Linux & git 보고서 작성

정보융합학부 2025404008 김다은

Linux와 git은 현대 정보기술 분야에서 중요한 기술 중 하나이다. 각각 Linux 운영체제와 Git 버전 관리 시스템은 소프트웨어 개발과 시스템 관리의 핵심 도구로 자리잡았다. Linux는 서버, 슈퍼컴퓨터, 모바일 기기부터 임베디드 시스템까지 광범위하게 사용되는 오픈소스 운영체제이며, Git은 전 세계 개발자들이 협업을 위해 사용하는 분산 버전 관리 시스템이다.

1990년대 초반부터 발전해온 linux는 오픈소스 철학을 기반으로 전 세계 사람들이 자유롭게 수정하고 사용할 수 있는 운영체제이다. 서버, 데스크톱, 모바일, 임베디드 시스템 등 다양한 분야에서 영향력을 확대하면서, it 인프라의 중추적인 역할을 하고있다.

Git은 위에서 말 했듯이 분산 버전 관리 시스템으로서, 개발자들이 소프트웨어 개발 협업과 코드관리를 효율적으로 할 수 있게 도와주는 혁신적인 도구 중 하나이다. 2005년 리누스 토르발스 로부터 개발된 이후에 git은 빠르고 안정적인 브랜칭과 병합 기능, 그리고 분산 저장 구조를 통해 엄청 큰 대규모 프로젝트뿐 아니라 소규모 개인 프로젝트에서도 필수적으로 사용되는 시스템으로 발전해 나갔다. Git이 제공하는 다양한 기능들은 개발 과정에서의 코드 변경 추적, 공동 작업환경 개선, 그리고 배포 자동화, 포트폴리오로도 까지 가능하게 한다.

리눅스의 정의

리눅스 (Linux)는 1991년 리누스 토발즈가 개발한 오픈소스 기반의 유닉스 (unix) 계열 운영체제이다. 운영체제의 핵심인 커널(kernel)을 중심으로 다양한 유틸리티와 응용 프로그램으로 구성되어 있고, 사용자는 커널과 시스템 소프트웨어의 소스코드를 자유롭게 열람, 수정, 재배포 할 수 있는 자유 소프트웨어이다. 이런 오픈소스 특성 덕분에 전 세계 개발자와 기업이 공동으로 발전시키는 그런 형태를 형성 하고 있다.

리눅스는 멀티유저 및 멀티태스킹을 지원하여 여러 사용자가 동시에 여러 작업을 수행할 수 있으며, 강력한 네트워크 기능과 안정성을 보여준다. 또한 다양한 하드웨어 플랫폼에서 이식성이 뛰어나고, 서버, 데스크톱, 모바일(Android 기반), 임베디드 시스템 등 폭넓은 분야에서 활용이 되고 있다.

또한 리눅스는 posix(portable operating system interface) 표준을 준수하여 유닉스 환경과의 호환성을 유지하며, 시스템 자원 관리, 프로세스 및 메모리 관리, 파일, 시스템 관리 등 운영체제가 가져야 할 핵심 기능을 탁월하게 수행할 수 있다. 또한 커널과 셸(shell), 파일 시스템, 응용 프로그램 간의 계층적인 구조를 가지고 있고, 명령어 해석기 역할을 하는 셸을 통해 사용자가 시스템과 상호작용을 한다고 한다.

리눅스의 주용 장점으로는 오픈소스 기반으로 빠른 버그 수정과 업데이트가 가능하다는

점, 비용이 저렴하거나 무료로 제공된다는 점, 그리고 강력한 보안성과 안정성으로 기업과 개인 사용자 모두에게 각광받는 운영체제이다. 수많은 배포판이 존재하여 사용자 목적에 맞는 환경을 구축하고 제공한다.

리눅스 기본 개념

운영체제 개요

운영체제는 컴퓨터 하드웨어를 관리하고 응용 소프트웨어가 원활히 실행되도록 지원하는 시스템 소프트웨어이다. 컴퓨터 부팅 시 최초로 실행되어 시스템 자원을 초기화하고, 사용자와 하드웨어간의 인터페이스 역할을 수행한다. 운영체제는 프로세스 관리, 메모리 관리, 차일 시스템 관리, 입출력 장치 제어, 보안 및 네트워크 기능 등을 포함하며 리눅스는 이 운영체제의 한 형태로, 오픈소스 구조를 기반으로 자유롭게 수정 또는 배포가 가능하다. 안정적인 자원관리와 확장된 네트워크 기능이 있다.

