GIT 기초 관련

GIT 분석에 사용 할 만한 정보들

-사용한 언어

-코드의 줄 개수

-사용한 라이브러리

-branch 및 node 종류 및 개수 & 상태

GIT 저장소 만드는 법

1. 기존 Directory를 Git 저장소로 지정

cd ‘내가 정한 기존 Directory’ -> git init -> .git이라는 하위 디렉토리가 저장소에 필요한 필수요소와 함께 생산됨 & 프로젝트의 어떠한 파일도 관리하지는 않는 상황

git add ‘추가하고자 하는 파일’ -> git commit -> 위에서 만든 저장소에 파일을 추가하고 커밋하게 됨

2. GIT 저장소를 Clone

다른 프로젝트에 참여 및 Contribute하거나 GIT 저장소를 복사하고자 할 때 Clone 명령어를 사용

Clone으로 서버에 있는 거의 모든 데이터를 복사할 수 있고, 프로젝트의 히스토리 또한 다 가져올 수 있다. 물론 서버의 설정을 복구할 순 없다.

ex) git clone <https://github.com/A/A> -> A 저장소 내에 A라는 디렉토리를 만든 다음 해당 디렉토리 내에 .git 디렉토리를 만든다. 이후 A 저장소의 소스코드를 복사하여 디렉토리 A에 저장하게 된다

git clone <https://github.com/A/A> 3A -> 위와 같은 상황이지만 디렉토리 이름을 3A로 지정했다는 차이가 있다

GIT의 파일 관리 종류

committed : 데이터가 로컬 저장소에 안전하게 저장된 상황

modified : 수정한 파일을 아직 로컬 저장소에 commit하지 않은 상황

staged : 현재 수정한 파일을 곧 commit할 것이라고 표시한 상황

GIT의 영역 종류

GIT Directory : GIT이 프로젝트의 메타데이터와 객체 데이터베이스를 저장하는 곳. 일명 .git라고도 불린다.

Working Directory : 수정할 파일들이 있는 Directory

Staging Area : Git Directory 내에 있는 영역이며, 단순한 파일이고 곧

GIT 브랜치의 종류

Master Branch : 제품으로 출시될 수 있는 브랜치. 배포(Release)이력을 관리하기 위해 사용하며, 배포 가능한 상태만을 관리한다고 보면 된다.

Develop Branch : 다음 출시 버전을 개발하는 브랜치. 기능 개발을 위한 브랜치들을 병합하기 위해 사용하며, 모든 기능이 추가되고 배포가 가능할 정도로 안정된 브랜치라면 해당 develop branch를 master branch로 병합한다. 평소 개발은 이 Branch로 진행된다.

Feature Branch : 기능을 개발하는 브랜치. 새로운 기능 개발 및 버그 수정이 필요할 때마다 develop branch로부터 따로 분기된다. Feature branch에서의 작업은 공유 할 필요는 없기 때문에 로컬 저장소에서 관리하기도 하며, 개발이 완료되면 develop branch로 병합하여 다른 개발진들과 공유한다

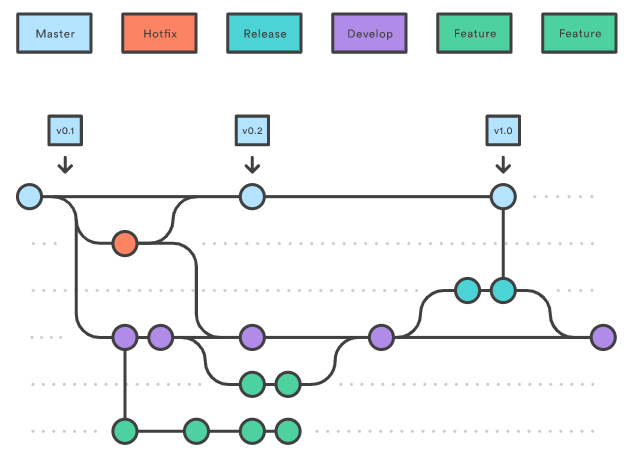
Release Branch : 이번 출시 버전을 준비하는 브랜치. 배포를 위한 전용 브랜치를 사용함으로써 다른 팀은 기능 개발을 계속 진행할 수 있다. 단계적 절차로 이루어진 개발 단계를 정의하기에 좋다.

