

Git

# Git 사용하기 전 준비사항

- ▶ Git을 download 해서 설치 (Windows 기준)
  - <https://git-scm.com/downloads>
  - Linux 환경으로 git bash
  - Windows의 powershell, cmd에서 git 명령 사용 가능
    - 불가능하면, path 환경변수가 Git 실행 directory 포함시킴
    - vs code에서도 사용 가능하게 된다.
- ▶ 설치 후 가장 먼저 할 일 (git bash 터미널에서)
  - User name, mail 설정 - 앞으로 만들 모든 repository에 적용됨
    - \$ git config --global user.name "Jin Pyo Hong"
    - \$ git config --global user.email jinpyohong@gmail.com
  - CRLF - LF 간 변환 설정
    - Line ending 변환이 자동인가 확인
      - \$ git config -global core.autocrlf
    - Windows에서는 true로 설정 (Linux/OSX: input)
      - \$ git config -global core.autocrlf true
- ▶ 원격 저장소 확보: GitHub 가입: <https://github.com>

# 저장소 만들기

## ▶ GitHub에 remote 저장소 만들기

- .gitignore file을 반드시 만들자 (version 관리 대상 제외 지정)
- README.md 파일도 권장

## ▶ Local 저장소 만들기

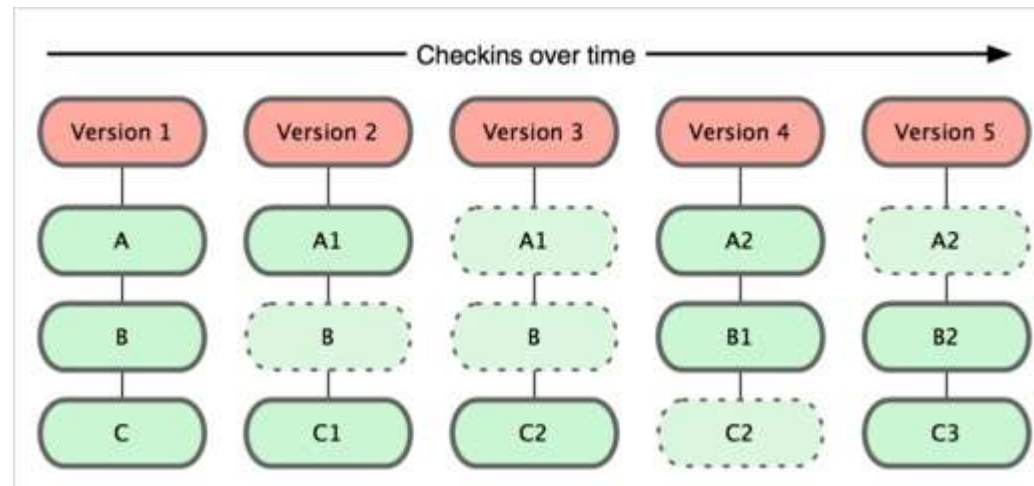
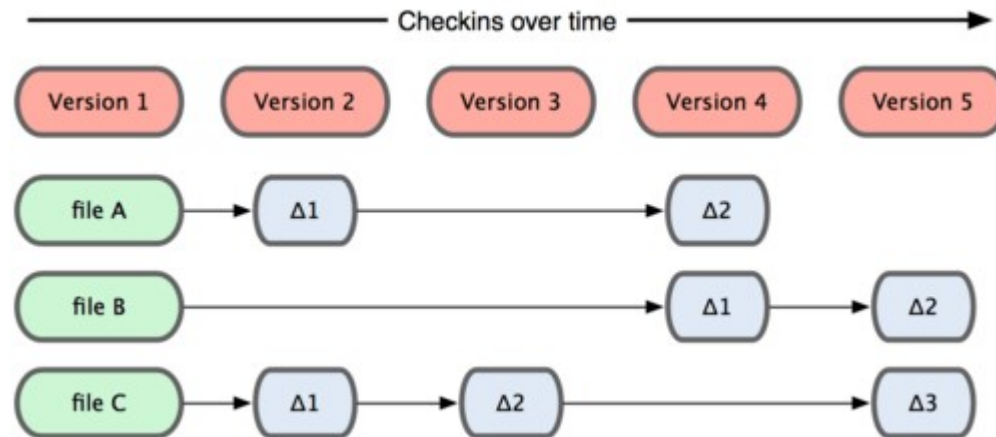
- 기존 (remote) 저장소를 복제한다

```
$ git clone <repository url>
```