리눅스와 다른 운영체제 비교

리눅스는 위에서 말했듯이 오픈소스 자유 소프트웨어로, 커널과 주요 구성 요소가 공개되어 있어 사용자나 개발자가 직접 수정 및 배포할 수 있는 반면에, 우리가 흔히 쓰는 windows와 macOS는 상용 운영체제로 소스코드가 제공되지 않았다. 그래서 windows는 높은 사용자 친화성과 광범위한 소프트웨어 호환성이 강점이며, macOs는 애플 하드웨어 최적화와 사용자 경험을 중시하는 운영체제이다. 반면에 리눅스는 서버, 개발 환경, 클라우드 및 임베디드 시스템에 강점을 보여주고, 커스터마이징과 보안성 면에서 뛰어난 유연성을 제공한다

리눅스의 커널 개념과 구조

리눅스 커널은 운영체제의 핵심 부분으로 하드웨어와 소프트웨어 간의 중재자 역할을 한다. 프로세스 스케줄링, 메모리 관리, 파일 시스템 관리, 네트워크 기능, 장치 드라이버 제공 등을 담당하고 모놀리식 구조를 가지며, 다양한 모듈(확장 기능)을 로드해 유연성을 제공한다. 커널 모드는 프로세스나 하드웨어 접근 제어를 수행하며, 사용자 공간에서는 셸이나 응용 프로그램이 실행된다, 커널은 시스템 자원을 효율적으로 관리하고 보안을 유지하는데 중추적인 역할을 한다.

리눅스 배포판 종류

리눅스 배포판은 커널과 다양한 소프트웨어 패키지를 포함하여 특정 용도에 따라 알맞게 조합한 운영체제의 세트이다. 대표적인 배포판으로는 데스크톱과 서버용으로 널리 사

용되며, 사용이 쉬워 초보자에게 적합한 우분투, 안정성과 자유 소프트웨어 철학을 중시하며, 커뮤니티 중심으로 개발하는 데비안, 기업용으로 안정성과 장기 지원을 제공하는 레드햇 엔터프라이즈 리눅스, 최신 기술을 빠르게 도입하는 배포판으로 개발 및 테스트 환경에 적합한 페도라, 레드햇 기반의 무료 서버용 배포판 중 하나 였으며, 최근에는 centos stream으로 전환된 센트 os등 이외에도 많은 배포판들이 특정 보안, 경량화, 교육 등 목적에 특화되어 있으며, 각 배포판은 패키지 관리 시스템, 기본 설치 소프트웨어, 사용자 지원 커뮤니티에 따라 차이가 있다.

리눅스 설치와 설정

하드웨어 관련 요구사항

리눅스는 일반 용량만 있으면 설치가 가능한 것이 아니라 설치를 위해서는 해당 배포판이 요구하는 최소 하드웨어 사양을 충족해여 한다. 일반적으로 CPU는 최소 1GHz 이상의 프로세서, 메모리는 최소 1~2GB RAM 이상을 권장한다. 특히 대규모 클라우드, 가상화 환경에선 RAM 요구량이 크게 증가할 수 있다. 그리고 하드디스크는 설치 및 운영에 최소 10GB 이상의 여유공간이 필요하고, 서버환경이나 응용에 따라서 50GB 이상도 권장될 수 있다. 그래서 ssd 사용 시 성능 향상에 유리하다. 또한 gui 환경을 사용하려면 기본적인 그래픽 출력 장치가 있어야 하고 설치 및 운영시 네트워크 연결을 위한 호환성 있는 NIC(네트워크 인터페이스 카드)가 필요하다 하드웨어의 호환성은 리눅스 커널 모듈과 드라이버가 지웒나느 범위 내에서 결정되며, 최신 커널을 사용하면 다양한 하드웨어를 원활이 지원받을 수 있다. 일부 특수 하드웨어는 별도 드라이버 설치가 필요할 수 있다

하나를 예시로 이번에 다운받은 우분투로 설치 방법을 설명하자면, ubuntu 공식 웹사이트에서 iso 이미지를 다운로드 받고 Rufus(윈도우용), Etcher(크로스 플랫폼) 등의 툴로 USB(16GB이상) 부팅 디스크를 만든다. USB 또는 DVD부팅 후 파티션 설정, 네트워크, 사용자 계정 생성 단계 등을 거쳐서 데스크톱용인지 서버용인지 설티 옵션을 선택하여 설치 할 수 있다. 그래서 각 배포판 설치 과정은 유사하지만, 패키지 관리자와, 기본 제공 소프트웨어, 업데이트 정책 등 그리고 사용자의 pc에 따라 다 다르니 공식 문서를 찾아보고 참고해가며 설치하는 것이 좋다

초기 설정 및 사용자 관리

설치 후 처음으로 해야 하는 것은 ip주소, DNS, 게이트웨이 등을 확인 및 설정하고, 시스템 시간대와 언어 설정을 하고 apt update && apt upgrade등의 명령어로 최신 보안 패치를 적용해야 한다.

