git 정리

작성일: 2020.03.05
작성자: 도원진

로컬저장소 커밋관리

git init

```
git status //StageArea 상태를 보여줌
git add hello1.txt //stageArea로 이동
git diff //파일을 수정해서 변경사항이 생기면 이 명령어로 확인함. 사항을 상세하게 + , - 로 설명
git commit -m "커밋 메세지"
git commit -am "Message" //add도 동시에 해결함.(단, 커밋된적 없는 파일은 직접 add해줘야 함)
git commit --amend //커밋메세지 수정가능
```

- **상태 조회**
  - git log : 커밋아이디 확인하기 (앞 7자리만 복사해도 가능)
  - git status: 상태 조회(untracked, unstaged, staged)
- **커밋되돌리기(시간여행이 아님, head만 이동하여 작업할 브랜치 선택)**
  - git checkout 체크썸
  - git checkout **-** : 최신커밋으로 이동(- 는 최신커밋아이디를 뜻함)

원격저장소 만들기

github에서 repository 생성후, 주소복사

- 로컬저장소에 원격저장소 주소를 등록

  git remote add origin https://Github.com/wonjin-do/iTshirt.git
- 로컬의 커밋----(push)----> 원격저장소
  - **git push origin master**
  - master브랜치가 가르키는 커밋을 origin(원격저장소 별칭) 에 push해라
- 원격저장소----(push)----> 로컬의 커밋
  - **git pull origin master**

Part1

1. origin과 master의 기본 개념

origin

원격저장소 별칭

따라서, `git remote add origin https://github.com/wonjin-do/iTshirt.git` 은 https://github.com/wonjin-do/iTshirt.git 원격저장소를 origin이라는 이름으로 로컬저장소에 등록한다 라는 뜻이다.

origin이 아닌 myOrigin이라고 하면 닉네임은 origin이 아닌 myOrigin 으로 등록됨

PUSH 설정

`git push --set-upstream origin master` 를 통해 원격저장소의 master를 지정한다.

위의 `git remote add origin url` 은 원격저장소를 등록하는 것이므로 차이를 구별할 것!

줄기종류	의미
master	로컬 저장소의 브랜치
origin/master	원격저장소의 브랜치

2. 깃의 원리

커밋은 Delta(차이점)가 아니라 SnapShot(스냅사진)

- 버전관리
  - SVN
    - 커밋에 **바뀐 내용만** 저장
    - 버전을 보여줄 때, 맨 처음까지 거슬러 올라가며 바뀐 점을 모두 반영해야함. 즉, 계산의 수고를 해야함
  - Git
    - 커밋에 **전체코드를 저장**
    - 바로 앞 선 커밋과 비교 1번만 하면 된다.
- Git의 4가지 상태
  - Untracted
    1. **추적안됨**
  - Tracked
    2. **스테이지됨**
    3. **수정없음(unStaged)**(수정없음 일지라도 이전에 커밋한 이력이 있으면, `수정없음` 상태도 보이지 않게 스테이지에 파일이 올라와져 있다.)
    4. **수정함(unStaged)**

3. 브런치(협업하기)

`git log --all --graph --oneline` : 브랜치 flow보기

3-1. 브런치 만들고 커밋진행하기

`git branch 브랜치이름` : 브랜치 생성

- 시간순

(과거)---------->(현재)

커밋1 <----- **커밋2** <----- **커밋3** <----- **커밋4**

- 커밋은 자신의 이전 커밋을 가르키고 있다.
- 첫 커밋이 만들어진 순간 master 브랜치(포인터)로 진행이 된다.
- 브렌치를 줄기라고 생각하지 말고 포인터라고 생각하라.

커밋1(줄여서 C로 표기)

↑

master (줄여서 M으로 표기)

ex1)

C1 <---------- **C2**

↑

M

ex2)

C1 <---------- **C2** <---------- **C3**

↑

M

- B1 브랜치 생성

```
                                                                **B1**
```

↓

C1 <---------- **C2** <---------- **C3**

↑

M

- 독자적으로 B1 브랜치에서 커밋진행

B1

↓

C4

↙

C1 <---------- **C2** <---------- **C3**

↑

M

- M브렌치에서 B2 브랜치 생성

B1

↓

C4

↙

C1 <---------- **C2** <---------- **C3**

↑

M, B2

- B2브런치에서 커밋진행

B1

↓

C4

↙

C1 <---------- **C2** <---------- **C3**

↑ ↖

M **C5**

↑

B2

- 위 그림에서 HEAD 포인터는 자유자재로 이동하여 checkout할 수 있음.
- HEAD가 Master가 아닌 다른 브랜치를 가르키면 `Detached HEAD` 라고 부름

