목차

1. git 기초

1. git

리누스 토르발스가 만든 문서 관리 도구.

2. git의 핵심기능

1) 버전 관리

git은 문서 수정 시간, 수정 내용에 대한 정보를 기록함.

2) 백업

git 파일은 원격/온라인 저장소에 간단히 백업할 수 있음.

백업 서비스 중에는 github를 가장 많이 사용함.

3) 협업

문서를 주고받을 수 있고, 수정 내용에 대한 정보를 기록함.

협업 과정에서 일어날 수 있는 여러 문제들을 해결하는 기능을 제공함.

3. git 프로그램

git은 git 프로그램으로 사용함.

git 프로그램에는 여러가지 종류가 있음.
크게 GUI, CLI로 나눌 수 있는데, CLI가 더 효율적이기 때문에 많은 개발자들이 CLI git 프로그램을 사용함.
CLI git 프로그램에서 사용하는 명령은 리눅스 명령과 같음.

이 책에서는 CLI git 프로그램을 사용함.

메모 포함[이1]: 사실 실무에서는 GUI 기능이 탑재된 유료 git 프로그램을 사용한다고 한다.

2. 버전 관리

1. git 초기화 (git initiation)

디렉토리를 생성한 후, git 초기화 명령을 입력하면 해당 디렉토리 안에 .git 파일이 생성됨.

git 초기화 명령은 git init 임.

git init

2. 버전 생성

1) 스테이지, 리포지토리, 작업 트리

.git 파일 밖에 있는 공간을 작업 트리(working tree)라고 함.

git 초기화로 만든 .git 파일은 스테이지(stage)와 리포지토리(repository, 저장소)로 구분됨.
스테이지는 .git/index 파일, 리포지토리는 .git/HEAD 파일임.

스테이지는 버전으로 만들 파일들이 대기하는 공간임.
리포지토리는 생성한 버전을 저장하는 공간임.

2) 커밋

스테이지에 있는 파일들로 리포지토리에 버전을 생성하는 것.
또는 커밋으로 리포지토리에 생성된 버전을 의미함.

git commit <option>

git commit 명령어로 커밋함.

커밋 시에는 변경 사항을 메시지로 기록해 두는 것이 좋음. (커밋 메시지)
-m "<message>" : 커밋 메시지 작성 옵션.

3) 버전 생성 과정

1. 작업 트리에 파일을 생성 또는 수정함.
2. 버전으로 저장할 파일들을 스테이지에 올림. (git add 명령을 사용.)
3. 스테이지에 있는 파일들을 리포지토리로 커밋함.

git add <file>

메모 포함[이2]: 스테이징(staging) 또는 인덱스에 등록하기 라고도 한다.

3. 버전 정보 확인

1) git log

커밋 로그 출력 명령어.
커밋 로그에는 리포지토리에 저장된 각 버전들의 세부정보가 들어 있음.

커밋 로그 : git log 명령어를 사용했을 때 출력되는 정보.
커밋 해시 : 커밋을 구분하는 아이디. 영어와 숫자로 된 긴 문자열.

```
git log
```

2) git diff

변경 사항 확인 명령어.

(이런 게 있다 정도만 알면 될 듯.)

```
git diff
```

4. 작업 되돌리기

1) 수정한 내용 취소하기

git checkout 명령어로 수정한 내용을 취소할 수 있음.

```
git checkout -- <file>
```

2) 스테이징 취소하기

git reset HEAD 명령어로 스테이지에서 파일을 내릴 수 있음.

```
git reset HEAD<file>
```

3) 특정 커밋으로 되돌리기

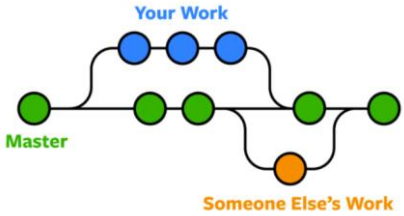
git reset, git revert 명령어 등으로 특정 커밋으로 HEAD를 옮길 수 있음.

커밋 해시만 안다면 다른 branch의 커밋으로도 갈 수 있음.

+ 기타 git 명령어들

git status : 깃 상태 출력 명령어.

3. Branch



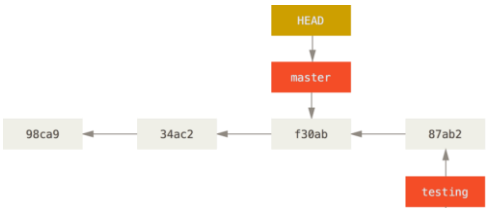
1. Branch

git에서 branch는 커밋을 가리키는 일종의 포인터.
여러 갈래로 퍼지는 데이터 흐름.