또한 사용자 관리로 기본적으로 root 계정이 최고 관리자 권한을 가지며 보안상 직접 로그인 제한하는 것이 권장된다. Useradd 명령어로 새 사용자를 추가하고, passwd로 비밀번호를 설정(비밀번호와 사용자이름을 설정할 때는 최대한 짧은게 좋다. 매번 작성을 해야 하기에 최대한 짧게 하는 것이 좋다.)하고 그룹 생성과 권한 설정을 통해 리소스 접근 제어를 수행하고, sudo 명령어를 통해 일시적 관리자 권한 위임을 해야 한다. 그래서 처음에 설치하고 이러한 설정을 잘해야 시스템 보안 강화와 원할한 운영을 위해 잘 해야 하 한다.

파일 시스템과 디렉터리 구조

리눅스의 파일 시스템은 유닉스 철학에 따른 트리 구조로 되어 있다. 모든 파일과 디렉터리는 루트(/) 아래에 위치한다. 주요 디렉터리는

/bin : 실행 파일(명령어 저장)

/sbin : 시스템 관리자용 명령어

/etc : 주요 설정 파일

/home : 사용자별 홈 디렉터리

/var : 로그 파일, 데이터베이스, 캐시 등 가변데이터

/usr : 일반 사용자 프로그램 및 라이브러리

/dev : 장치 파일(하드웨어 인터페이스)

/tmp : 임시 파일 저장

리눅스는 Ext4, XFS, Btrfs, ReiserFS 등 다양한 파일 시스템을 지원하고, 스왑 파티션을 통해 물리 메모리가 부족할 때 디스크 공간을 추가 메모리처럼 활용한다. 디렉터리와 파일은 유저와 권한이 명확히 구분되어 있어 보안에 중요한 역할을 한다.

쉘과 터미널 기본 사용법

쉘은 텍스트 기반 명령어 인터페이스로 사용자가 운영체제와 상호작용하는 방식이다. 가장 대표적인 쉘을 Bash이다

명령어 실행 : ls(목록 보기), cd(디렉터리 이동), pwd(현재 경로 출력)

파일 조직 : cp(복사), mv(이동), rm(삭제), touch(빈 파일 생성)

텍스트 편집기 : vim, nano 를 이용한 파일 편집

프로세스 관리 : ps, top, kill 프로세스 조회 및 종료

터미널에서 쉘을 실행하는 콘솔창으로 gui 환경에서도 쉽게 접근이 가능하고 빠른 시스템을 관리 할 수 있다.

라눅스 명령어 및 셸 스크립팅

기본 명령어 (파일, 프로세스, 네트워크 등)

- 파일 및 디렉터리 관리
    - ls: 디렉터리 내용 목록 표시 (ls -l, ls -a)
    - cd: 디렉터리 이동 (cd /home)
    - pwd: 현재 작업 디렉터리 경로 출력
    - cp: 파일 복사 (cp file1.txt file2.txt)
    - mv: 파일 이동 또는 이름 변경
    - rm: 파일 삭제 (rm -r directory/)
    - mkdir: 새 디렉터리 생성
    - touch: 빈 파일 생성 또는 수정시간 변경
- 프로세스 관리
    - ps: 현재 실행 중인 프로세스 정보 조회 (ps aux)
    - top: 실시간 시스템 및 프로세스 상태 모니터링
    - kill: 프로세스 종료 (kill PID)
    - htop: 보다 편리한 프로세스 관리 도구 (별도 설치 필요)
- 네트워크 관리
    - ifconfig 또는 ip addr: 네트워크 인터페이스 설정 및 상태 확인
    - ping: 네트워크 연결 체크
    - netstat 또는 ss: 네트워크 연결 상태 및 포트 확인
    - scp: 원격 시스템에 파일 복사
    - ssh: 원격 서버 접속

텍스트 처리 명령어 (grep, awk, sed 등)

- grep: 파일 내용 가운데 특정 문자열 검색 (예: grep "error" /var/log/syslog)

