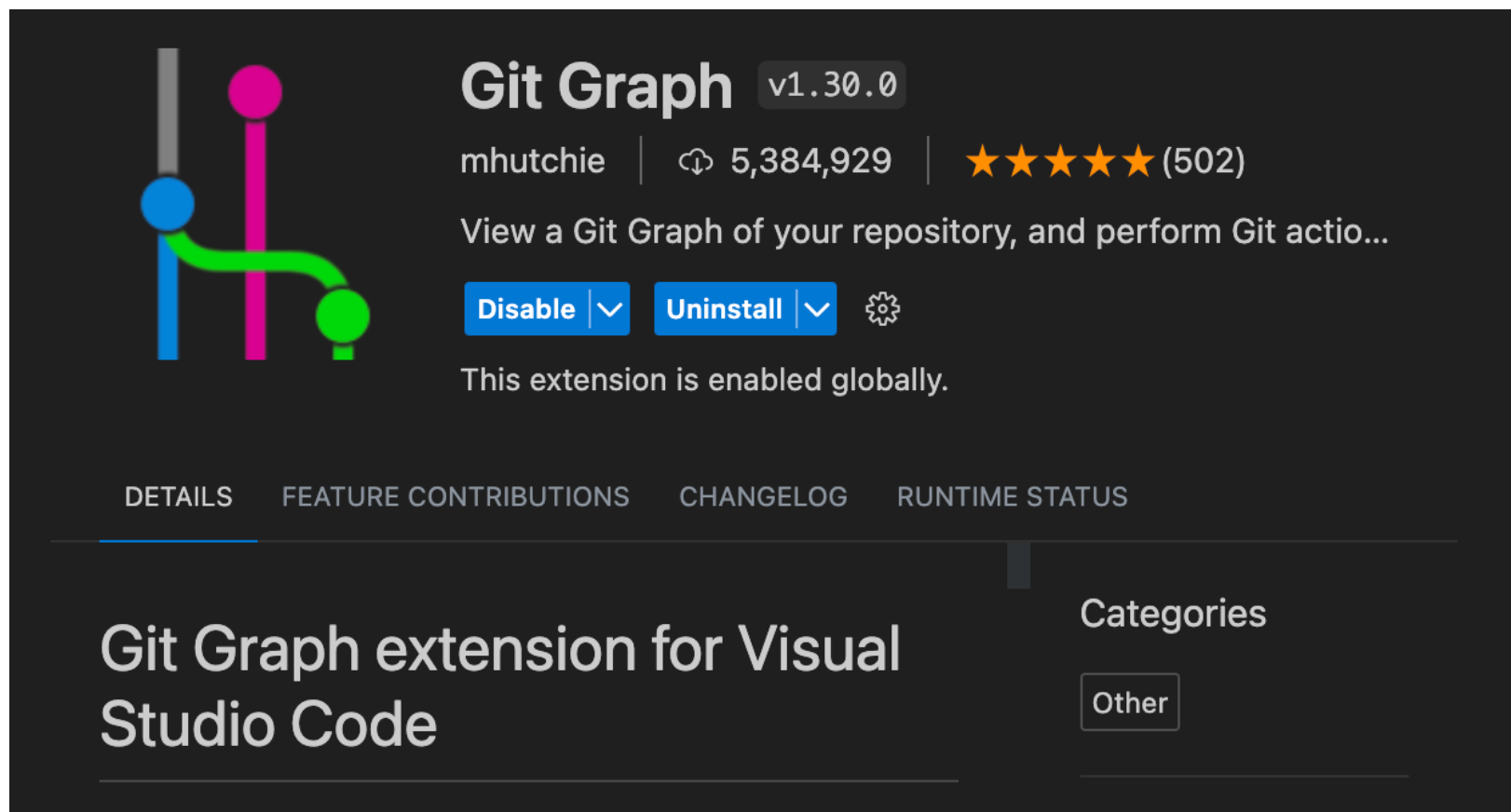


좋은 말씀 전합니다.