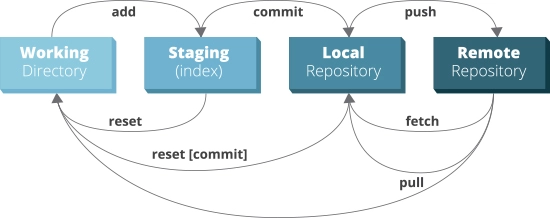
Develop branch에서 배포할 수 있는 수준의 기능이 모이거나 정해진 배포 일정이 되면 release 브랜치를 분기 및 생성한다. 이때서부터 배포 사이클이 시작된다. 해당 브랜치에선 배포를 위한 최종적인 버그 수정, 문서 추가와 같은 release와 관련된 작업들을 진행한다. 그 외의 새로운 기능같은 것은 병합하지 않는다.

Release branch에서 위 과정을 거친 이후 배포 준비가 완료됐다면(새 기능을 포함한 상태로 모든 기능이 잘 돌아간다면), master branch에 병합하게 된다. 이때, 병합한 commit에는 Release버전 태그를 부여해야 구분이 잘 된다. 또한, 배포 준비하는 과정 중 release 브랜치가 변경됐을 수도 있으니 배포 완료 후 develop branch에도 병합한다. 이후 다음 번 release를 위한 작업은 develop branch에서 진행한다.

Hotfix Branch : 출시 버전에서 발생한 버그를 수정하는 브랜치. 배포 버전에 수정해야 할 사안이 있다면 master branch에서 분기하는 브랜치. Develop branch에서 문제가 되는 부분을 수정하고 release할 만한 버전을 만들기엔 시간이 오래 걸리고 안전성도 낮아서 master branch에서 직접 브랜치를 만들어 분기한 다음, 필요한 부분만 수정한 다음 master branch로 다시 병합한다. 또한, hotfix branch에서의 변경 사항은 develop branch에서도 병합한다.

branch 흐름에 대한 상상도





GIT 명령어를 살펴보기 전 대강의 흐름도

다양한 GIT 명령어 관련 설명

git add <파일이름> : 새로운 파일을 추가하거나 존재하는 파일을 스테이징

git add --all : status에 나온 변경사항을 모두 스테이지에 올림

git add . : 위에있는 add --all과 같은 기능

git add -i : add 명령에서 git 대화모드를 사용하여 파일 추가하기

git add -u [<경로>[<경로>]] : 수정되고 추적되는 파일의 변경 사항 스테이징하기

git commit -m “<메시지>” -a : 수정되고 추적되는 모든 파일의 변경사항 커밋하기

git checkout HEAD <파일> [<파일>] : 작업 트리의 변경 사항 돌려놓기

git commit -m “<메시지>” - -amend : 마지막 커밋 고치기

git commit -C HEAD - -amend : 이전 커밋을 수정하고, 커밋 메시지를 재사용하기

git reset HEAD <파일> [<파일>] : 커밋되지 않고 스테이징된 변경 사항 재설정하기

git reset – hard HEAD^ : 커밋한 이전 코드 취소하기

git reset -soft HEAD^ : 코드는 살리고 커밋만 취소하기

git reset – merge : merge 취소하기

GIT 브랜치 관련 명령어 설명

git branch : 지역 브랜치 목록 보기

git branch -r : 원격 브랜치 목록 보기

git branch -a : 지역과 원격을 포함한 모든 브랜치 목록 보기

git branch <새로운 브랜치> : 현재 브랜치에서 새로운 브랜치 생성하기

git branch -d<삭제할 브랜치> : 삭제 할 브랜치가 현재 브랜치에 합쳐졌을 경우에 브랜치 삭제하기

git branch -D<삭제할 브랜치> : 삭제 할 브랜치가 현재 브랜치에 합쳐지지 않았어도 브랜치 삭제하기

git branch <새로운 브랜치> <브랜치 생성위치> : 브랜치를 생성할 다른 지점에서 브랜치 생성하기

git branch -f <기존 브랜치> <브랜치 생성위치> : 기존에 있는 브랜치를 새로운 브랜치로 덮어쓰기

git branch -m <브랜치명> <바꿀 브랜치명> : 브랜치 이름을 바꾸고자 하는 브랜치 이름으로 바꿔버린다

git checkout <기존 브랜치> : 기존 브랜치 checkout하기

git checkout -b <새로운 브랜치> : 현재 브랜치에서 새로운 브랜치를 생성한 다은 checkout하기

git checkout -m <기존 브랜치> <새로운 브랜치> : <새로운 브랜치>가 존재하지 않을 경우, 기존 브랜치를 옮기거나 브랜치명을 변경한다.