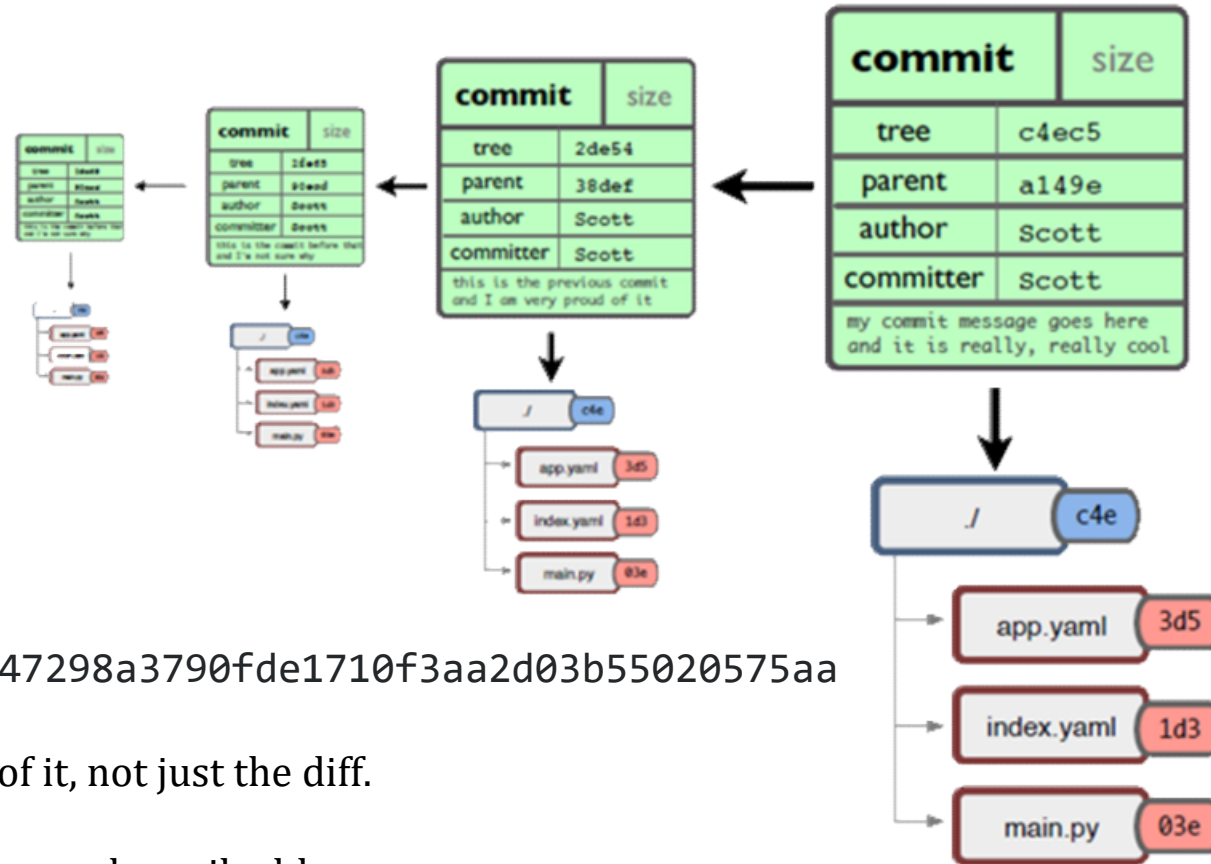
### ❖참고: 기존 project file을 버전 관리하기

- git init 으로 저장소 만들고 remote 저장소와 연결할 수 있으나
- GitHub에 일단 저장소를 만든 후 clone함의 편리하다.