The image shows the Git Graph extension interface in Visual Studio Code. On the left is a logo with a blue vertical line, a pink vertical line, and a green curved line connecting a blue circle to a pink circle. To the right of the logo, the text 'Git Graph' is displayed in a large font, followed by the version 'v1.30.0' in a smaller font. Below this, the author 'mhutchie' is listed, followed by a cloud icon and the number '5,384,929'. To the right of this is a row of five yellow stars and the text '(502)'. Below the stars is a description: 'View a Git Graph of your repository, and perform Git actio...'. Under the description are two buttons: 'Disable' and 'Uninstall', both with a dropdown arrow, and a gear icon for settings. Below these buttons is the text 'This extension is enabled globally.'.

Git Graph v1.30.0

mhutchie | 5,384,929 | ★★★★★ (502)

View a Git Graph of your repository, and perform Git actio...

[Disable](#) | [Uninstall](#) | ⚙️

This extension is enabled globally.

DETAILS | FEATURE CONTRIBUTIONS | CHANGELOG | RUNTIME STATUS

Git Graph extension for Visual Studio Code

Categories

[Other](#)

어머! 저건 붙여야해

- TAG
 - 기념할 만한 무엇인가에 붙일 수 있는
깃의 배려



깃 허브에서 붙여 볼까요?

1. 커밋 히스토리를 봅니다.
2. 메인 커밋 바로 뒤 시점의 저장소를 browse 해 봅니다.
3. Create a new release
4. Choose a tag에서 Find or create a new tag 입력 후 create new tag
5. Publish release
→그러면 뭔가 좀 과한 일이 일어납니다.
6. 이 녀석을 pull해서, 태그를 선택하고 수정 하고 난 후에 커밋을 하면...

깃 허브에서 붙여 볼까요?

7. 뭔가 부모 없는 자식이 생겨 버렸습니다. 이 녀석을 어떻게 해보죠.

8. vs에서... 임시 브랜치를 선택하면 위에서 브랜치를 만들라고 합니다.

9. 하나 만들었어요.

10. 이 시점에서 git graph를 볼까요?

결론. tag는 브랜치와 다릅니다. 우리는 사실 하나의 선형 커밋을 하고 있었던 겁니다.

조금 더 가 봅시다.

- 11. 선형 커밋이긴 한데, 메인이 가장 최신 브랜치는 아닙니다.
- 12. 이때 병합을 하면 되지요.

요약하자면, 태그에서 커밋하면 태그를 만든 브랜치의 어떤 선상에 갈 곳 잃은 브랜치 상태가 됩니다.

(참고로, 이 상황은 tag를 checkout한 후에 하는 커밋에 해당됩니다.)

이 상태에서 브랜치를 만들고, 마스터에서 병합을 하면 적당한 답이 나옵니다.

이번엔 vs code에서 붙여 보지요.

Source Control에서 ... 에서 Tags > Create Tags (v1.1.0)
과거 커밋에 붙이고 싶을 땐 extention을 이용합니다.

이제 이렇게 태그를 붙였으면, 리모트에서도 공유를 하고
싶겠지요.

그런데, sync 알림도 없고, 그렇다고 깃헙에 가서 봐도 태그가
없습니다. 태그는 push가 되지 않는 녀석이었던 겁니다.

이번에도 extension을 이용 합시다.

Console에서 붙이려면

- `git tag -a tagname -m "message"` (anotated tag)
- `git tag tagname` (lightweight tag)
- 과거 버전에 붙이려면
- `git tag tagname commitid`
- 일단, (push하기 전인) 가장 최신 버전에 v1.1.1 태그를 붙여 봅니다. (lw)
- 그리고 1.2.0 1.3.0, ... 마구 붙입니다. 하나의 커밋에 여러 개의 태그가 붙을 수 있다는 사실을 알게 됩니다.
- 마지막으로, 로컬에서 tag를 push 하려면, `git push 저장소이름 tagname` 또는 `git push 저장소이름 --tags` (모두)

먹다 남아 버렸는데, 이걸 어떻게...

- stash
 - 일단 저장하고 다른 브랜치로 넘어 갑니다.
 - 무엇을?
 - Modified이면서 tracked인 대상과
 - Staging 된 대상을



설마

- 우리 문화인들은 github 웹에서 stash 어떻게 하냐고 묻지 맙시다.
- untracked 파일은 어떻게 되냐고도 묻지 맙시다.
 - 사실은 -u 옵션을 붙이면 가능해요
- 그리고 stash는 브랜치랑 무관한 것도 아시지요?