- awk: 텍스트 데이터 추출, 필터링, 형식 변경에 강력한 도구 (예: 표 형식 데이터 처리)
- sed: 스트림 편집기로 텍스트 치환, 삭제, 삽입 가능 (예: sed 's/old/new/g' filename)
- cut: 파일에서 특정 필드 또는 문자 추출
- sort: 텍스트 정렬
- uniq: 중복 제거, 주로 sort와 조합해서 사용
- wc: 단어, 문자, 줄 수 계산
- tr: 문자 변환 및 삭제

사용자 및 권한 관리 명령어

- useradd / adduser: 새 사용자 계정 생성
- passwd: 사용자 비밀번호 변경
- usermod: 사용자 계정 정보 수정
- userdel: 사용자 계정 삭제
- groups: 사용자가 속한 그룹 확인
- groupadd: 새 그룹 생성
- chmod: 파일 또는 디렉터리 권한 변경 (숫자 또는 기호 방식)
- chown: 파일 소유자 변경
- chgrp: 파일 그룹 변경
- sudo: 일시적으로 관리자 권한 획득

셸 스크립트 기초 및 실습

- 쉘 스크립트: 여러 명령어를 순차적으로 실행할 수 있는 스크립트 파일로, .sh 확장자를 주로 사용한다.
- 기본 구조: 맨 첫 줄에 실행할 쉘 경로 지정 (#!/bin/bash)
- 변수 선언과 사용: var=value, 사용 시 $var
- 조건문: if, else, elif 구문

- 반복문: for, while, until 구문
- 함수: 재사용 가능한 명령어 그룹
- 스크립트 실행 방법: 실행 권한 부여(chmod +x script.sh), 실행(./script.sh)

리눅스 시스템 관리

사용자 및 그룹 관리

리눅스 시스템에서 사용자 및 그룹 관리는 보안과 권한 분리를 위해 필수적이다.

사용자 계정 관리: useradd로 계정 생성, passwd로 비밀번호 설정, usermod로 계정 속성 변경, userdel로 계정 삭제

그룹 관리: 공통 권한 부여를 위해 그룹 생성(groupadd), 수정(groupmod), 삭제(groupdel)하며 사용자를 그룹에 추가(usermod -aG 그룹명 사용자)

권한 부여: 홈 디렉터리 위치와 기본 쉘 설정, 계정 활성화 여부 등을 관리하며, 로그인 허용 및 제한이 가능

/etc/passwd, /etc/group, /etc/shadow 파일을 통해 사용자 및 그룹 정보를 관리

프로세스 및 서비스 관리

프로세스 관리: ps, top, htop 명령어로 실행 중인 프로세스 상태를 확인하고, kill 명령어로 프로세스를 종료

서비스 관리: 시스템 서비스는 systemctl 명령으로 시작, 중지, 재시작, 상태 확인한다. 이전에는 service명령이나 /etc/init.d 스크립트를 사용

서비스 자동 시작 설정도 가능하며, 서비스 단위 파일을 통해 구성

패키지 관리 시스템 (apt, yum 등)

APT (Advanced Package Tool): 우분투, 데비안 계열에서 사용하며 apt update로 저장소 업데이트, apt install 패키지명으로 설치, apt upgrade로 업그레이드

YUM/DNF: 레드햇, 센트OS, 페도라 계열에서 사용한다. yum install, yum update, dnf는 차세대 도구로 빠르고 개선된 성능을 제공

패키지 관리자는 설치된 소프트웨어 관리뿐 아니라 의존성 해결, 소스 저장소 관리, 보안 패치 적용 등을 자동화

네트워크 설정 및 관리

네트워크 인터페이스 설정: ip 명령어 또는 ifconfig로 IP 주소, 네트워크 마스크, 게이트웨

이 설정.

네트워크 상태 확인: ping, netstat, ss 명령어로 연결 상태, 포트 개방 여부 점검.

DNS 설정은 /etc/resolv.conf 파일에서 관리.

방화벽 설정은 iptables, 차세대 방화벽 firewalld 등을 사용하며, 포트 개방 및 차단 정책을 설정

로그 파일 분석 및 시스템 모니터링

로그 파일은 대부분 /var/log에 위치하며, rsyslog 혹은 journald 시스템에서 관리된다.

주요 로그 파일:

/var/log/messages: 일반 시스템 메시지

/var/log/syslog: 시스템 전반 로그(데비안 계열)

/var/log/auth.log 또는 /var/log/secure: 인증 및 보안 관련 로그

/var/log/dmesg: 커널 및 부팅 로그

로그 분석 도구로 tail, less, grep 등이 사용되며, logrotate로 로그 파일 크기를 관리한다. 시스템 모니터링은 top, htop, vmstat, iostat 등으로 CPU, 메모리, 디스크, 네트워크 상태를 실시간 감시한다.

