git bash를 이용한 협업

Git 기본 명령

1. git init
   .git 숨김 폴더를 생성한다. .git 폴더가 있어야 git이 추적한다.

2. git add
   변경된 내용을 staging 영역으로 옮긴다.

3. git commit
   staging 영역의 내용을 local repository의 내용으로 확정한다.

4. git push
   local repository의 내용을 remote repository로 올린다.

5. git pull
   remote repository의 내용을 local repository로 내려 받는다.

Git 기본 명령

6. git remote add origin 깃허브주소
   깃허브주소 를 remote repository(원격 저장소)로 등록한다.

7. git clone 깃허브주소
   .git 디렉터리를 포함해서 remote repository의 내용을 모두 local repository에 복제한다.

8. git branch 브랜치명
   새로운 브랜치를 만든다.

9. git checkout 브랜치명
   다른 브랜치로 이동한다.

10. git merge 브랜치명
    다른 브랜치 내용을 현재 브랜치에 병합한다. 이 때 두 브랜치에서 같은 파일을 동시에 수정하면 충돌(conflict)이 발생하므로 주의한다.

git

GitHub

협업 시나리오

1. 3명이 한 조입니다.
2. 조원1이 로컬 레파지토리와 원격 레파지토리를 모두 만듭니다.
3. 조원1이 가장 먼저 .gitignore 작업을 처리합니다.
4. 조원1이 프로젝트를 만들고 기본 세팅을 처리한 뒤 github에 올립니다.
5. 조원2,3은 github에 올라간 프로젝트를 자신의 PC로 복제합니다.
6. 조원1,2,3은 각자의 PC에 작업 브랜치를 생성해서 작업을 수행합니다.
7. 각 조원들은 자신의 기능 구현을 마치면 자신의 작업 브랜치를 main 브랜치에 병합한 뒤 github에 올립니다.
8. 다른 조원들은 github에 올라간 최신 버전의 main 브랜치를 자신의 main 브랜치로 가지고 온 다음 작업을 계속 진행합니다.

조원1이 .gitignore 파일을 생성합니다.

- 조원1이 로컬 레파지토리(local repository)와 원격 레파지토리(remote repository)를 만듭니다.
- 로컬 레파지토리에 가장 먼저 .gitignore 파일을 만듭니다.
  (https://www.toptal.com/developers/gitignore/)
- 로컬 레파지토리의 .gitignore를 원격 레파지토리로 올립니다.

```
$ git init                                  # 로컬 레파지토리 생성
$ git remote add origin 깃허브주소          # 원격 레파지토리 등록
$                                           # 로컬 레파지토리에 .gitignore 파일 만들기
$ git add .
$ git commit -m '커밋 메시지'
$ git push origin main
```

git

GitHub

조원1이 기본 세팅을 마친 프로젝트를 github에 올립니다.

- 조원1은 프로젝트를 생성하고 초기 세팅을 합니다.
- 조원1은 초기 세팅이 끝난 프로젝트를 github에 올립니다.
- 조원1이 본인의 https://github.com/에서 [Settings] – [Collaborators] 메뉴를 이용해서 조원2,3을 collaborator로 초대합니다. 조원2,3은 조원1의 초대를 수락합니다.(이메일을 이용한 수락)
  이제 조원2,3도 조원1이 만든 remote repository에 pull와 push를 할 수 있습니다.

```
$ git add .
$ git commit -m '커밋 메시지'
$ git push origin main
```

git

GitHub

조원2,3은 github에 올라간 프로젝트(main 브랜치)를 자신의 PC로 가져옵니다.

- 조원2,3은 자신의 PC에 workspace를 준비하고 github에 올라간 프로젝트를 복제합니다.(clone)
  workspace로 이동한 뒤 clone 명령을 내리면 workspace에 프로젝트가 생성됩니다.
- 복제를 하면 자동으로 remote repository로 등록됩니다.
- 복제가 완료되면 모든 조원들은 본인의 PC에 동일한 main 브랜치를 가지게 됩니다.

```
$ cd workspace            # workspace로 이동하기
$ git clone 깃허브주소    # 복제만 하면 끝(어떤 작업도 추가로 필요하지 않음)
```

git

GitHub

조원1,2,3은 각자의 PC에 각자의 브랜치를 생성해서 작업을 수행합니다.

- 조원1,2,3은 각자의 PC에 자신의 작업 브랜치를 생성합니다.
- 각 조원의 작업 브랜치 이름을 a,b,c라 칭하겠습니다.
- 조원1,2,3은 각자 기능 구현하는 동안에는 자신의 작업 브랜치 a,b,c만 push합니다.
  main 브랜치는 건드리지 않습니다.
- 충돌(conflict)을 방지하려면 조원별로 작업할 범위를 잘 나누는 것이 중요합니다.
  같은 파일을 2명 이상의 조원이 열지 않도록 작업 범위를 잘 나누세요.

```
$ git branch a     # 브랜치 a 생성
$ git checkout a   # 브랜치 a로 이동
$ git add .
$ git commit -m '커밋 메시지'
$ git push origin a
```

조원1이 기능 구현을 마쳤습니다.

