Information

2019년 4월 3일 수요일 오후 3:07

- Repository : 작업자가 변경한 모든 내용을 추적하는 공간
- Working Tree : 저장소를 어느 한 시점을 바라보는 작업자의 현재 시점이다.
- Checkout : 작업자의 작업트리를 저장소의 특정 시점과 일치 하도록 변경하는 작업
- staging Area : 저장소에 커밋하기 전에 커밋을 준비하는 위치 변경사항을 적용하기 전에 한번 더 변경사항을 정리하고 다듬을수 있는 기회를 제공한다. 변경사항을 추가하기 위해서는 git add 를 사용한다. 커밋 예정인 변경사항이 있다고 보면 된다.)
- branch : 브랜치라는 것은 하나의 개발 라인을 의미 한다.

한개의 프로젝트에서도 여러개의 개발 라인이 존재 할 수 있다.
가장 기본이 되는 master branch에서 버그 수정이나 특정 기능을 추가하기 위해서 개발라인을 따로 두고 작업할 수 있다. 이러한 브랜치에는 HEAD(branch head)라는 것이 있는데 이는 한개의 브랜치 내에서 가장 최근에 커밋이 된 reference이다.

- 예를 들면 branch apple에 3개의 commit이 있는데 이중에 가장 최근에 추가된 커밋이 HEAD가 된다.

- master : master 브랜치는 복사해온 저장소 내의 HEAD의 복사본이다.
- origin : origin 은 단지 git가 복사해 온 저장소를 가리키기 위해 기본적으로 사용하는 이름
- push : 웹 상의 원격 저장소로 변경된 파일을 업로드하는 것을 푸시(PUSH)라고 합니다. push를 실행하면, 원격 저장소에 내 변경 이력이 업로드되어, 원격 저장소와 로컬 저장소가 동일한 상태가 됩니다.
- clone : 원격 저장소의 내용을 통째로 다운로드하는 것을 말합니다. 복제한 저장소를 다른 PC에서 로컬 저장소로 사용할 수 있게 됩니다.
- pull : 원격 저장소를 공유해 여러 사람이 함께 작업하게 되면, 모두가 같은 원격 저장소에 push하게 됩니다. 이럴 때 다른 사람의 변경사항에 대해서도 내 로컬에 적용 할 필요가 있습니다. 이 때 사용하는 것이 'PULL' 입니다.
- Rebase : Rebase 는 두 브랜치가 나뉘기 전인 공통 커밋으로 이동하고 나서 그 커밋부터 지금 Checkout한 브랜치가 가리키는 커밋까지 diff를 차례로 만들어 어딘가에 임시로 저장해 놓는다. 그 후 master 에 저장해 놓았던 변경사항을 차례대로 적용한다. Merge이든 Rebase든 둘 다 합치는 관점에서는 서로 다를 게 없다. 하지만, Rebase가 좀 더 깨끗한 히스토리를 만든다.
Rebase의 경우는 브랜치의 변경사항을 순서대로 다른 브랜치에 적용하면서 합치고 Merge의 경우는 두 브랜치의 최종결과만을 가지고 합친다.
- Fetch : master 나 다른 branch 에서 작업한 내용이 내 로컬 repository 와 버전이 맞지 않을때 최신버전으로 업데이트
- merge : git merge <branch name>브랜치의 변경사항을 master 에 반영. 혹은 다른 두 가지를 merge 합쳐 준다는 의미 입니다.
- log : 소스코드 버전의 id 를 알수있다
- Diff : git diff <original branch> <diff branch> merge 하기 전 어떤 내용이 변경되었는지 알아볼 수 있다.
- Blame : 코드의 한줄한줄 누가 마지막으로 commit 한 지 정보가 필요할때

