오전 수업

날짜	@January 16, 2024
태그	

깃의 사용 이유

- 소스코드 관리
- 협업 개발

깃 기본 설정

- 깃 커밋 정보
  - 소스코드 변경 내용과 함께 작성자의 이름과 이메일도 포함
  - 문제 상황이나 코드 리뷰시 중요한 정보
  - 특히 여러 명의 개발자가 동시에 작업해야 하는 상황에서는 더욱 중요
- 정확한 이름과 이메일 사용gt 중요

git config

- GIT 사용 환경 설정 도구(명령어)
  - GIT을 사용하기 위해서는 git config 명령을 통해 이름과 이메일 주소 설정 필수
  - 커밋 정보에 포함되어 누가 커밋을 했는지 알 수 있도록 해 줌

GIT 커밋 메세지 컨벤션

- GIT 커밋 메시지
  - 다른 개발자의 작업 내역이나 변경사항을 파악하는데 이용
  - 변경 이력 추적 및 문제 해결에 도움
- Conventional Commits
  - 가벼운 컨벤션으로 명확한 커밋 히스토리를 생성하기 위한 간단한 규칙 제공
  - 커밋 메시지에 신규 기능 추가, 문제 수정, 커다란 변화가 있음을 기술함

Conventional Commits

- 커밋 메시지 구조 (제목, 본문, 꼬리말)

```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

  - 타입
    - fix와 feat 이외의 타입 허용
    - 앵귤러 컨벤션 : build, chore, ci, docs, style, refactor, perf, test 등
  - 설명
    - 작업한 내용을 최대한 함축하여 대략 50자 이내로 작성
  - 본문 (선택사항)
    - 기본적으로는 선택사항이지만 **가급적 작성**
    - 자유로운 형식으로 필요시 여러 단락으로도 작성 가능
    - 무엇을 변경했는지 보다는 **왜 수정 했는지**를 설명
  - 꼬리말
    - 필요시 작성
    - 연관되어 있는 JIRA 이슈 번호 등을 작성
- 커밋 메시지를 통해 코드 리뷰어에게 정보 제공
  - Gerrit을 통한 코드 리뷰시 커밋 단위로 코드 리뷰
  - 빠른 리뷰를 위해서는 리뷰어에게 정보 제공 필요
  - 커밋 메시지 본문을 통해 정보 제공

GIT 브랜치 전략

- Git 브랜치

    - 독립적인 개발 공간 제공
    - 쉽고 빠르게 브랜치를 생성하고 이동 가능
    - 아이디어를 쉽게 시험해 볼 수 있음
- 브랜치 전략
    - 쉽고 편리한 브랜치, 계획없이 만들다 보면 남용될 수 있음
    - 브랜치를 효율적으로 사용하는 방법에 대한 다양한 브랜치 전략 제시
    - 가장 대표적인 브랜치 전략은 gitflow
    - 최근에는 배포 주기가 빨라지면서 다른 전략도 모색함(Github flow, Gtlab flow 등) → Gitflow 보다 단순한 형태로 배포가 매우 잦은 환경에 적합함.
- Gitflow 전략
    - master, develop, feature, release, hotfix 5가지 브랜치로 관리
    - 브랜치 마다 목적이 명확
    - 브랜치별 생명 주기에 따른 처리 주의

Git 히스토리 중요성

- Git History
    - Git은 SW 변경 사항을 커밋 단위로 기록하기 때문에 커밋 이력의 모음이 있다
    - Git 그래프나 로그로 확인한다.
- Git 히스토리가 중요한 이유
    - 레거시 코드 유지 보수에 중요
    - 버그 발생 시점 파악 및 문제 해결 실마리 제공
- 버그 발생 시점 파악 및 문제 해결 실마리 제공
    - 과거 커밋 시점으로 돌아가서 동작 확인
    - git checkout 명령어 이용

유용한 기능 - git stash

- 갑자기 끼어드는 작업

    - A 기능 작업 중인데, 갑자기 B 기능 작업 해달라고 한다면
    - 새 작업 위해 새로운 브랜치를 생성하거나 기존 작업 중인 브랜치로 이동 필요
    - 생각보다 자주 발생
- 브랜치 이동 안되는 상황
    - 수정중인 파일이 있을 경우 브랜치 이동 불가
    - 커밋 하거나 수정 전 상태로 리셋 필요
- git stash
    - 현재 작업중인 내용을 모두 stash 공간에 차곡차곡 쌓아 줌

    ## $ git stash save # 현재 작업 상태 백업

    ## 새로운 작업 진행

    $ git stash pop # 이전 작업 상태 복구

Git 이해와 활용

Git 커밋 정보

- 소스 코드 변경 내용과 함께 작성자의 이름과 이메일도 포함
- 문제 상황이나 코드 리뷰시 중요한 정보

Git 사용 환경 설정 도구(명령어)

- git config —global user.name “(본인 이름)”
- git config —global user.email “(싸피에서 등록된 이메일)”

Git 커밋 메시지

- 다른 개발자의 작업 내역이나 변경사항을 파악하는데 이용
- 변경 이력 추적 및 문제 해결에 도움

Conventional Commits

- 가벼운 컨벤션으로 명확한 커밋 히스토리를 생성하기 위한 간단한 규칙 제공
- 커밋 메시지에 신규 기능 추가, 문제 수정, 커다란 변화가 있음을 기술함
- 커밋 메시지 구조 - 제목, 본문, 꼬리말
    - 본문 작성 추천!! (코딩 스타일이나 간단한 수정은 예외)

Git 브랜치

- 독립적인 개발 공간 제공
- 쉽고 빠르게 브랜치를 생성하고 이동 가능
- 아이디어를 쉽게 시험해 볼 수 있음

브랜치 전략

- 쉽고 편리한 브랜치, 계획없이 만들다 보면 남용될 수 있음
- 가장 대표적인 브랜치 전략은 gitflow
    - master, develop, feature, release, hotifx로 관리

Git 히스토리

- 레거시 코드 유지 보수에 중요
- 버그 발생 시점 파악 및 문제 해결 실마리 제공
    - git checkout {커밋 번호}

유용한 기능 - git stash

- 수정중인 파일이 있어서 브랜치 이동이 불가능할 경우 사용
- 현재 작업중인 내용을 모두 stash 공간에 쌓아줌