작업 트리에서는 현재 사용하고 있는 branch가 가리키는 커밋을 기준으로 파일들을 다루게 됨.

메모 포함[이3]: 그림을 보면 각 커밋의 커밋 해시를 branch가 가리키고 있고, 현재 사용하고 있는 branch는 HEAD가 가리키고 있다.

메모 포함[이4]: 다른 branch로 이동하거나 branch를 옮겨서 특정 커밋으로 간 경우에, 작업 트리에서는 이동 전에 다루던 파일들이 아니라, 해당 커밋(버전)의 내용을 다룰 수 있게 된다.



2. branch 기본 사용법

1) master branch

버전 관리를 시작하면(커밋을 하면) 기본적으로 생성되는 branch.

2) branch 생성

git branch 명령으로 branch를 생성할 수 있음.

```
git branch <이름>
```

기본적으로 branch는 최신 커밋을 가리킴.

3) branch 전환

git checkout 명령어로 사용할 branch를 전환할 수 있음.
(HEAD가 가리키는 branch를 전환하는 것.)

```
git checkout <이름>
```

다른 branch로 이동하는 것을 "해당 branch로 체크아웃한다."라고 함.

4) branch 정보 확인

git log 명령을 사용함.
git log --oneline --branches --graph 등의 명령어로 더 직관적으로 정보를 확인할 수도 있음.

5) branch 삭제

병합되었거나 더 이상 사용하지 않는 branch는 git branch -d 명령어로 삭제할 수 있음.

branch의 삭제는 리포지토리에서 완전히 없애는 것이 아니라 일시적으로 감추는 것임.
branch를 삭제한 후 동일한 이름으로 branch를 생성하면 이전의 내용을 다시 사용할 수 있음.

3. 분기(branch)와 병합(merge)

1) 분기

branch에서 뻗어 나오는 새 branch를 만드는 것.

branch 생성을 의미함.

2) 병합

분기되었던 branch들을 합치는 것.

```
git merge <이름>
```

git merge 명령어로 branch를 병합할 수 있음.
특정 branch에서 merge 명령어를 사용하면 해당 인자의 branch가 현재 branch로 병합됨.

상황에 따라 병합 양상이 달라짐.
1. 두 branch가 서로 다른 파일을 가지고 있는 경우 (버전이 다른 게 아니라 아예 다른 파일)
-> 병합 시 단순히 커밋에 파일이 추가됨.

2. 두 branch가 같은 파일을 가지고 있는 경우. (같은 문서의 다른 버전)
(내용이 동일한 부분은 병합되어도 유지됨.)
-> 파일이 변경된 위치가 겹치지 않는 경우, 두 branch의 수정 내용이 병합 파일에 각각 자동 반영됨.
-> 파일이 변경된 위치가 겹치는 경우, 병합 파일을 직접 수정하거나 병합 자동화 프로그램을 사용해야 함.

+ HEAD

branch들 중 현재 작업 중인 branch를 가리킴.

4. 백업 (github)

1. 지역 저장소 (local repository)

사용자의 컴퓨터에 있는 저장소.

지금까지 생성하고 버전 관리를 진행한 파일들은 모두 지역 저장소에 저장한 것.

2. 원격 저장소 (remote repository)

사용자의 컴퓨터가 아닌 다른 곳에 있는 저장소.

원격 저장소를 직접 구축하는 것은 번거로우므로, 보통 github를 사용해서 원격 저장소를 생성하여 사용함.

3. github

github에서는 원격 저장소를 제공함.

1) 장점

원격 저장소에서 바로 git을 사용할 수 있음.
지역 저장소의 파일들을 백업해 둘 수 있음.
협업을 위해 사용할 수 있음.
개발 이력을 남길 수 있음.
오픈 소스를 분석하거나 오픈 소스에 참여할 수 있음.

4. github(원격 저장소) 사용하기

1) 지역 저장소 생성하기

원격 저장소와 연결할 지역 저장소를 우선 만들어야 함.

지역 저장소를 만들고 커밋 생성.

2) 원격 저장소(repository) 생성하기

github 로그인 후 + 버튼을 눌러 생성함.

README : 저장소에 대한 소개와 설명을 작성하는 파일.
.gitignore : 이 파일에 특정 프로그래밍 언어를 지정하면 해당 언어의 기능을 깃에서 무시함.
License : 오픈 소스 라이선스 등의 라이선스를 선택하여 적용할 수 있음.