먹다 남은 것을 킵 해 둡시다. git stash

- 어느 브랜치건 파일 하나 추가한 후
- git stash
- 브랜치를 바꾸고난 후에
- git stash apply
- 다시 git stash
- 다시 파일 생성, git add .
- 다시 git stash
- stash 목록을 보려면? git stash list
 - 어라 apply를 했는데도, 여전히 목록이 남아 있네.

먹다 남은 것을 킵 해 둡시다. git stash

- 어느 브랜치건 파일 하나 추가한 후
- git stash
- 브랜치를 바꾸고난 후에
- git stash apply
- 다시 git stash
- 다시 파일 생성, git add .
- 다시 git stash,
- stash 목록을 보려면? git stash list
 - 어라 apply를 했는데도, 여전히 목록이 남아 있네.

먹다 남은 것을 킵 해 둡시다. git stash

- 어느 브랜치건 파일 하나 추가한 후
- git stash
- 브랜치를 바꾸고난 후에
- git stash apply
- 다시 git stash
- 다시 파일 생성, git add .
- 다시 git stash,
- stash 목록을 보려면? git stash list
 - 어라 apply를 했는데도, 여전히 목록이 남아 있네.

먹다 남은 것을 킵 해 둡시다. git stash

- 그래서 목록을 지워줘야 합니다.
 - git stash drop
 - git stash pop (적용하면서 삭제)
- 그리고 선택적으로 stash를 하고 싶을 때는
 - git stash -patch
- shash가 있을 때
 - git stash branch branchname → 새 브랜치를 만들고 거기에 stash apply를 하고, stash 목록을 삭제
- vs code에서는 source control에서 추적 창에서 선택적으로 stash 가능

종종 보는 에러인데...

- 나는 그저 push를 하려고 pull을 했을 뿐인데...

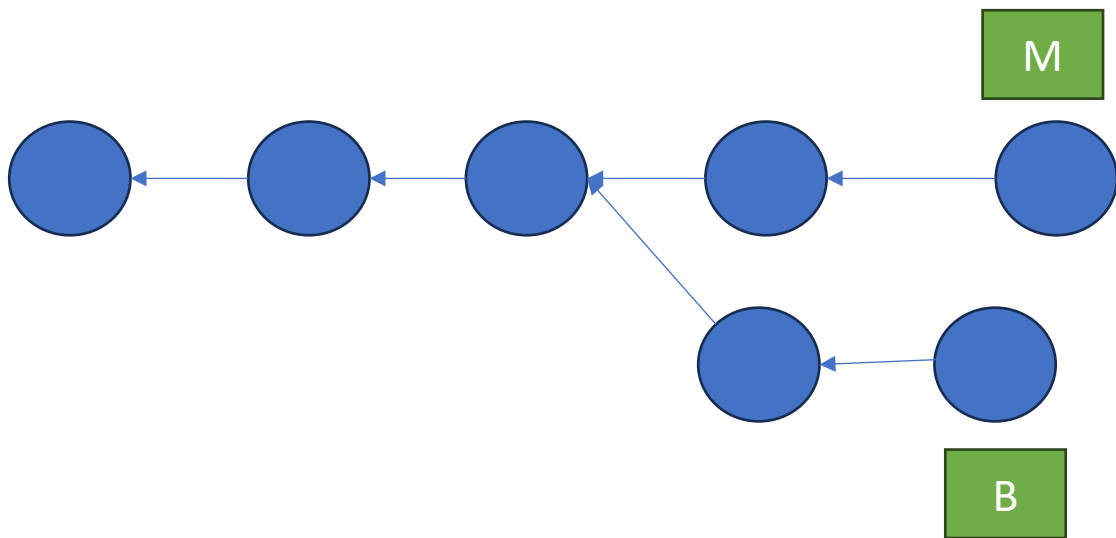
```
git config pull.rebase false # merge
git config pull.rebase true  # rebase
git config pull.ff only      # fast-forward only
```

- 그럼 rebase가 뭔지 알아야 겠습니다.