# What is Version Control?



# Git stores data as snapshots of the project over time

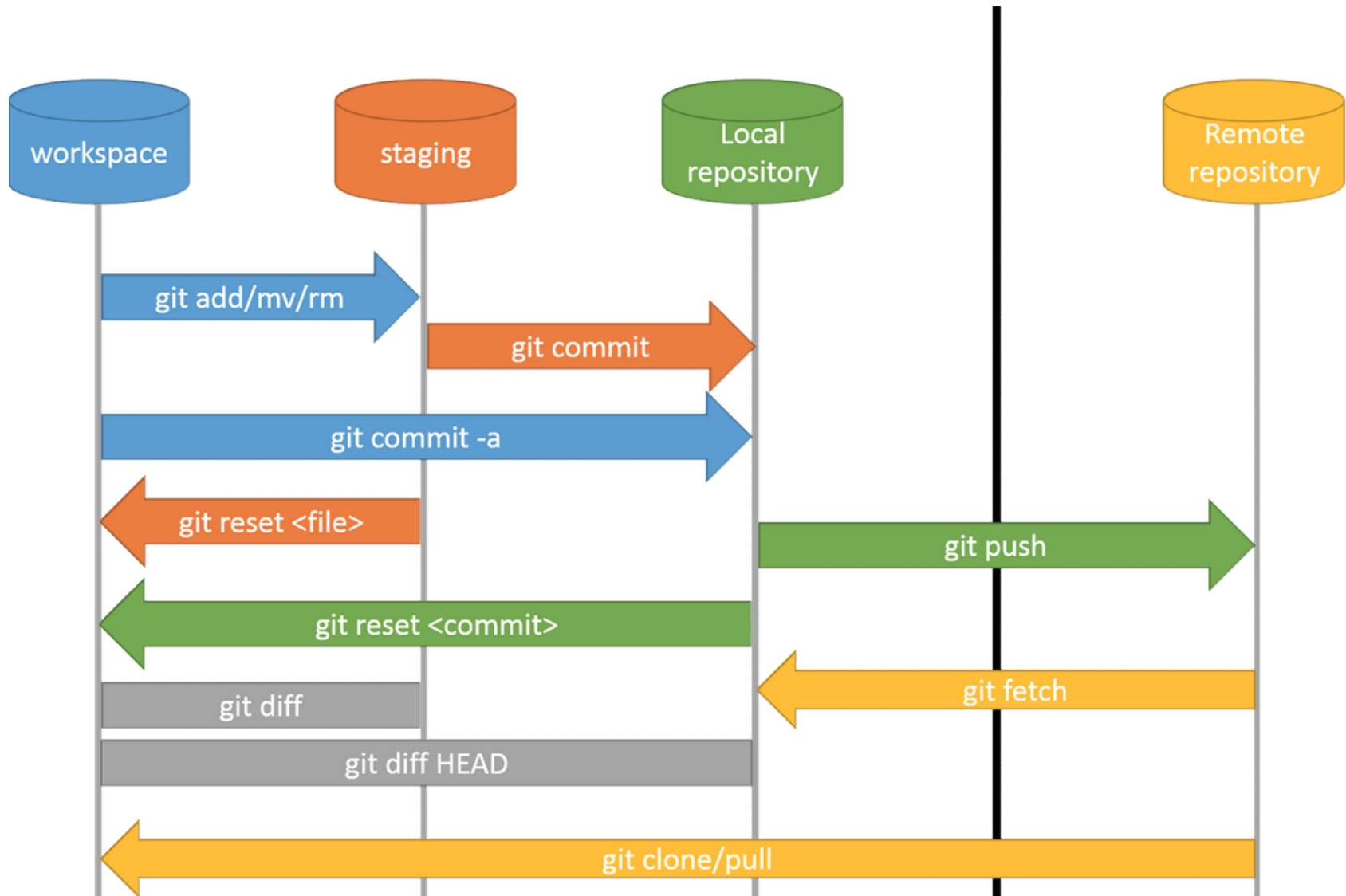


Commit ID: 521747298a3790fde1710f3aa2d03b55020575aa  
SHA-1 hash:

- The content, all of it, not just the diff.
- Commit date.
- Committer's name and email address.
- Log message.
- The ID of the previous commit(s).

commit 내용이 다르면 ID도 다르다.

# Working with Git



# Fetch, Pull, and Push

## ▶ Fetch

- remote 저장소의 branch 에서 local 저장소의 branch에 없는 commit 을 가져와 저장한다. Local 저장소에서 '<remote>/<branch>'

## ▶ Pull

1. Fetch remote branch (예: origin/master)
2. Merge the fetched remote branch into local tracking branch (master)

## ▶ Push

- Local branch 를 remote 저장소의 branch에 추가 merge(fast-forward)한다.
- 3-way merge가 필요하다면 push는 fail된다.

## ❖ Push 하기 전에 먼저 pull 하라.

- 작업 기간에 동료가 새로운 commit을 push했을 수 있으니, 최근 것 가져와서 local branch와 merge 한 다음 push해야 한다. (fast-forward 가능해야)
  - conflict 발생 가능. 나중에 push할 사람이 해결할 책임이 있다.
- 주기적으로 fetch 하는 습관 갖자. (vscode에서 AutoFetch를 true로 setting) 진행 history 추적하고, 반영함(merge)

# Git Branching

Pro Git, 2<sup>nd</sup> Ed.