3-2. 브런치 병합하기

병합종류	특징	비고
Merge commit	두 브랜치 간의 다른 라인에 다른 코드가 있을 때, 상호 합집합이 이뤄지는 경우	충돌없음
Fast - forward	두 브랜치 중 한개가 다른 한 개에서 앞선 경우	충돌없음
Conflict	같은 라인에 다른 코드	충돌

과정1)

master는 뿌리의 기둥이며, 기능단위로 `브랜치`를 만들고 그 브랜치를 검증하고 master에 병합한다.

아래 상황은 , 브랜치 두 개를 생성 `feature/cart` `feature/detail-page`하고 각각 커밋을 진행한 케이스

master는 분기가 일어나는 지점에 멈춰있다. 각각의 다른 브랜치는 기능을 구현한 상태다.

```
C5      feature/cart
|   C4  feature/detail-page
|    |
|   C3
|  /
C2      master, origin/master, origin/HEAD
|
C1
```

과정2)

- `master`브랜치로 checkout 한 후, `feature/detail-page`브랜치와 fast-forward병합한다.
  - 편의를 위해 feature/detail-page 브랜치는 아래에서 더 이상 표기하지 않는다.
  - fast-forward병합은 브랜치 그림상에서 별도의 줄기를 필요로 하지 않는다.
    - 포인터만 이동하기 때문.
    - 따라서, 브랜치가 3개(master, detail-page, cart)여도 줄기는 두 갈래다.

```
//fast-foward Merge
C5       feature/cart
|   C4   feature/detail-page,   master
|   |
|   C3
|  /
C2       origin/master, origin/HEAD //master브랜치 위로 이동함 즉, fast-forward방식으로 병합함
```

- push를 하지 않았기 때문에 원격저장소의 origin/master브랜치는 2개 커밋 뒤 떨어져 있다.
  - master 브랜치를 push해서 같은 커밋을 가르키도록 조치한다.

```
//Master브렌치 PUSH진행
C5       feature/cart
|    C4   feature/detail-page, master, origin/master, origin/HEAD
|    |
|   C3
|  /
C2
```

과정3)

- `feature/cart`브랜치에 `master`브랜치를 병합한다.
  - `feature/cart`에 checkout을 한 후, `master`를 merge한다.
  - `master`에 `feature/cart`를 올리는게 (병합) 아니다.
    - 다른 브랜치에서 merge를 진행하고 충돌을 해결한 후에 master에 반영하기 위해서.

이때, `커밋하지 않은 변경사항`이 우선 생성된다.(충돌 코드 수정 후 바로 커밋이 될 것임)

```
□       <!-- 커밋하지 않은 변경사항(충돌이 있는) -->
| \
C5  \    feature/cart  origin/feature/cart
|   C4   master, origin/master
|    |
|   C3
|  /
C2
```

과정4)

- 친절하게 git 이 병합에 필요한 두 커밋사이에 차이를 발생시키는 충돌코드를 잡아냄.
- 어떻게?
  - 병합이 이뤄지는 소스파일에 충돌이 일어나는 라인에 기존코드와 새로 추가된 코드를 구분을 해놓음.

    - 또한, 그 소스파일을 `unstaged` 상태로 간주.
    - 개발자가 코드수정(충돌조정)을 하도록 요구함.

과정5)

- 커밋이 완료된 상황
    - `feature/cart` 브랜치는 `origin` 과 3개의 커밋차이를 보임
        - 계산법 : C5 에서 C4로 가는 edge의 갯수의 총합

```
C6   feature/cart<!--origin과 3개의 커밋차이를 보임 -->
| \
C5  \    origin/feature/cart <!-- 로컬의 feature/cart브랜치 위로 이동함  -->
|   C4   master, origin/master
|    |
|   C3
|  /
C2
```

- feature/cart 브랜치를 원격저장소에도 반영하자 (PUSH)

```
C6      feature/cart     origin/feature/cart
| \
C5  \    <!-- push를 통해 origin/feature/cart브랜치 위로 이동함  -->
|   C4   master, origin/master
|    |
|   C3
|  /
C2
```

과정6)

- Master에 Merge하기
    - master에 checkout한 후, `feature/cart` 브랜치를 master에 Merge
    - fast-forward merge가 됨.

```
C6       feature/cart   origin/feature/cart     master
| \
C5  \
|   C4   origin/master  <!-- merge를 통해 master브랜치 위로 이동함  -->
|    |
|   C3
|  /
C2
```

과정7)

- Master Push하기

```
C6      // feature/cart origin/feature/cart     master  origin/master
| \
C5  \
|   C4  // feature/detail-page  origin/feature/detail-page
|    |
|   C3
|  /
C2
```

3-3. pull request (병합요청하기 맛보기)

한 저장소에 프로젝트 담당자가 여럿일 때, 병합을 승인받는 작업을 진행할 수 있게 도와준다.