Command - 기본

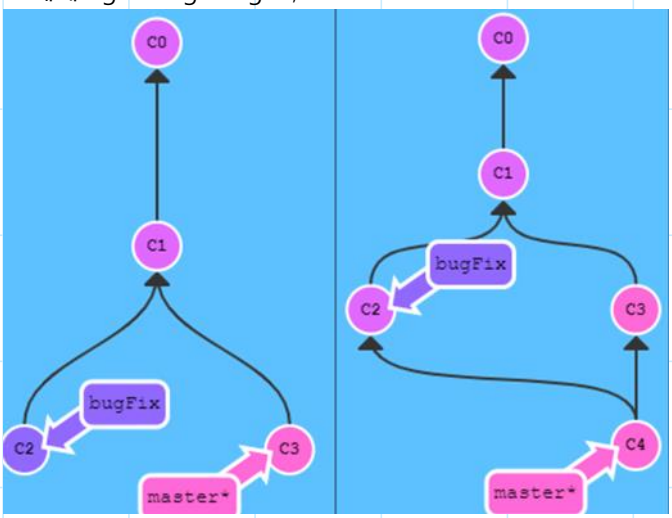
2019년 4월 3일 수요일 오후 3:07

< commit >

1. 의미 : git 저장소에 모든 파일에 대한 스냅샷을 기록
2. 형식 : git commit;

< merge >

1. 의미
 - 두개의 부모 브렌치를 가리치는 특별한 커밋을 만듬
 - 부모 브렌치의 모든 작업내역을 포함한다.
2. 형식 : git merge (Name);
3. 예제 : git merge bugFix;

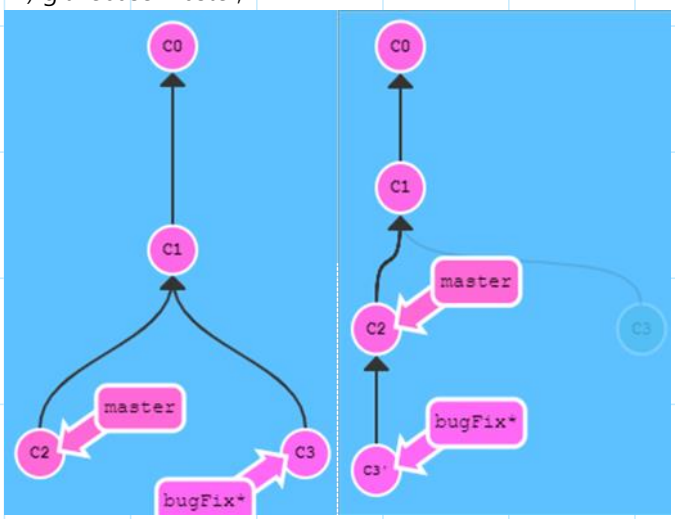


< branch >

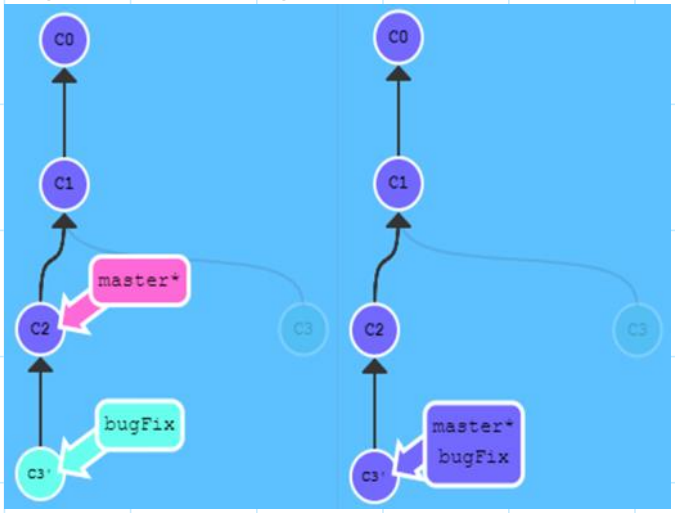
1. 의미 : 특정 커밋에 대한 참조
2. 형식 : git branch <Name>; (git checkout (Name);)
3. 예제
 0) 갱신 : git remote update;
 1) 삭제 : git branch -d <Name>;