- 조원1이 가장 먼저 기능 구현을 마쳤습니다.
- 조원1은 먼저 본인의 main 브랜치에 a 브랜치를 병합합니다. (local repository 작업입니다.)
  이제 main 브랜치에는 a 브랜치에서 작업한 내용이 모두 포함되어 있습니다.
- main 브랜치를 github에 push 합니다.

```
$ git checkout main    # main 브랜치로 이동
$ git merge a          # main 브랜치에 a 브랜치 병합하기
$ git push origin main
```

조원2,3은 조원1이 올린 main 브랜치를 가져온 다음 작업을 계속 진행합니다.

- 이제 github의 main 브랜치가 최신 버전이므로 조원2,3은 github의 main 브랜치를 각자 자리로 가져와야 합니다.
- 조원2,3은 본인의 main 브랜치로 이동한 다음 github의 main 브랜치를 pull 합니다.
- 조원2,3은 본인의 main 브랜치를 자신의 작업 브랜치(b 또는 c)에 병합합니다.

```
$ git add .
$ git commit -m '커밋 메시지'
$ git checkout main       # 본인의 main 브랜치로 이동
$ git pull origin main    # github의 최신 main 브랜치 가져오기
$ git checkout b          # 작업 브랜치 b로 이동
$ git merge main          # 작업 브랜치 b에 최신화된 본인의 main 브랜치를 병합
```

조원2가 기능 구현을 마쳤습니다.

- 조원2도 기능 구현을 마쳤습니다.
- 조원2는 먼저 본인의 main 브랜치에 b 브랜치를 병합합니다. (local repository 작업입니다.)
  이제 main 브랜치에는 b 브랜치에서 작업한 내용이 모두 포함되어 있습니다.
- main 브랜치를 github에 push 합니다.

```
$ git checkout main      # 본인의 main 브랜치로 이동
$ git merge b            # main 브랜치에 b 브랜치 병합하기
$ git push origin main   # github에 main 브랜치 올리기
```

git

GitHub

조원1,3은 조원2가 올린 main 브랜치를 가져옵니다

- 이제 github의 main 브랜치가 최신 버전이므로 조원1,3은 github의 main 브랜치를 각자 자리로 가져와야 합니다.
- 작업이 끝난 조원1은 main 브랜치를 가져오면 끝입니다.
- 아직 작업 중인 조원3은 main 브랜치를 가져온 다음 자신의 작업 브랜치 c에 main 브랜치를 병합 합니다.

```
$ git add .
$ git commit -m '커밋 메시지'
$ git checkout main       # 브랜치 main으로 이동
$ git pull origin main    # github의 최신 main 브랜치 가져오기
$ git checkout c          # 작업 브랜치 c로 이동
$ git merge main          # 작업 브랜치 c에 브랜치 main 병합
```

git

GitHub

드디어 조원3도 기능 구현을 마쳤습니다.

- 마지막 조원3도 기능 구현을 마쳤습니다.
- 조원3은 먼저 본인의 main 브랜치에 c 브랜치를 병합합니다. (local repository 작업입니다.)
  이제 main 브랜치에는 c 브랜치에서 작업한 내용이 모두 포함되어 있습니다.
- main 브랜치를 github에 push 합니다.

```
$ git checkout main      # 본인의 main 브랜치로 이동
$ git merge c            # main 브랜치에 c 브랜치 병합하기
$ git push origin main   # github에 main 브랜치 올리기
```

git

GitHub

조원1,2는 조원3이 올린 main 브랜치를 가져옵니다

- 이제 github의 main 브랜치가 최신 버전이므로 조원1,2는 github의 main 브랜치를 각자 자리로 가져와야 합니다.
- 조원1,2는 모두 작업이 끝났으므로 main 브랜치를 가져오면 끝입니다.
- 이제 조원1,2,3은 모두 최신 main 브랜치를 가지고 있습니다.

```
$ git checkout main
$ git pull origin main
```

협업 시 주의사항

1. push 하기 전에 반드시 local repository는 remote repository의 최신 버전을 유지해야 한다.
2. pull은 remote repository의 내용과 local repository의 내용을 비교하고 병합(merge)한다.
3. 조원1이 index.jsp를 수정하고, 조원2도 index.jsp를 수정했다면 충돌이 발생하므로 주의한다.
4. 충돌이 나면 git이 자동으로 fast-forward(최근 커밋으로 변경)할 수 있다. 하지만 대부분은 사람이 직접 일일이 확인해서 수정하는 작업을 해야 한다.
5. 기능마다 커밋이 하나씩만 있는게 좋다. 한 기능에 커밋이 여러 개 있으면 디버깅이 불편하다. 가능하면 기능이 완료된 뒤 커밋을 하자. 만약 커밋 이력이 너무 많다면 git rebase를 활용해서 커밋 이력을 합칠 수 있다.
6. 어느 시점으로 돌아가려면 git reset 또는 git revert를 이용한다.
   - git reset 커밋아이디
     1. 해당 커밋시점으로 이동하고 그 동안의 커밋 이력도 삭제
     2. 커밋 이력이 삭제되면 remote repository와 이력이 안 맞아서 push가 불가함
   - git revert 커밋아이디
     1. 해당 커밋시점으로 이동하지만 그 동안의 커밋 이력은 유지
     2. 이미 remote repository에 push를 한 상태에서 되돌아가려면 git revert 이용