GIT 이해하기

만든이 SI2팀 엄예지

초보도 이해하는 깃의 기초

목차

1. 깃 시작하기 with 소스트리

- ☑ 깃 사용 목적
- ☑ 깃의 흐름
- ☑ 소스트리란 무엇인가

깃 시작하기

준성 : 파일을 편집 전 상태로 되돌리기 쉽게 하려고 복사본을 만든건데..

수영 : 어떤 파일이 최신 파일이죠? 또 어떤 부분이 변경된건가요

성진 : 파일을 편집할때마다 매번 복사하려니까 너무 힘들어요

예지 : 아앗!! 성진선배가 제가 작업하고 있던 파일을 덧씌우는 바람에 제 작업물이 날아갔어요 ㅠㅠ

Git이란?

- 소스코드를 효과적으로 관리하기 위해 개발된 '분산형 버전 관리 시스템'
- 매우 빠른 속도와 분산형 저장소 지원
- 오픈 소스
- 효과적인 협업, 손쉬운 개발 및 테스트 환경 구축, 효율적인 배포관리

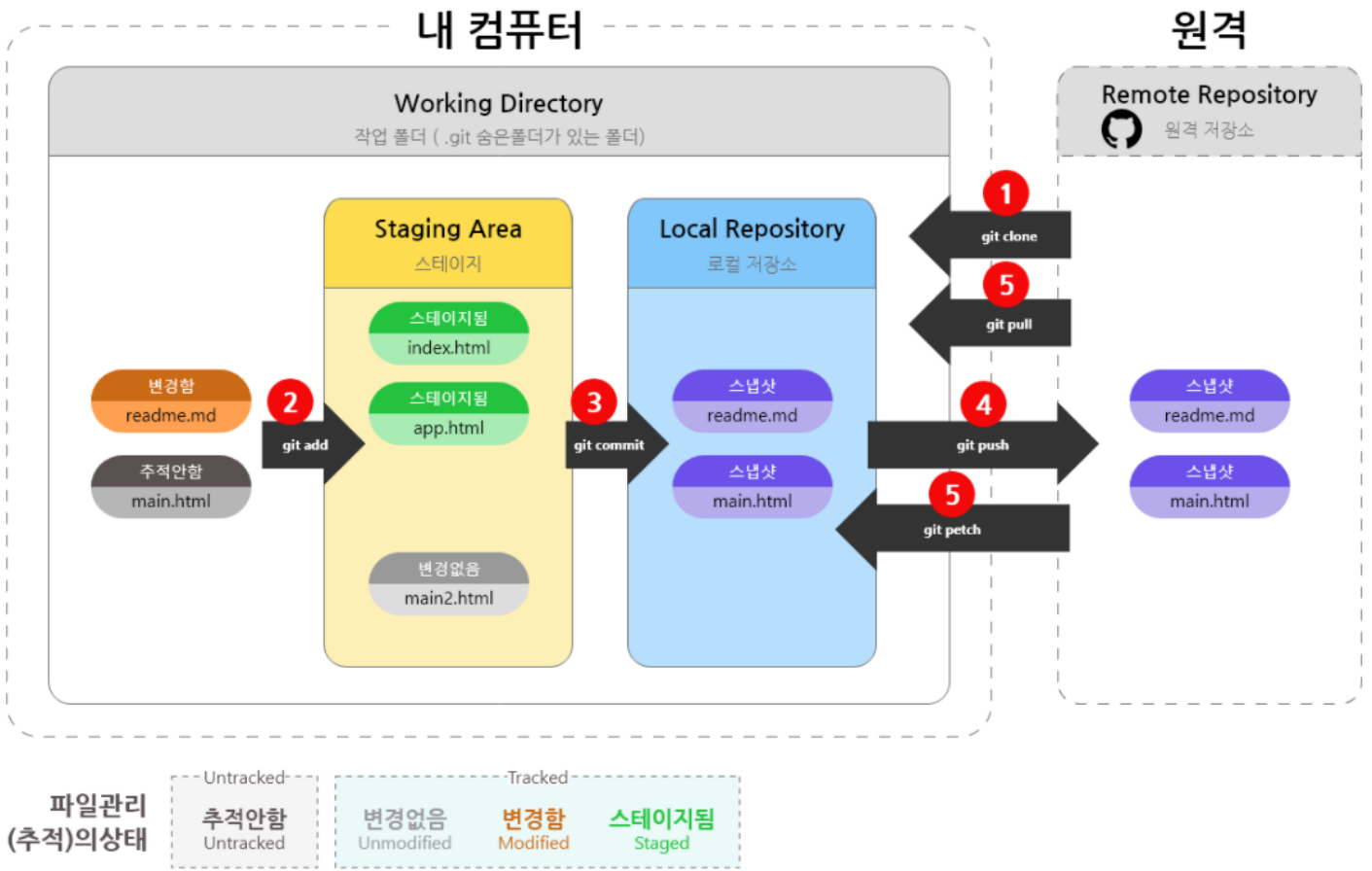
Git GUI

- Graphical User Interface, 초보자가 명령이나 작업을 이해하기 쉽도록 프로젝트 히스토리를 시각화하여 도와주는 도구
- Github desktop / SourceTree / GitKraken / SmartGit 등등..

저장소

- 원격 저장소 (Remote Repository) : 파일이 원격 저장소 전용 서버에서 관리되며 여러 사람이 함께 공유하기 위한 저장소
- 로컬 저장소 (Local Repository) : 내 PC에 파일이 저장되는 개인 전용 저장소

전체 과정



파일관리 (추적)의상태

Untracked	Tracked		
추적안함 Untracked	변경없음 Unmodified	변경함 Modified	스테이지됨 Staged

서버에서 Clone해오기

- Download ZIP : 그냥 순수하게 파일들만 압축해서 다운로드
- **Clone 주소복사** : 순수파일들과 이 프로젝트의 커밋 히스토리 정보까지 모두 다운로드 되어 새로운 로컬 저장소를 만들어줌
- 내 컴퓨터의 지정된 폴더에 .git이라는 폴더 생성(작업폴더)
- `git clone <저장소 url>`

- `git status` : 파일의 상태 확인하기

```
git status
On branch master
nothing to commit, working directory clean
```

작업폴더 구성

- **추적안함 (Untracked)** : 관리대상이 아님
- 추적함(Tracked)
  - **수정없음 (Unmodified)** : 변경이 없는 파일
  - **수정함 (Modified)** : 변경된 파일