< rebase>

1. 의미 : 커밋들을 모아서 복사한 뒷, 다른 곳에 저장하는 것
2. 형식 : git rebase (Name);
3. 예제
 1) git rebase master;



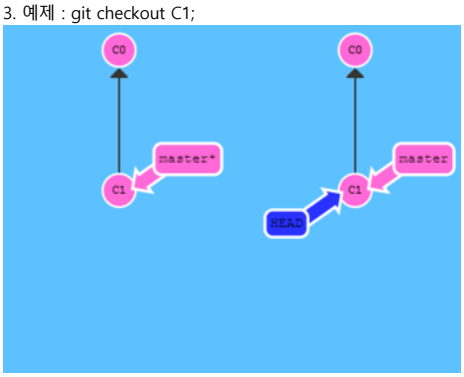
2) git checkout master; git rebase master;



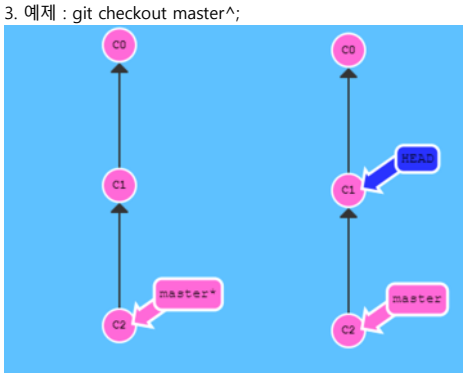
Command - 코드 이리저리 옮기기

2019년 4월 3일 수요일 오후 3:57

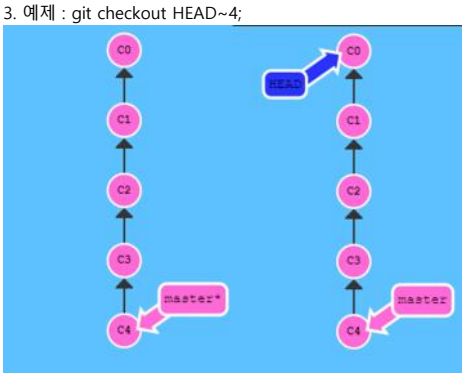
<head : 현재 참조>
1. 의미 : 현재 작업 중인 커밋을 기리킴
2. 형식 : git checkout (work tree name);
3. 예제 : git checkout C1;



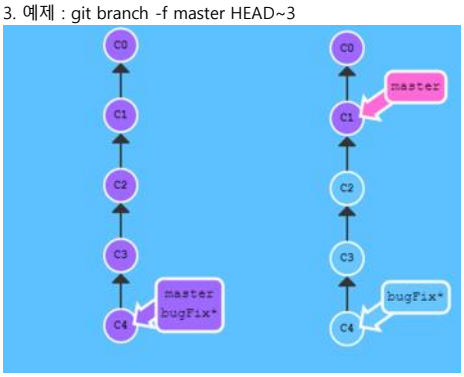
< ^ : 상대 참조 >
1. 의미 : 한 커밋 위로 움직이는 커밋
2. 형식 : git checkout (name)^;
3. 예제 : git checkout master^;



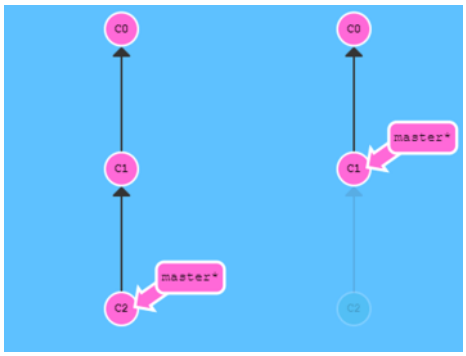
~<num> : 상대 참조
1. 의미 : 다중 커밋 위로 움직이는 커밋
2. 형식 : git checkout (name)~<num>;
3. 예제 : git checkout HEAD~4;