3) 원격 저장소와 지역 저장소 연결하기

원격 저장소와 지역 저장소를 연결하는 데에는 여러 가지 방법이 있음.
원격 저장소를 생성하면 연결 방법들을 확인할 수 있음.

원격 저장소와 지역 저장소는 한 번만 연결하면 됨.

origin : github 저장소 주소를 가리키는 단어. 주소를 그대로 쓰는 것은 불편하기 때문에 사용함.

4) push, pull 하기

최초 push 시에는, 지역 저장소의 branch와 원격 저장소의 branch를 연결해 줘야 함. (-u 옵션 사용.)
원격 저장소의 branch와 지역 저장소의 branch 사이의 연결은 처음 한 번만 연결하면 됨.

이젠 이 방법이 아니라 토큰을 이용해서 clone하는 게 좋음.
(https://gibro.tistory.com/9 참고)

push : 지역 저장소의 상황을 원격 저장소로 올리는 것.
pull : 원격 저장소의 상황을 지역 저장소로 내려 받는 것.

+ github 사이트에서 커밋하기

보통은 지역 저장소에서 커밋한 후 원격 저장소에 push하지만, github에서 파일을 작성, 수정하여 커밋할 수도 있음.

+ README 파일

필요성

https://dev-hyun.tistory.com/147

기본 작성법

https://m.blog.naver.com/jooeun0502/221956294941

https://gist.github.com/ihoneymon/652be052a0727ad59601 (중요. 정리 잘 되어있음.)

마크다운 문법 사용법

https://simhyejin.github.io/2016/06/30/Markdown-syntax/#headers

+ pull request

기본적 설명

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

설명+사용법(중요. pull request 전에 읽어보자.)

https://brunch.co.kr/@anonymdevoo/9

사용법

https://wayhome25.github.io/git/2017/07/08/git-first-pull-request-story/

+ wiki

프로젝트나 리포지토리에 대한 문서를 저장하는 공간.

README 파일은 주로 해당 리포지토리에 대한 간단한 설명을 제공하지만, wiki는 더 길고 세부적인 내용을 제공한다.

설명

https://docs.github.com/en/communities/documenting-your-project-with-wikis/about-wikis

https://git.jiny.dev/github/tools/wiki/

5. 협업

1. clone (cloning)

원격 저장소의 내용을 지역 저장소로 복제하여 가져오는 것.

원격 저장소를 기존에 연결된 지역 저장소 외에 다른 지역 저장소에서 사용하려면 원격 저장소의 내용 전체를 지역 저장소로 cloning 해야 함.

메모 포함[이5]: 다른 방법이 있는 지는 모르겠으나, 이 책에서 설명하는 바로는 그렇다.

cloning 하여 만든 디렉토리는 원격 저장소에 연결되어 있기 때문에, 커밋한 후 push하면 원격 저장소에 백업됨.

+ fork

다른 계정이 가지고 있는 원격 저장소를 내 github 계정으로 가져오는 것.

다른 계정이 가지고 있는 원격 저장소는 우선 fork 하고 clone 함.

+ github 로 협업하는 방식

1. fork, pull request 사용하기.
팀원마다 fork 로 리포지토리를 복사하여, 수정한 내용을 자신의 리포지토리에 자유롭게 반영함.
이후 upstream(fork 로 복사한 리포지토리)로 merge 하기 위해 pull request 를 사용함.

2. 하나의 리포지토리 공유하기.
하나의 리포지토리를 모든 팀원이 공유하여 사용함.
내용을 수정/추가하는 경우 새로운 branch 를 생성하여 작업한 후 merge 함.

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/getting-started/about-collaborative-development-models

2. 협업 기본 방식

1) 과정

1. 팀원 중 한 명이 github에 원격 저장소를 생성하고, 나머지 팀원들을 공동 작업자로 추가한다.
2. 팀원들은 지역 저장소를 생성하고 자신의 이름과 이메일을 설정한다.
3. 팀원 중 한 명이 첫 파일을 만들어 원격 저장소에 push한다.
4. 첫 파일을 push한 팀원을 제외한 팀원들은 원격 저장소를 cloning해 자신의 지역 저장소에 복사한다.
5. 이후 push와 pull을 사용해서 작업을 수행한다. -> 작업 전에는 항상 최신 커밋을 pull해야 함.

메모 포함[이6]: 공동 작업자로 추가된 사용자만 커밋을 추가할 수 있다.