리눅스 개발환경

GCC(GNU Compiler Collection)는 C, C++ 등 여러 프로그래밍 언어를 지원하는 표준 컴파일러로, 리눅스 개발에 필수적이다. 우분투에서는 sudo apt install build-essential 명령어를 사용하여 gcc와 관련된 필수 개발 도구를 간편하게 설치할 수 있다. Make는 소스 코드 빌드 자동화 도구로, Makefile을 통해 소스 파일 간의 의존성을 관리하며, 변경된 부분만 효율적으로 컴파일하도록 지원한다. 리눅스에서 Make는 sudo apt install make 명령어로 별도 설치할 수 있다. Git은 분산형 버전 관리 시스템으로, 소스 코드의 변경 이력을 관리하며 팀 협업에 꼭 필요하다. Git도 우분투에서는 sudo apt install git으로 설치하고, GitHub, GitLab 같은 온라인 플랫폼과 연동해 사용한다. 이 개발 도구들은 주로 명령어 기반으로 작동하며, 효율적인 빌드, 코드 관리 및 팀 협업 환경을 구성하는 데 필수적이다.

또한 리눅스는 다양한 프로그래밍 언어를 지원하여 개발 환경 구축이 자유롭다. Python, Ruby, Java, Go, Node.js 등 여러 언어에 대한 컴파일러와 인터프리터를 설치할 수 있다. 개발 효율을 높이기 위해 Vim, Emacs, Visual Studio Code, Sublime Text 같은

텍스트 편집기와 통합 개발 환경(IDE)을 설치해 사용할 수 있다. 특히 쉘 스크립트는 Bash를 기반으로 강력한 자동화 스크립트를 작성할 수 있어 시스템 관리와 개발 작업에 자주 활용된다. 또한, apt, yum, pip 같은 패키지 매니저를 통해 필요한 라이브러리와 도구를 체계적으로 설치하고 관리할 수 있다. 이러한 환경은 다양한 언어 지원과 자동화 도구, 강력한 편집기 덕분에 리눅스에서의 프로그래밍과 스크립트 개발을 한층 효율적으로 만든다.

그리고 리눅스 개발 문화는 오픈소스 정신을 기반으로 하며, 전 세계 개발자들이 Git, 메일링 리스트, 이슈 트래킹 시스템 등을 통해 협업한다. 프로젝트 소스 코드 분석, 버그 보고, 기능 패치 제출, 코드 리뷰 과정 참여가 협업의 기본 방식이다. GitHub, GitLab, Bitbucket과 같은 플랫폼을 활용해 포크, 풀 리퀘스트, 자동 빌드 및 테스트를 수행하며, 실시간으로 소스 코드 변경 내용을 공유한다. 개발자는 커뮤니티 내에서 문서 작성, 번역, 디자인 개선, 테스트 코드 추가 등 다양한 방법으로도 기여할 수 있다. 또한 커뮤니티 밖에서도 개발자 회의, 컨퍼런스 참여 등을 통해 활발한 교류와 협력이 이루어진다. 이런 협업 체계 덕분에 오픈소스 프로젝트는 지속적으로 발전하며, 투명성과 품질을 유지할 수 있다.

리눅스 커널 개발은 공식 Git 저장소에서 커널 소스 코드를 클론하여 로컬에서 수정 작업을 수행한 뒤, 작업 내용을 패치 형식으로 메일링 리스트에 제출하는 방식으로 진행된다. 수정 내역은 커널 메인터이너의 리뷰와 승인을 거쳐 메인 소스 코드에 병합된다. 개발자는 git format-patch, git send-email 같은 도구를 활용하며, 패치를 작성할 때는 커밋 메시지 규칙과 코딩 표준을 반드시 준수해야 한다.

리눅스 커널 기여는 고도의 시스템 프로그래밍 지식을 요구하지만, 작은 버그 수정이나 문서 개선 같은 간단한 기여부터 시작해 점차 기여도를 높여가는 접근 방식이 효과적이다. 개발 환경에서는 GCC, Make, GDB, perf 등 디버깅 및 성능 분석 도구가 필수적으로 사용된다.

