렉토리 내로 이동한 상태에서 실행해야 합니다.

**Init:** 프로젝트 버전 관리를 시작하는 기능입니다.

git init 명령어로 생성할 수 있습니다.

**Track:** 버전을 관리할 파일을 지정하는 것을 뜻하며, 스테이징은 추적할 파일을 준비하는 것을 말합니다.

**Commit:** 현재 작업 내용을 하나의 버전으로 기록합니다. git add <>, git add \*명령어가 있습니다.

**Log:** 생성한 버전의 내용, 내역, 변경 사항 등을 확인합니다.

git log로 모든 이력을 확인할 수 있으며, -p를 뒤에 붙이면 변경 사항을 보여주는 패치와 함께 표시됩니다. -1과 같이 -숫자를 붙이면 원하는 개수의 항목만 보이도록 로그의 개수를 제한할 수 있습니다.

git log –since=”시간”으로는 해당 시간 동안의 커밋 로그만 볼 수 있고, git log –before=”시간”으로는 해당 시간 전까지의 커밋 로그를 볼 수 있습니다.

**Branch:** 버전 관리에서 각 분기점을 뜻합니다.

git branch로는 지역 브랜치의 목록을 열람할 수 있고, git branch -r, -a등 명령어를 더 붙여 원격 브랜치 목록을 보거나 모든 브랜치 목록을 확인할 수 있습니다.

git branch <새로운 브랜치>,로는 현재 브랜치에서 새로운 브랜치를 생성할 수 있고

git checkout <브랜치>에서는 다른 브랜치를 체크아웃할 수 있습니다. 마찬가지로 git checkout git checkout master 등 명령어가 있습니다. -b <새로운 브랜치>로 -b를 덧붙여 현재 브랜치에서 새로운 브랜치를 생성하고 체크아웃할 수 있습니다.

git branch <새로운 브랜치> <브랜치를 생성할 위치>로 현재 있는 지점 외에서 브랜치를 생성하거나

git branch -f <기존 브랜치> [<브랜치를 생성할 위치>]기존 브랜치를 덮어쓰기할 수 있습니다.

git checkout -m <기존 브랜치> <새로운 브랜치>로 브랜치를 옮기거나 브랜치명을 변경할 수 있습니다. 여기서 새로운 브랜치가 존재하지 않는 경우에는 m을 M으로 바꾸어 줍니다.

git cherry-pick <커밋명> 명령어로는 선택해 합치기를 할 수 있고 뒤에 -n을 붙이면 커밋하지 않습니다.

만약 브랜치를 삭제하고 싶으면 git branch -d <삭제할 브랜치>를 사용하고 git branch -D <삭제할 브랜치>로 삭제할 브랜치가 현재 브랜치에 합쳐졌을 경우에도 대상 브랜치를 삭제할 수 있습니다.

**Push:** 로컬 저장소의 버전 내역을 원격 저장소로 업로드합니다. git push origin master, git push <remote> <>, git push -u<remote> <>등 명령어가 있습니다.

**Rollback:** 작업 중인 변경 사항을 모두 취소하고 제거합니다.

**Reset:** 특정 버전으로 이동한 다음, 그 이후의 버전 내역을 제거합니다.

**Revert:** 특정 버전을 취소하고, 취소한 새로운 버전을 생성합니다.

**Stash:** 변경사항을 버전으로는 생성하지 않고 별도로 저장합니다.

**Rebase:** 현재 브랜치 내역을 대상 브랜치의 최신 버전 다음으로 배치시킵니다.