2) branch의 사용

협업 시에는 주로 작업자마다 branch를 생성해서 작업함.

branch도 push하여 원격 저장소에 쉽게 추가할 수 있음.

branch 병합 방식

1. branch를 원격 저장소에 push함.
2. branch들을 병합하기 위해 pull request를 github에서 전송함.
3. 또 다른 작업자가 github에서 pull request를 확인하고, 병합하여 원격 저장소에 반영함.

메모 포함[이7]: 원격 저장소에 등록된 pull request는 공동 작업자 중 누구나 확인하고 병합할 수 있다.

3) 주의점

github에서 협업할 때는 원격 저장소의 최신 커밋을 pull한 후 push해야 함.
둘 이상의 컴퓨터를 연결하여 사용 또는 협업할 때는 push와 pull을 습관화해서 사용하는 것이 좋음.

6. vs code에서의 git 사용

1. local repository

1) git 초기화

작업 막대에서 세 번째(소스 제어) 항목을 클릭하면 git 초기화로 git 사용을 시작할 수 있음.

2) 스테이징

소스 제어 항목에 들어가서 파일이나 폴더에 마우스 커서를 올리면 + 기호가 나오는데, 이 기호를 누르면 해당 파일이나 폴더가 스테이징 됨.

스테이징을 하면 스테이지에 있는 파일 목록을 확인할 수도 있음.

3) 커밋

소스 제어 창 위에 메시지 창에 커밋 메시지를 입력한 후, 메시지 창 위의 체크 기호를 누르거나 Ctrl + Enter 키를 입력하면 커밋 됨.

4) 스테이징 전, 수정 내용 확인하기

소스 제어 창의 변경 내용 항목에 있는 파일들 중, M 기호가 붙은 파일의 이름을 클릭하면 변경 내용을 확인할 수 있음.

여기서 M은 수정되었다는 것을 의미하는 기호임.

+ 스테이징, 커밋 취소

메시지 창 위에 있는 . . . 기호를 클릭하여 스테이징이나 커밋을 취소하는 기능을 사용할 수도 있음.

+ U 기호

파일 옆에 있는 U 기호는 아직 한 번도 커밋되지 않았다는 것을 의미함. (Untracked)

+ 소스 제어에 뜨는 숫자

변경된 파일의 개수만큼의 숫자가 나타남.

2. remote repository

1) 원격 저장소 연결

1. github에서 원격 저장소를 생성한다.
2. 원격 저장소의 github 주소를 복사해서 vs code이 터미널 창에 연결 명령을 입력한다. 오류 메시지가 뜨지 않았으면 성공한 것.
(명령 : git remote add origin <주소>)

```
git remote add origin <주소>
```

이젠 이 방법이 아니라 토큰을 이용해서 clone하는 게 좋음.
(https://gibro.tistory.com/9 참고)

2) github로 push하기

소스 제어 창의 메시지 창 위에 있는 . . . 기호를 누른 후 push 기능 사용.

원격 저장소를 선택하여 push할 수도 있음.

3) github에서 pull하기

소스 제어 창의 메시지 창 위에 있는 . . . 기호를 누른 후 pull 기능 사용.
'가져올 위치 . . .' 항목을 사용함.

pull할 때는 가져올 branch를 선택함.

+ 이외의 기능들

대부분의 기능들은 소스 제어 창의 메시지 창 위에 있는 . . . 기호를 누르면 찾을 수 있음.

7. git CLI 명령어 정리

1. 환경 설정

1) git config --global user.name "<이름>", git config --global user.email "<이메일>"

사용자 정보 저장 명령어. (git은 커밋을 만들 때마다 사용자 정보도 함께 저장함.)

--global 옵션을 추가하여 현재 pc의 모든 저장소에서 동일한 사용자 정보를 사용하도록 설정할 수 있음.

2) git init

작업 디렉토리를 깃 초기화하는 명령어. (git 환경 구축)

+ 파일의 상태

tracked	-> 한 번 이상 버전 관리를 한 파일. (추적 중인 파일.)
untracked	-> 아직 한 번도 버전 관리가 되지 않은 파일. (추적 중이 아닌 파일.)
modified	-> 수정되었고 아직 스테이징되지 않은 상태.
unmodified	-> 수정되지 않은 상태.
staged	-> 스테이징된 상태.

+ HEAD

현재 작업 트리가 어떤 버전을 기반으로 작업 중인지를 가리키는 포인터.