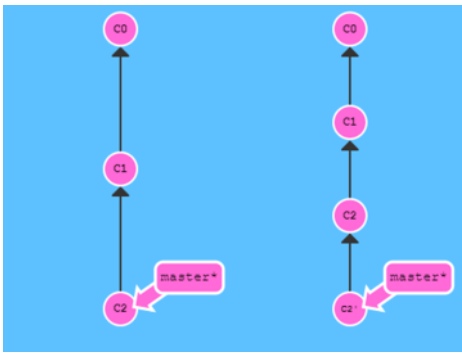
< -f : branch 강제로 옮기기 >
1. 의미 : 특정 커밋에 직접적으로 재지정
2. 형식 : git branch -f (tar_name) (des_name);
3. 예제 : git branch -f master HEAD~3



< reset : 작업 되돌리기 >
1. 의미 : 브랜치로 하여금 예전 커밋을 가리키도록 이동시키는 방식
- 로컬 브랜치 히스토리를 고쳐쓴다.
2. 형식 : git reset (name);



< revert : 작업 되돌리기 >
1. 의미 : 변경분 되돌림
- 리모트 히스토리를 고쳐쓴다.
2. 형식 : git revert (name);

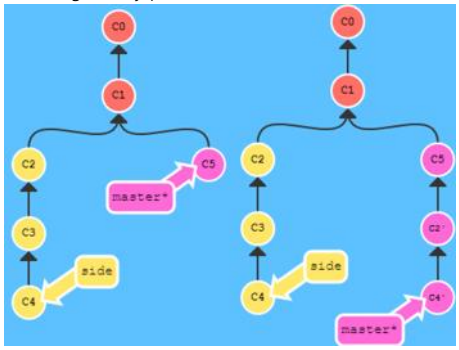


Command - 종합선물세트

2019년 4월 3일 수요일 오후 4:42

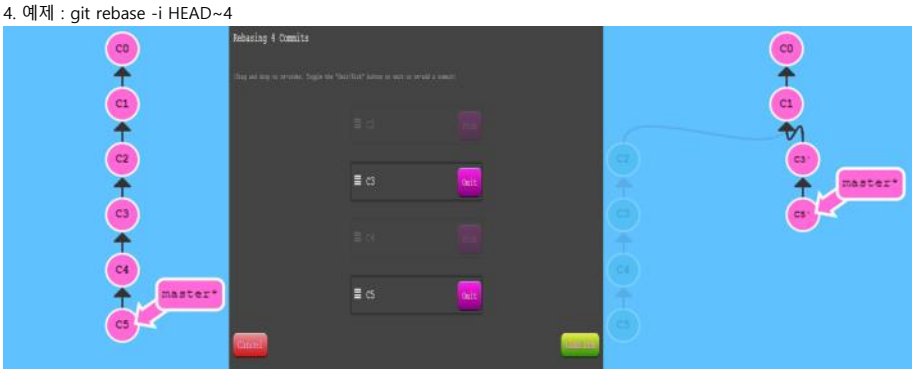
< Cherry-pick >

1. 의미 : 현재 위치 아래에 있는 일련의 커밋들에 대한 복사본을 만듬
2. 형식 : git cherry-pick <Commit1> <Commit2> <...>
3. 예제 : git cherry-pick C2 C4;



< Interactive Rebase >

1. 의미 : 원하는 커밋을 모를 경우 (rebase + -i)
 - 리베이스 목적지가 되는 곳 아래에 복살 될 커밋들을 보여주는 UI를 뛰움
2. 기능
 1) 적용할 커밋들 순서 변경
 2) 원하지 않는 커밋 제거 (pick을 통한 지정)
 3) 커밋 Squash
3. 형식 : git rebase -i <name>
4. 예제 : git rebase -i HEAD~4