리베이스

- 시나리오

- master에서 브랜치를 만들고, 마스터에서 수정을 하고 커밋을 했음
- 그 후에 브랜치에서 수정 커밋 (모든게 충돌이 없도록)



b1	b_2	11 Jul 2023...	crapas	05e07f27
b_1		11 Jul 2023...	crapas	9323c303
master	m_2	11 Jul 2023...	crapas	3b63bb7b
m_1		11 Jul 2023...	crapas	f10c8ec9
5th		11 Jul 2023...	crapas	1fcdcf6d

리베이스

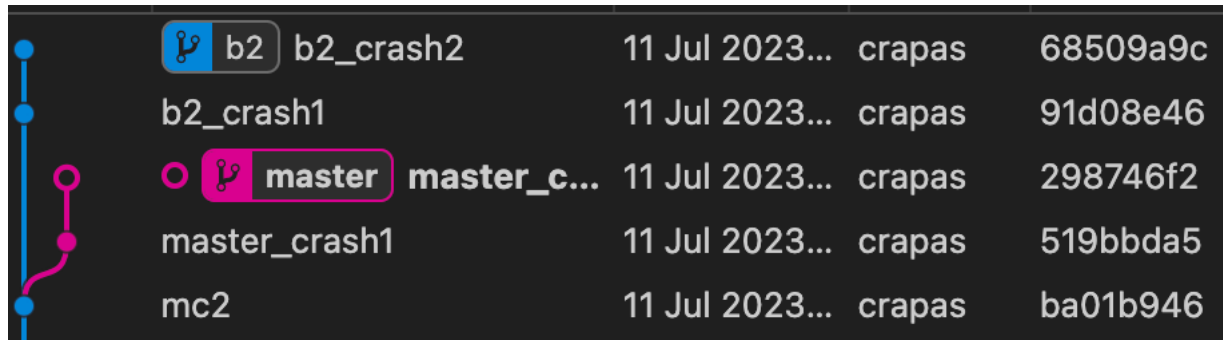
- 패치!패치!
 - (in master) git rebase b1



병합의 경우



충돌이 날 때 경우



b2	b2_crash2	11 Jul 2023...	crapas	68509a9c
	b2_crash1	11 Jul 2023...	crapas	91d08e46
master	master_crash1	11 Jul 2023...	crapas	519bbda5
	mc2	11 Jul 2023...	crapas	ba01b946

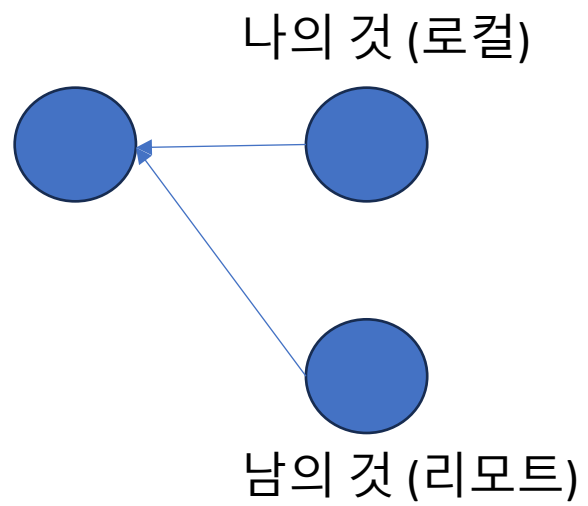
- master의 각각의 commit에 대해서 브랜치의 최종 커밋의 '패치'를 적용



master	master_crash2	11 Jul 2023...	crapas	e0c8190d
	master_crash1	11 Jul 2023...	crapas	cf1f0a2e
b2	b2_crash2	11 Jul 2023...	crapas	68509a9c
	b2_crash1	11 Jul 2023...	crapas	91d08e46
	mc2	11 Jul 2023...	crapas	ba01b946

revisit

- git pull : git fetch + git merge
- 언제 발생하는가?
 - fast-forward merge가 안되는 경우
 - 이 경우에 merge냐 rebase냐의 전략을 옵션으로 설정해줘야 함
- 어떻게 해결할까?
 - merge로 선택
 - rebase로 선택
 - ff-only를 유지하고 안되면 fetch 후 확인하고 필요에 따라 병합
 - 안전하고
 - 리뷰를 거침



기타

- remote에서 만들어진 branch 정보는 git fetch를 한 다음에 알 수 있습니다.