기본적으로 master branch를 가리키고 있음. master branch는 기본적으로 가장 최근 커밋을 가리킴.

2. 출력

1) git status

git 상태 출력 명령어.

2) git log

저장소에 들어 있는 커밋 기록 확인 명령어.

--stat 옵션을 사용하여 관련 파일들까지 출력할 수 있음.

3) git log --oneline --branches --graph

branch와 커밋들의 관계를 그래프를 출력하는 명령어.

3. 버전 관리

1) git add <file1> <file2> . . .

명시한 파일을 스테이징하는 명령어.
폴더 또한 이름을 작성하면 됨.
한 번에 모두 스테이징하려면 * 기호를 사용함. (git add *)

2) git commit -m "<message>"

스테이지의 파일들을 커밋하는 명령어.

-m 옵션을 사용해 명시한 커밋 메시지를 함께 기록해야 함.

3) git commit -am "<message>"

한 번 이상 커밋한 파일을 한 번에 스테이징하고 커밋하는 명령어.
(기능이 불확실하므로 최대한 사용을 자제하기.)

4) git commit --amend

커밋 메시지 수정 명령어.
가장 최근의 커밋 메시지를 수정할 수 있음.

4. branch

1) git branch

branch 확인 명령어.

2) git branch <이름>

branch 생성 명령어.

3) git checkout <이름>

명시한 branch로 branch를 변경하는 명령어.

4) git merge <이름>

현재 명시한 branch를 현재 branch에 병합하는 명령어.
파일이나 파일의 내용이 겹치지 않으면 단순 병합되고, 겹치면 수작업으로 수정해 줘야 함.

5) git branch -d <이름>

명시한 branch를 삭제하는 명령어.
완전히 제거하는 것이 아니라 잠시 숨겨두는 것.

5. github 사용

1) git remote add origin <원격 저장소 웹 주소>

원격 저장소를 현재 지역 저장소로 연결하는 명령어.

cloning을 한 후에 remote도 해줘야 함.

2) git remote -v

원격 저장소와 지역 저장소의 연결 상태를 확인하는 명령어.

3) git push

push 명령어.

최초 push 시에는 -u origin master 옵션을 추가하여 지역 저장소의 branch를 원격 저장소(origin)의 master branch에 연결해 줘야 함. (git push -u origin master 으로 작성함. origin(원격 저장소)의 master branch에 연결한다는 의미.)

이후에는 git push만 작성하면 됨.

4) git pull origin master

pull 명령어.

기본 원격 저장소가 origin이고, 지역 저장소의 기본 branch가 master이기 때문에 git pull만 입력해도 됨.

6. github로 협업하기

1) git clone <원격 저장소(github) 주소> <디렉토리 이름>

cloning 명령어.

해당 디렉토리로 해당 원격 저장소(커밋과 branch들)가 복사됨.

해당 디렉토리 이름이 존재하지 않으면 새로 생성함.

+ 1학년 2학기 스플럭 스터디에서의 git 사용방식

1. github 그룹에 참가.
2. github 그룹의 원격 저장소를 내 계정으로 포크.
3. cloning하여 로컬 저장소 생성. -> 생성할 폴더로 이동해서 'git clone <주소>' 명령어를 사용함.
   -> 이때 주소는 내 계정으로 포크한 원격 저장소의 주소 사용.
   -> 이렇게 생성된 로컬 저장소에는 .git 파일이 생성되어 있음.
   -> git init이나 git remote 명령어는 사용하지 않음.
   -> cloning 한 직후에는 pull 할 필요 없음. (이미 최신 상태임.)
4. 로컬 저장소에서 파일 생성.
5. 로컬 저장소에서 스테이징, 커밋.
6. 원격 저장소(내 계정.)로 push. -> 최초 push 시에는 'git push -u origin master' 명령어를 사용함.
   -> 그냥 git push인 것 같기도 함.
7. 내 계정의 원격 저장소에서 그룹의 원격 저장소로 풀 리퀘스트 전송. (github 웹페이지 메뉴에서.)

+ 추가 정보

git init을 취소할 때는 .git 파일을 삭제함.
.git 파일이 여러 군데 존재할 경우 작업이 꼬이기 매우 쉬우므로, .git 파일 생성 시에는 특히 주의해야 함.
branch를 삭제할 때는 로컬 저장소 branch 삭제 명령어와 원격 저장소 branch의 삭제 명령어가 다름.

+ 인증 방식 변경

이젠 토큰을 이용해서 clone하는 게 좋음.
(https://gibro.tistory.com/9 참고)