리눅스 커널 소스 코드를 클론 할 때 밑에 명령어를 사용한다:

git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git

이후 새 브랜치를 생성해 소스를 수정하고, 커밋 시 -s 옵션으로 서명을 넣는다:

git checkout -b feature/patch-name

git add modified_file.c

git commit -sm "feature: 설명"

커밋 후에는 패치를 생성한다:

git format-patch HEAD^

Git의 개념과 역사

Git은 2005년 리눅스 커널을 개발한 리누스 토발즈(Linus Torvalds)가 기존에 사용하던 BitKeeper라는 유료 분산 버전 관리 시스템의 라이선스 변경 및 사용 제한으로 인해, 자체적으로 개발한 분산형 버전 관리 시스템(DVCS)이다. 리누스는 리눅스 커널 개발 커뮤니티의 자유롭고 효율적인 협업을 위해 빠르고 비선형적인 작업이 가능하며 네트워크 연결이 없어도 로컬에서 모든 버전 관리가 가능한 도구가 필요했다. 이에 Git은 빠른 속도, 단순한 구조, 완전한 분산, 수천 개 브랜치를 동시 지원하는 설계를 목표로 탄생했다.

Git은 기존의 중앙집중식 버전 관리 시스템(CVS, SVN)과 달리, 모든 개발자가 각자의 로컬 저장소에서 완전한 버전 기록을 가지며, 네트워크 연결 없이도 작업할 수 있는 완전 분산 시스템이다. 이러한 특성으로 인해 병렬 개발과 복잡한 병합 작업이 용이하며, 대규모 프로젝트에 적합하다.

Git 설치 및 환경 설정

Git은 다양한 운영체제에서 설치할 수 있으며, 각 플랫폼별 설치 방법은 다 다른데 지금 리눅스에 설치를 할 예정이니 리눅스로 설치 방법을 알아보자면, 대부분의 배포판에서 패키지 관리자를 통해 쉽게 설치 가능하다. 예를 들어, Ubuntu 계열에서는 sudo apt install git 명령어를 사용하고, Red Hat 계열에서는 sudo yum install git 명령어를 사용하여 설치 할 수 있다.

초기 설정

Git 설치 후에는 커밋 기록에 사용되는 사용자 이름과 이메일을 반드시 설정해야 하며,

git config --global user.name "사용자 이름", git config --global user.email "사용자 이메일" 이렇게 명령어를 사용한다

또한 Git 사용 중 텍스트 편집기를 변경하고자 할 때는 git config --global core.editor "vim" 명령어로 기본 편집기를 변경 할 수 있다

설정된 내용을 확인하려면 git config --list 명령어를 사용한다.

SSH 키 생성과 Git 서버 인증

원격 저장소와 안전하게 통신하기 위해 SSH 키를 이용한 인증 방식을 주로 사용한다. SSH 키 생성은 ssh-keygen -t rsa -b 4096 -C "사용자 이메일 이 명령어로 생성한다"

생성된 공개 키(~/.ssh/id_rsa.pub)를 GitHub, GitLab 등의 원격 서비스 계정에 등록하면, SSH를 통한 비밀번호 입력 없이도 안전하게 접근할 수 있다. SSH 연결이 정상적으로

설정되었는지 확인하기 위해서는 ssh -T git@github.com 명령어를 사용한다. 추가로, SSH 키 관리를 편리하게 하기 위해 SSH 에이전트에 키를 등록하여 사용할 수도 있다.

위 처럼 Git은 설치와 초기 환경 설정 과정이 간단하며, SSH 키 기반 인증으로 보안성 높인 상태에서 작업할 수 있다. 이 기본환경 구성을 통해 Git의 다양한 기능을 원활히 사용할 수 있다.

Git 기본 사용법

Git은 효과적인 소스 코드 관리를 위한 필수 도구로, 여러 명령어를 통해 저장소를 초기화하고 파일 변경 사항을 관리하고 브랜치를 활용한 병렬 개발과 원격 저장소와의 연동까지 지원한다.

저장소 초기화와 클론

git init : 현재 디렉터리를 새로운 Git 저장소로 초기화한다. 이 명령어 실행 시 .git 폴더가 생성되어 Git이 버전 관리를 시작할 수 있게 된다.

git clone [저장소 URL] : 원격 저장소를 로컬에 복제하여 작업할 수 있도록 한다. 복제된 저장소는 원격 저장소와 연결돼 추후 변경 사항을 주고받을 수 있다.

파일 상태 확인, 추가, 커밋

git status : 현재 저장소의 상태를 확인할 수 있다. 변경된 파일, 스테이지에 올라간 파일, 커밋되지 않은 파일 등을 알 수 있으며, 작업 흐름을 관리하는 데 필수적이다.

git add [파일명] : 수정하거나 새로 생성한 파일을 커밋 전 스테이지 영역에 추가한다. 전체 변경 사항을 추가하려면 git add .이나 git add -A를 사용한다.

git commit -m "커밋 메시지" : 스테이지에 올라간 변경 사항을 하나의 커밋으로 저장한다. 커밋 메시지는 변경 내용을 명확하게 설명하는 것이 중요하다.