나만 보고 싶은 좋은 것들이 있어요

- .gitignore

개념

- 저장소가 자동으로 추적하지 말아야 하는 것들을 지정합니다.
 - 빌드 및 컴파일 관련 파일 (목적 파일, 실행 파일, 중간 산출물들)
 - 캐시 파일
 - 로그 파일
 - 개인 정보
 - 스왑 성격의 파일

어디에서 무시

- 디렉토리에

- 일단 작업 디렉토리의 최 상위에 .gitignore 파일을 생성해 봅시다.
- 그 다음, 문서에 *.swp 이라고 입력하고 저장해 봅시다. 그 순간 부터, 모든 .swp으로 끝나는 파일이 git에서 관리 되지 않습니다. (디렉토리 무관)
- 이번에는 하위 디렉토리를 생성하고, 그 안에 .swp 파일을 생성해 봅시다. .gitignore 파일의 영향은 하위 디렉토리에도 미친다는 것을 알 수 있습니다.
- 이번에는 .gitignore 파일을 하위 디렉토리로 옮겨 봅시다. 이 실험을 통해 상위 디렉토리에는 영향을 미치지 않는다는 사실도 알 수 있습니다.
 - 정화히는 상위디렉토리의 존재를 모른다고 봐야 합니다. 즉 하위 디렉토리에서 / 디렉토리의 의미는 작업 디렉토리 최상위가 아니라 현재 디렉토리를 의미합니다.
- 디렉토리를 명시해주고 파일을 주면 정확히 그 디렉토리만 영향을
 - 디렉토리 이름에도 와일드카드 사용 가능하지만 하위 디렉토리는 영향이 없음

하지만 무시하기에 무서운 녀석은

- 하지만 무시할 수 없는 녀석들은 예외로
 - 다시 상위 디렉토리로 .gitignore를 옮긴 후 파일의 두 번 째 줄에 !a.swp을 추가합니다. 지정한 패턴의 예외를 지정할 수 있음을 알 수 있습니다.
 - 두 줄의 순서를 바꿔 봅니다. 뒤의 규칙이 앞의 규칙을 덮어 씌울 수 있습니다.
 - 이제 하위 디렉토리에 .gitignore를 추가하고 상위 디렉토리의 무시 파일에 *.swp을, 하위 디렉토리는 아무 조건이나 추가합니다.
 - 하위 디렉토리에 .gitignore 파일이 있어도 충돌이 나지 않으면 상위 디렉토리의 조건이 캐스캐이딩 됨을 알 수 있습니다.
 - 이제, 여기서 하위 디렉토리에 !*.swp을 추가해 보면 상-하위 디렉토리의 규칙에 충돌이 발생하면 하위 디렉토리의 규칙이 우선함을 알 수 있습니다.
 - 그리고 git add -f a.swp을 해 보면 무시를 무시할 수 있어요.