git checkout -M <기존 브랜치> <새로운 브랜치> : <새로운 브랜치>의 존재유무 상관없이 무조건 기존 브랜치를 새로운 브랜치로 덮어씌운다.

git merge <브랜치> : 다른 브랜치를 현재 브랜치로 병합한다

git merge - -no-commit <브랜치> : 다른 브랜치를 커밋 없이 현재 브랜치로 병합한다

git cherry-pick <커밋명> : 선택하여 합치기

git cherry-pick -n <커밋명> : 커밋하지 않고 선택하여 합치기

git merge - -squash <브랜치> : 브랜치의 이력을 다른 브랜치에 합치기

GIT 이력 관련 명령어

git log : 모든 이력 보기

git log -p : 변경 사항을 보여주는 패치와 함께 로그 보여주기

git log -1 : 1개의 항목만 보이도록 로그 개수 제한하기

git log -20 -p : 20개의 항목과 패치만 보이도록 로그 제한하기

git log - -since=”6 months” : 6개월 동안의 커밋 로그 보기

git log - -before=”2 days” : 이틀 전까지의 커밋 로그 보기

git log -1 HEAD-3 또는 git log -1 HEAD^^^ 또는 git log -1 HEAD~1^^ : HEAD보다 3개 이전의 커밋 로그 보기

git log <시작 지점> … <끝 지점> : 두 지점 사이의 커밋 로그 보기. 시작 지점이나 끝 지점은 커밋명, 브랜치명 혹은 태그명이 될 수 있고 조합하여 사용 가능하다.

git log - -pretty=oneline : 각 항목의 로그 이력 한 줄씩 보기

git log - -stat : 각 항목마다 영향 받은 줄의 통계 보기

git log - -name-status : 커밋할 시점의 파일 상태 보기

git diff : 현재 작업 트리와 인덱스의 차이점 보기

git diff - -cached : 인덱스와 저장소의 차이점 보기

git diff HEAD : 현재 작업 트리와 저장소의 차이점 보기

git diff <시작지점> : 현재 작업 트리와 특정 위치 간의 차이점 보기. 이때 시작 지점은 커밋명 or 브랜치명 or 태그명이다

git diff <시작지점> <끝 지점> : 저장소와 두 지점 사이의 차이점 보기

git diff - -<시작지점> [<끝 지점>] : 차이점의 통계 보기

git blame <파일> : 파일의 커밋 정보 줄 단위로 보기

git blame -M<파일> : 파일의 줄 단위의 복사, 붙여 넣기, 이동 정보 보기

git blame -C -C <파일> : 파일의 줄 단위의 이동과 원본 파일 정보 보기

git log -C -C -p -1 <특정 지점> : 로그에서 복사와 붙여 넣은 정보 보기

GIT 원격 저장소 관련 명령어 설명

git clone <저장소> : 저장소 복제하기

git clone - -depth 200 <저장소> : 마지막 200개의 커밋만 포함하여 저장소 복제하기

git remote add <원격 저장소> <저장소 url> : 새로운 원격저장소 추가하기

git branch -r : 모든 원격 브랜치 목록 보기

git branch <새로운 브랜치> <원격 브랜치> : 원격 브랜치에서 지역 브랜치 생성하기

git branch <새로운 브랜치> <원격 태그> : 원격 태그에서 지역 브랜치 생성하기

git fetch : origin 저장소에서 합치지 않고 지역 브랜치로 변경 사항 가져오기

git fetch <원격 저장소> : 원격 저장소에서 합치지 않고 지역 브랜치로 변경 사항 가져오기

git pull <원격 저장소> : 원격 저장소에서 변경 사항을 가져와 현재 브랜치로 합치기

git pull : origin 저장소에서 변경 사항을 가져와 현재 브랜치에 합치기

git push <원격 저장소> <지역 브랜치>:<원격 브랜치> : 지역 브랜치를 원격 브랜치에 푸싱하기

git push <원격 저장소> <지역 브랜치> : 지역 브랜치를 동일한 이름의 원격 브랜치에 푸싱하기 or 새로운 로컬 브랜치를 원격 저장소에 푸싱하기

git remote prune <원격 저장소> : 원격 저장소에서 쓸모가 없어진 원격 브랜치 제거하기

git remote rm <원격 저장소> : 원격 저장소를 제거하고 관련된 브랜치도 제거하기

자료 출처

https://medium.com/@joongwon/git-git-%EB%AA%85%EB%A0%B9%EC%96%B4-%EC%A0%95%EB%A6%AC-c25b421ecdbd