브랜치 관리

git branch : 현재 저장소 내 브랜치 목록을 확인하거나 새 브랜치를 생성할 수 있다.

git checkout [브랜치명] : 다른 브랜치로 전환할 수 있다
git checkout -b [브랜치명] : 브랜치를 생성하고 즉시 전환한다.

git merge [브랜치명] : 특정 브랜치의 변경 사항을 현재 브랜치에 병합하는 작업으로, 병합 시 충돌 해결이 필요할 수 있다.

로그 확인, 원격 저장소 설정 및 동기화

git log : 저장소의 커밋 히스토리를 확인할 때 사용하며, 커밋 ID, 작성자, 날짜, 메시지를 상세히 보여준다.

git remote add origin [원격 저장소 URL] : 원격 저장소를 연결하고 git remote -v로 설정된 원격 저장소 목록을 확인할 수 있다.

git push origin [브랜치명] : 로컬 커밋을 원격 저장소에 업로드하며, git pull은 원격 저장소의 변경 사항을 로컬로 받아와 병합한다.

팀 협업과 워크플로우

팀 협업과 워크플로우는 분산형 버전 관리 시스템인 Git의 핵심 기능으로, 효율적인 개발 프로세스를 마련하는 데 필수적이다. 대표적인 브랜치 전략으로는 Git Flow와 GitHub Flow가 있다. Git Flow는 대규모 프로젝트에 적합한 구조로, master 브랜치와 개발용 develop 브랜치, 기능 개발용 feature 브랜치, 릴리즈용 release 브랜치, 긴급 수정용 hotfix 브랜치로 구분하며 역할을 분리해 관리한다. 반면 GitHub Flow는 단순하고 빠른 개발을 지향하는 방식으로, 기본 브랜치에서 각 기능별 브랜치를 만들어 개발하고, 완료 시 풀 리퀘스트를 생성해 신속한 리뷰와 병합이 이루어진다.

풀 리퀘스트는 작업한 코드를 기본 브랜치에 병합하기 전에 팀 내에서 코드 변경 사항을 검토하는 과정이다. 리뷰를 통해 버그 수정, 코드 스타일 개선, 기능향상 등의 피드백을 주고 받으며, 승인이 완료되면 병합된다. 이러한 코드 리뷰는 코드 품질 유지와 지식 공유에 매우 중요한 역할을 한다.

개발 과정에서 동일한 파일의 같은 부분을 수정할 경우 충돌이 발생하며, Git은 자동 병합하지 못하고 충돌 영역을 표시한다. 충돌 해결을 위해 개발자는 해당 파일을 열어 코드를 수동으로 조정한 뒤, 변경 내용을 저장하고 스테이지에 올려 커밋하여 문제를 해결한다. 충돌을 최소화하기 위해서는 자주 병합하거나 리베이스(rebase)를 활용해 최신 상태를 유지하는 것이 바람직하다.

이슈 트래킹 시스템은 버그, 기능 개선, 작업 진행 관리 등을 효과적으로 수행할 수 있게 돕는다. GitHub, GitLab 등 플랫폼에서는 프로젝트 보드, 마일스톤, 라벨링 기능을 제공하여 작업 현황을 시각화하고 우선순위를 체계적으로 관리한다. 또한, 이슈와 커밋, 풀 리퀘스트를 연동해 개발 이력을 투명하게 관리하며, CI/CD 같은 자동화 도구와 연계해 효율적인 워크플로우를 구성할 수 있다. 이러한 통합 시스템은 개발 생산성과 협업의 질을 크게 향상시킨다.

고급 git 기능

Git의 고급 기능에는 Rebase, Cherry-pick, Stash, Tag, Submodule, Git Hooks 등이 포함된다. Rebase는 한 브랜치의 커밋들을 다른 브랜치 기반 위에 재적용하여 커밋 히스토리를 깔끔하고 직선형으로 만들어 주는 기능이다. 협업 시 history 관리에 유용하지만, 이미 공개된 커밋을 Rebase하면 충돌과 혼란이 생길 수 있어 주의가 필요하다. Cherry-pick은 특정 커밋 하나 또는 여러 개를 선택해 다른 브랜치에 적용하는 기능으로, 필요한 변경 사항만 골라 적용할 때 사용한다.