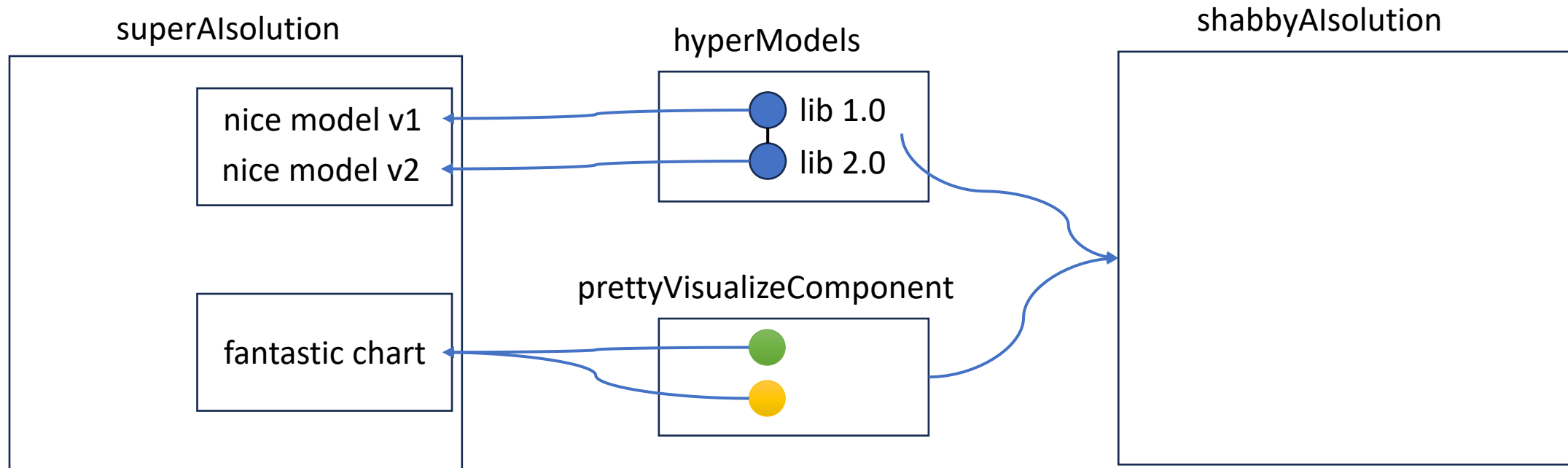
조금 더 자세하게 무시

- 파일 작성 기본 규칙
 - Glob 패턴 사용
 - 주석은 줄의 첫 글자를 #로
 - /로 시작하면 하위 디렉토리에 적용되지 않음
 - 디렉토리를 대상으로 하려면 마지막에 /를 꼭 붙여주기
 - !는 무시하지 않음
- 한번 관심 받은 파일은
 - 계속 관심을 받습니다. 추후에 무시를 하라고 누가 시켜도
 - 그러므로 일단 추적된 파일을 무시하고 싶을 때는
 - `git rm --cached filename`
 - 으로 강제 무시가 가능합니다.

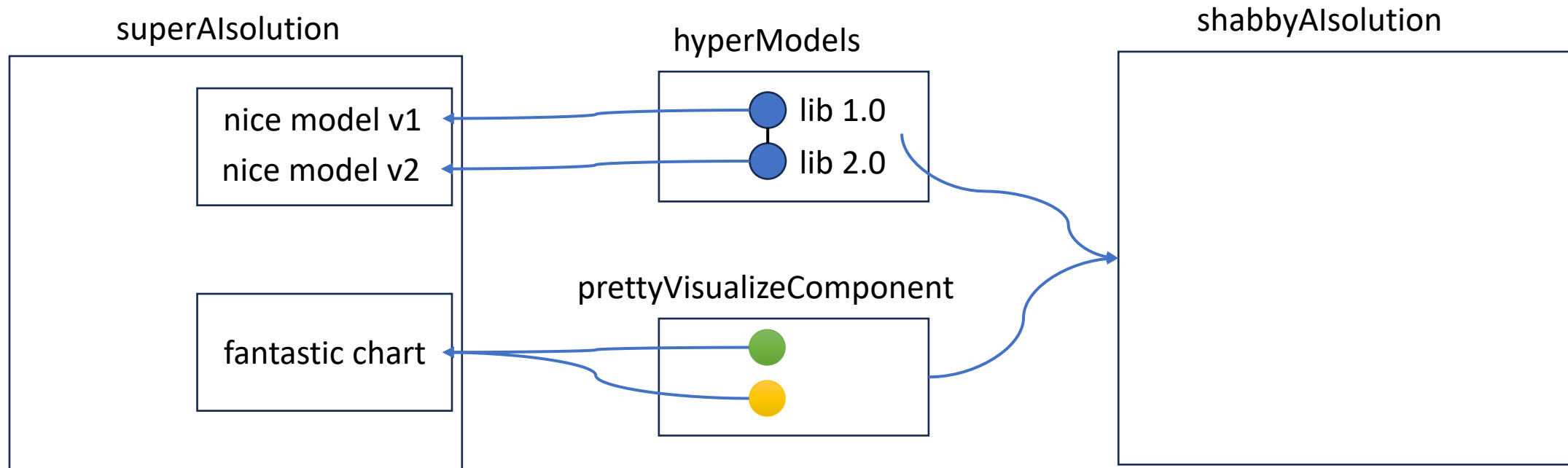
패턴 일람

- 알아두면 좋은 glob 패턴
 - 와일드카드 허용 (*, ?)
 - [abc] → a, b, c : ex) *.py[co] → *.pyc, *.pyo
 - [a-c] → a, b, c
 - [0-9] → 0, 1, 2, ..., 9, [1-0] → match for nothing
- 언어별 패턴
 - <https://github.com/github/gitignore/>