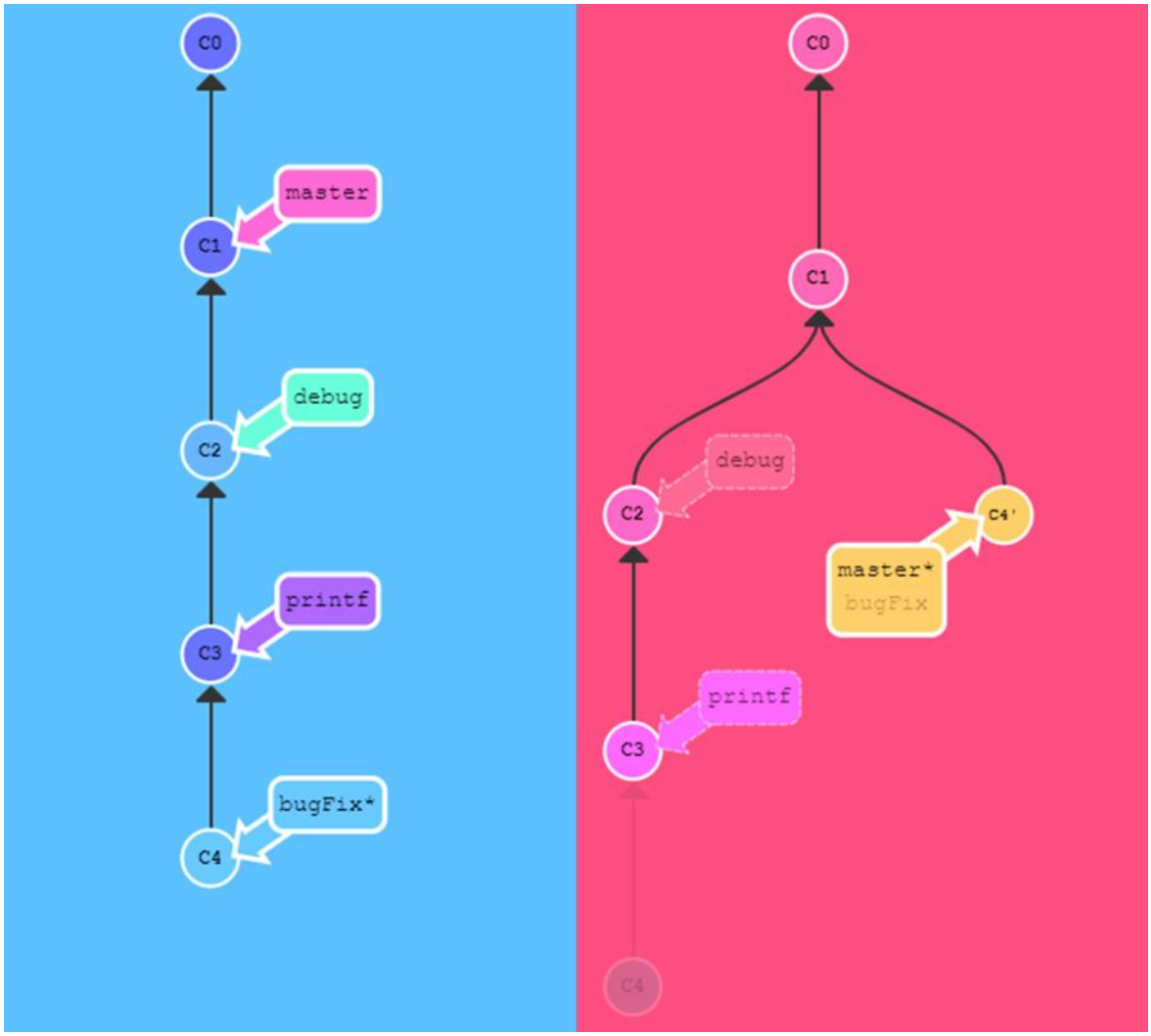


Command - 고급

2019년 4월 3일 수요일 오후 5:13

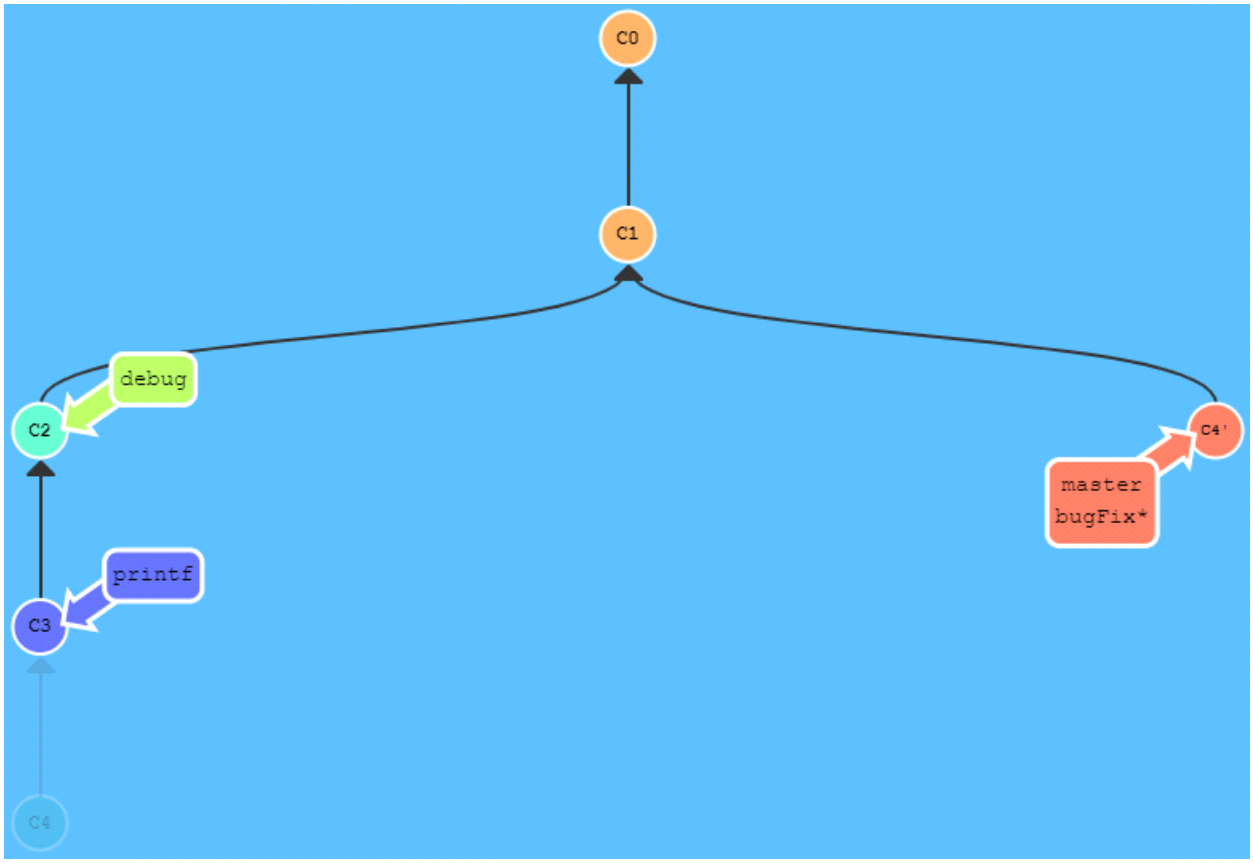
< 딱 하나의 커밋 가져오기 >

1. 목표



2. Interactive rebase

- git rebase -i master;
- git branch -f master bugFix;



3. Cherry-pick

- git checkout master;
- git cherry-pick bugFix;