  - **스테이지됨 (Staged)** : 스태이지에 올라간 파일

→ 바로 이전의 커밋을 기점으로 4가지 상태로 관리됨

☆ git add : 파일을 관리 대상으로 삼기 위해 실행하는 명령어, 스테이지로 올림

☆
```
git add<파일이름> //특정 파일만 추가하기
git add.          //모든 파일을 추가하기
```

작업폴더 구성

- git commit : 현재 스테이징된 파일들을 그대로 로컬저장소에 보관
- 하나의 히스토리 기록, 기준점
- git commit -m "참고할 설명 작성"
- git push : 원격 저장소에 업로드
- git push -u [저장소 이름] [브랜치 이름]
- git pull : 원격 저장소에 올라온 최신 파일을 로컬 저장소로 가져오기

주요 개념 정리

- Branch(브랜치) : 프로그램 소스 버전을 관리하기 위한 개념, 가지, 복사본
- Merge(병합) : 뻗어나간 브랜치 작업이 완료되면 상위 브랜치에 반영되는것
- Fetch(페치) : 원격 저장소의 업데이트 이력만 확인하고 병합은 하지 않음
- Checkout : 브랜치로 작업을 시작하기 위해 해당 브랜치를 선택하는것
- Git Ignore File : 원하지 않는 Backup File, Log File 등을 제외시키는 설정파일

Branch(브랜치)

- 필요에 의해 만들어진 각각의 브랜치는 다른 브랜치의 영향을 받지 않기 때문에 여러 작업을 동시에 진행할 수 있다.
- git branch : 내가 위치한 브랜치 확인, 현재 활성화된 브랜치(*)
- git branch (브랜치명) : 브랜치 생성하기
- git checkout (브랜치명) : 생성한 브렌치로 이동
- git branch -d (브랜치명) : 브렌치 삭제

Git Ignore File

- 저장소에서 관리하지 않아도 될 파일들
- 추적안함(Untracked) 상태로 만든다

```
# : comments

# no .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in the build/ directory
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory
doc/**/*.pdf
```

2. YML이란?

- ☑ YML이 무엇일까 (초급편)
- ☑ 음 이런게 있군

Yml파일이란? (개념)

- Yet Another Markup Laguage, 사람이 읽을 수 있는 데이터 직렬화 언어
- Xml 파일과 Json 파일
- Key : Value

Yml파일이란? (개념)

YML 이란?

Xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <apiVersion>v1</apiVersion>
  <kind>Pod</kind>
  <metadata>
    <name>hello-pod</name>
    <labels>
      <app>hello</app>
    </labels>
  </metadata>
  <spec>
    <containers>
      <name>hello-container</name>
      <image>tmkube/hello</image>
      <ports>
        <containerPort>8000</containerPort>
      </ports>
    </containers>
  </spec>
</root>
```

Object

Array

Key

Value

VS

Json

```
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "hello-pod",
    "labels": {
      "app": "hello"
    }
  },
  "spec": {
    "containers": [
      {
        "name": "hello-container",
        "image": "tmkube/hello",
        "ports": [
          {
            "containerPort": 8000
          }
        ]
      }
    ]
  }
}
```

Object

Array

Key

Value

Yml파일이란? (개념)

YML 이란?

Yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-pod
  labels:
    app: hello
spec:
  containers:
  - name: hello-container
    image: tmkube/hello
    ports:
    - containerPort: 8000
```

Object

Array

Key

Value

- -를 통한 Array 구분
- 가독성 굿
- 쿠버네티스에서 API 전송시 권장

Gitlab-ci.yml 작성

YML 이란?

- Git 리포지토리에서 호스팅되는 애플리케이션 코드
- .gitlab-ci.yml CI/CD 구성을 포함하는 저장소의 루트에서 호출

에서 `.gitlab-ci.yml` 파일, 당신은 정의 할 수 있습니다 :

- 실행할 스크립트입니다.
- 포함 할 기타 구성 파일 및 템플릿.
- 종속성 및 캐시.
- 순서대로 실행하려는 명령과 병렬로 실행하려는 명령.
- 애플리케이션을 배포 할 위치입니다.
- 스크립트를 자동으로 실행하거나 수동으로 트리거할지 여부.

Gitlab-ci.yml 작성

YML 이란?

```
stages:
  - build
  - test

build-code-job:
  stage: build
  script:
    - echo "Check the ruby version, then build some Ruby project files:"
    - ruby -v
    - rake

test-code-job1:
  stage: test
  script:
    - echo "If the files are built successfully, test some files with one command:"
    - rake test1

test-code-job2:
  stage: test
  script:
    - echo "If the files are built successfully, test other files with a different command:"
    - rake test2
```

병렬 실행

코드 전개

3. 깃, 깃 허브, 깃 랩?

☑ 깃과 깃 허브와 깃 랩의 차이점

Git 저장소를 서비스!

깃, 깃 허브, 깃랩

코드가 코드를 낳고…… → 어디다가 저장하지?

깃(코드 저장소)가 소프트웨어 관리 툴로 각광받고 있지만,
이것만으로는 충분하지 않아!

☆ **깃허브(GitHub)** / 비트버킷(Bitbucket) / **깃랩(GitLab)** 등

리눅스를 만든 천재 개발자, 버전관리의 필요성을
느껴 깃을 만들다 무려 오픈소스로!

리누스 베네딕트 토르발스(Linus Benedict Torvalds)

깃 허브

- 가장 큰, 깃 리포지토리 호스팅에 전문화된 최초의 대규모 웹사이트
- 마이크로 소프트에서 인수
- 인기 개발자 팔로우 가능, 대형 커뮤니티
- Push, Pull 속도가 매우 빠르다
- 설치버전, 개인 저장소를 사용할시에는 일부 비용 유료
  -> 무료 버전 사용시 모든 코드를 공개해야 함.
- 여러 다른 VCS를 기반으로 repos 가져오기를 지원
  (Git, SVN, HG, TFS)

깃 랩

- 설치형 Github (중앙저장소를 설치할 서버가 필요)
- 프로젝트 관리를 위한 자체 CI와 사용자 인터페이스 제공
- Private 서비스를 무료로 이용 가능
- 다른 형상관리툴에서 데이터 가져오기가 복잡
- Git 가져오기 지원

4. 토론시간

- ☑ 깃헙 vs 깃랩 무엇을 사용할까
- ☑ 아이엔씨 전체가 깃을 사용하기 위해
- ☑ 올바른 개발자의 길

감사합니다.