나는 너를 품고 싶지만 그러기에는 너는...



submodule



hyperModels 와 prettyVisualizeComponent 저장소를 복제해서 넣는다?

설정극

- 3개의 저장소를 만듭니다.
 - sAls, hm, pvc
 - sAls의 lib 폴더 밑에 hm이, components 폴더 밑에 pvc를 가져다 놓기
- 어떻게?
 - src로 가져다 쓰는 방법
 - clone해서 저장소를 가져다 쓰는 방법 → submodule
 - sAls에서
 - git submodule add <https://github.com/.../hm.git> lib
 - git submodule add <https://github.com/..../pvc.git> components
 - 그런데 잠깐... 일단 설정극이니 hm과 pvc에 뭔가 좀 커밋한 후 clone해요
 - v1 in hm, green in pvc

submodule의 상태, 그리고 준비 완료

- sAls/lib의 git log와 hm의 git log를 비교
- sAls에서 git status
 - .gitmodules, lib, components의 스테이징
 - lib, components의 스테이징 내용은? (git diff --cached 또는 vs code에서...)
- sAls commit
 - ready to use submodules!

시간이 흘러서...

- hm에 v2가 출시 되었습니다.
 - v1을 v2로 바꾸고 커밋
 - tag도 답시다. 첫 번째 커밋은 v1.0, 두 번째 커밋은 v2.0
 - `git tag -a v1.0 commitid -m v1.0`, `git push origin v1.0`, ...
- pvc에 yellow가 출시 되었습니다.
 - pvc에 yellow 추가하고 커밋
 - 첫 번째 커밋은 v1.0, 두 번째 커밋은 v2.0
- sAIs는 습관적으로 submodule을 확인합니다.
 - `git submodule update --remote`
 - 화들짝 놀랍니다. 우리는 v2.0들에 대한 준비가 되어 있지 않은데

돌아가 봅시다.

- git submodule update
 - --remote : remote의 최신
 - 없으면 : local의 최신
 - 그래서 실수로라도 커밋을 하면 안되지만...
- 이제, yellow에 대한 준비가 되었다면...
 - component로 들어가서 git checkout v2.0
 - 그리고 sAIs로 돌아와서 commit
 - 이 상태에서 깃허브를 한번 가 볼까요?

시간이 더 흘러서...

- red가 나왔는데 버전은 그대로...
 - pvc에서 red 추가, 커밋, 푸쉬
- 습관적으로 `git submodule update --remote` 를 해 보던 개발자는
 - 또 화들짝 놀라는데, 이번엔 금방 냉정을 되찾고 `git submodule update`
 - 그리고 component로 들어가서 `git pull`을 하려고 해요. red를 살펴봐야 하니... 그런데...
 - 우리는 지난 번 `git checkout tag`를 했기 때문에 pull을 받을 브랜치에 있지 않아요. 그래서...
 - `git checkout -b branch_v2`를 한 다음 `git pull`
- 그리고 red가 예뻐보이니, red를 후딱 지원하기로 했습니다.
 - 그래서 commit, 그리고 내부에서는 tag를 찍고 씁니다.
 - (sAls)에서 `git commit`, `git tag`

흥에 겨운 오바질

- component 폴더에서 태그를 찍고
- 심지어 requirement.txt 라는 파일도 추가 해 버립니다.
- 화가 난 pvc 개발자들이
 - 일단 파일을 지우고 태그도 날려 버립니다.
 - 그리고복수를 하겠노라고. v2.0도 지워 버립니다.
 - 태그 지움을 푸쉬 하는 방법 : git push origin :v2.0
- 하지만 sAIs는 뭐.. 그다지 복수가 되지 않습니다.