Stash는 작업 도중 변경 사항을 임시로 저장해 두고, 다른 브랜치 작업 후 다시 적용하는 기능이며, 작업 흐름 전환을 부드럽게 한다. Tag는 중요한 커밋에 별칭을 붙여 릴리즈 버전 관리 등에 활용한다.

Submodule은 하나의 Git 저장소 안에 다른 Git 저장소를 포함시키는 방법으로, 여러 프로젝트를 하나로 묶어 관리할 때 유용하다. 서브트리(Subtree)도 비슷한 기능을 제공하지만 복잡도와 관리 방식에서 차이가 있다.

Git Hooks는 커밋, 푸시, 머지 등 특정 이벤트 발생 시 자동으로 스크립트를 실행하게 해 커밋 메시지 검증, 자동 테스트, 배포 등 다양한 자동화 작업에 활용된다. 이를 통해 Git 운영과 협업 체계가 한층 효율적이고 안정적으로 운영될 수 있다.

Git 서버 및 서비스 소개

GitHub, GitLab, Bitbucket 등은 현재 가장 널리 사용되는 대표적인 Git 호스팅 서비스다. 이들 서비스는 원격 저장소 제공뿐 아니라 이슈 트래킹, 풀 리퀘스트(코드 리뷰), 위키, 통합 CI/CD 파이프라인, 보안 관리 등 다양한 기능을 지원한다. GitHub는 오픈소스 프로젝트와 대규모 개발자 커뮤니티가 활발하며, 소셜 코딩에 특화되어 있다. GitLab은 자체 호스팅이 가능하고 CI/CD 기능이 강력하여 기업 환경에 적합하다. Bitbucket은 Atlassian 제품군과의 통합성을 제공하며, 소규모 팀과 기업에서 주로 활용된다.

자체 Git 서버 구축은 주로 GitLab, Gitea, 혹은 Bare Git 저장소를 이용해 가능하다. 자체 서버 구축 시 네트워크 설정, 사용자 권한 관리, 백업 정책 등을 직접 관리할 수 있어 보안과 커스터마이징에 유리하다. GitLab CE(Community Edition)나 Gitea는 설치와 운영이 비교적 쉬우며, 웹 인터페이스를 통해 협업 및 코드 리뷰를 지원한다.

CI/CD(Continuous Integration/Continuous Deployment)와의 연계는 개발 프로세스 자동화를 위해 필수적이다. Git 서버에서 코드 푸시가 감지되면 빌드, 테스트, 배포 과정을 자동으로 실행할 수 있다. Jenkins, GitLab CI, GitHub Actions, CircleCI 등이 대표적인 CI/CD 도구 및 서비스이다. 이를 통해 개발 효율성과 품질을 크게 향상시키며, 일관된 배포 환경을 유지할 수 있다.

Git 주요 명령어

git init : 새로운 Git 저장소를 로컬에 생성 및 초기화한다. 프로젝트 시작 시 한 번만 실행한다.

git clone [저장소 URL] : 원격 저장소를 로컬로 복제해 가져온다.

git status : 현재 작업 디렉터리와 스테이지 상태를 확인한다. 변경 사항, 추가된 파일, 추적되지 않는 파일 등을 보여준다.

git add [파일명] : 변경하거나 새로 만든 파일을 스테이지에 올린다. 전체 변경 사항은 git add . 또는 git add -A로 추가한다.

git commit -m "메시지" : 스테이지에 올라간 변경 사항을 커밋하며, 메시지로 변경 내용을 기록한다.

git log : 저장소의 커밋 히스토리를 확인한다.

git branch : 브랜치 목록을 확인 또는 새 브랜치 생성(git branch [브랜치명]).

git checkout [브랜치명] : 작업할 브랜치로 전환한다. git checkout -b [브랜치명]은 브랜치 생성과 전환을 동시에 수행한다.

git merge [브랜치명] : 다른 브랜치를 현재 작업 중인 브랜치에 병합한다.

git remote add origin [원격 저장소 URL] : 원격 저장소를 추가한다.

git push origin [브랜치명] : 로컬 커밋을 원격 저장소에 푸시한다.

git pull : 원격 저장소 변경사항을 로컬로 가져와 병합한다.

git diff : 변경 사항을 비교하여 보여준다.

git reset : 커밋 또는 스테이징 상태를 되돌린다.

git stash : 작업 중이던 변경 사항을 임시로 저장하고 작업 공간을 깨끗하게 만든다. 나중에 다시 되돌릴 수 있다.