1. 브랜치 새로 생성하기
2. 커밋, PUSH 진행하기
3. Merge
    - 원격저장소
        - 깃헙에서 pull request 진행하기
        - merge 승인을 하면 merge진행
        - origin/master 가 merge된 커밋을 가르킴
    - 로컬저장소
        - 소스트리에서 Fetch를 진행하면, master는 기존과 동일한 커밋을 가르키고
        - origin/master는 merge된 커밋을 가르킴
    - master로 checkout하고 , pull을 진행하여 마무리한다.

3-4. Release 하기 (태그 붙이기)

- 소스트리에서 `태그` 선택
    - 특정 커밋에 붙이는 `태그`
- 버전 적어준다
    - 예) v1.0.0
- push한다
    - 아래 `모든 태그 푸시`선택후 PUSH
- GitHub 저장소에서 `release`를 선택하여 조회 확인한다.

4. 둘 이상의 원격저장소 협업하기

PUSH 규칙

1. 기본적으로 원격저장소를 만든 본인만 PUSH를 할 수 있다
2. Settings 에서 collaboration에 협업자를 등록하면 타인도 PUSH할 수 있다.

한 저장소에서 작업하는 협업자가 많아 진다면?

- 브랜치의 갯수도 많아져서 복잡하다.
- 따라서, 원격저장소 복제 (Fork)를 이용한다.

Fork & Pull Request

- 원격저장소 복제를 하여 별도로 커밋진행.
- 이후, 원본저장소의 주인에게 pull request를 하여 자신의 성고를 기여한다
- reactjs.org/docs/how-to-contribute.html 참고

묵은 커밋을 새 커밋으로 이력 조작하기(REBASE)

주의사항

- rebase할 브랜치가 push 되기 전에만 가능하다
- 만약 이를 어기고 그 브랜치를 push한 후, 로컬에서 rebase가 진행된다면 엄청 복잡한 오류를 발생시킨다.
- 왠만하면 REBASE 쓰지말것
- 그냥 Merge를 쓸것
    - 이유. ReBase와 Merge의 결과는 동일하기 때문!!!!!

5. 실무사례

5-1. amend 수정못한 파일이 생겼을 때

remote - repo 에 이미 push한 상황이여도 amend로 커밋 수정가능함.

1. `마지막 커밋 정정` 옵션을 통해 커밋 생성
2. 강제PUSH

5-2. cherry-pick 커밋 하나 분리하여 현재 브랜치로 옮기는 방법

5-3. checkout / reSet / reVert 차이점

명령어	특징	비고
상황	**커밋 히스토리가 A-B-C 순.** **master는 C를 가르키고 , HEAD는 master를 가르킨다.**	
checkout	과거 미래 시간이동이 아님, 한 저장소에 있는 협업자들 중 원하는 협업자(브랜치)로 이동하는 방법 (브랜치를 가르키는 HEAD가 변경된다.커밋을 가르키는 브랜치가 이동하는 것이 아님)	협업자 교체
reset	(시간여행) 커밋을 가르키는 **브랜치가 이동**함. 브랜치가 C를 가르킬 때, A를 가르키도록 이동. **B,C커밋은 삭제됨.**	복권사기전 시간으로 이동
-mixed, soft	A 커밋으로 브랜치가 이동할 때, **A 직후인 B 커밋에 있었던 변경사항을 살려둠, mixed**: B의 변경사항을 working Area에 놓아둠 **soft**: B의 변경사항을 stage에 놓아둠	잘못된 복권번호를 적던 **당시만 기억**하고 있음
- hard	A 커밋 직후 기록들은 모두 삭제	**기억잃고** 과거로감.
revert	B, C 커밋을 남겨두고 타임머신을 타고 A로 이동.	복권사기전 시간으로 이동**(단, 이동할 시점 이후의 기록들을 모두 기억)**

```
git reset --hard //바로 이전 버전으로 이동
git reset --hard 커밋ID //커밋ID버전으로 이동
```

```
git revert <타겟커밋의 바로 뒤 커밋ID>
```