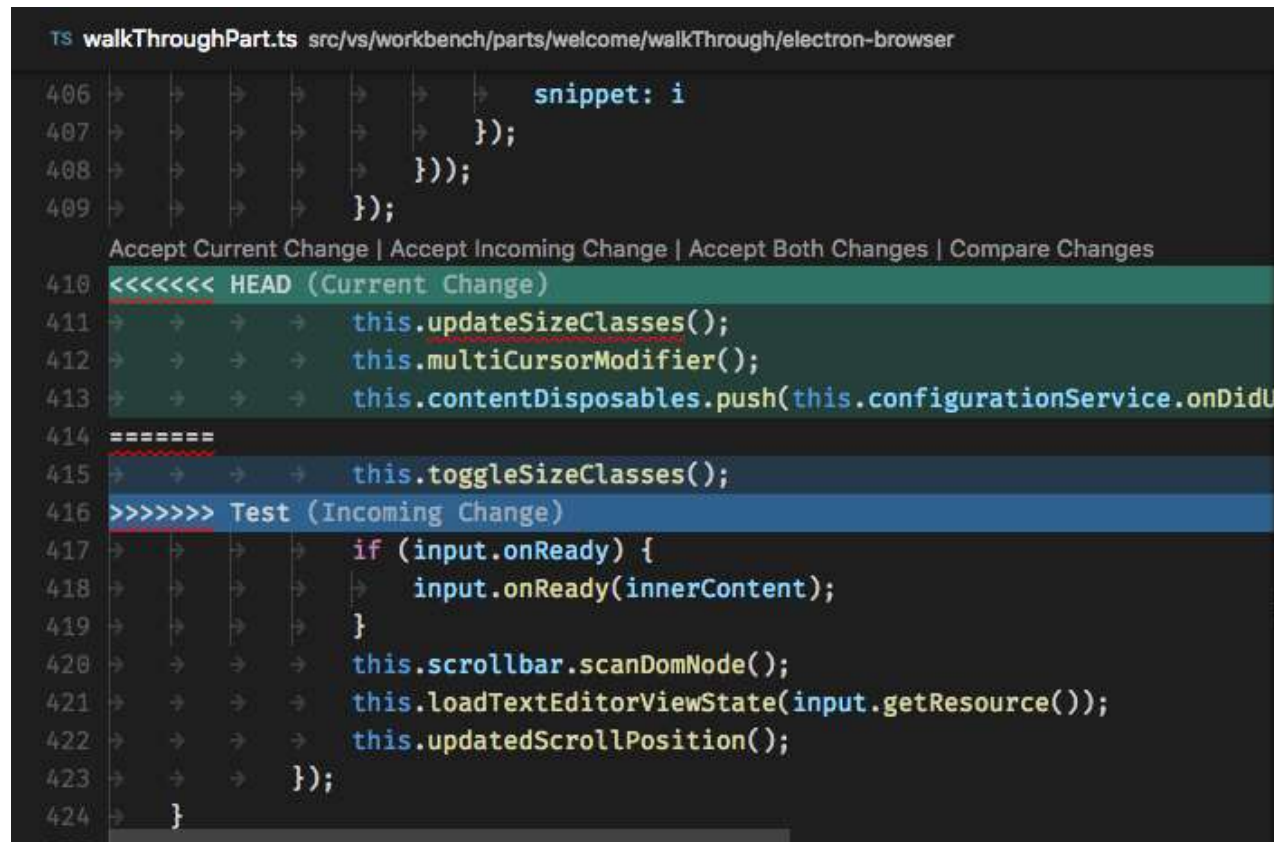
<https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>



# Branching

- ▶ Branch란?
  - Commit history는 commit들의 linked list.
  - Branch name은 commit을 refer한다.
  - master에는 중요한 commit을 등록
    - master와 동기화할 remote 저장소의 master에는 내 것, 동료 것이 들어 있어 team에서 공유할 중요 결과물이다.
    - local 저장소의 master는 remote 저장소의 master branch를 tracking하고 동기화한다.(pull/push)
    - origin/master는 local 저장소에 있는 특별한 branch name 이다. 다만, remote 저장소에서 fetch할 때 이동한다.
    - HEAD는 current branch를 가리킨다.
- ▶ Branching: 가능한 feature 개발, issue 처리, hotfix는 새로운 branch를 만들어 작업하라
  - Checkout: branch 이동. working directory 내용도 새로 바뀐다.
  - Merge: 끝나면 상위 branch(예: master)로 merge하라.
  - Branch 공개/비공개: push하면 동료와 공유할 수 있고, 안 하면 private
- ▶ Merge vs Rebase
  - Rebase는 history를 선형으로 관리 간편하게 만든다.
  - Rebase하려면 local branch에서 하라.
  - Push한(공개한) commit들을 rebase하지 말라. Commit들이 변경되니까

# Merge Conflict



The screenshot shows a code editor with a file named `walkThroughPart.ts` located at `src/vs/workbench/parts/welcome/walkThrough/electron-browser`. The code is in TypeScript and shows a merge conflict. The current state of the file is as follows:

```
406 → → → → → → → snippet: i
407 → → → → → → → });
408 → → → → → → → });
409 → → → → → → → });

Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
410 <<<<<< HEAD (Current Change)
411 → → → → → → → this.updateSizeClasses();
412 → → → → → → → this.multiCursorModifier();
413 → → → → → → → this.contentDisposables.push(this.configurationService.onDidU
414 =====
415 → → → → → → → this.toggleSizeClasses();
416 >>>>>> Test (Incoming Change)
417 → → → → → → → if (input.onReady) {
418 → → → → → → → → input.onReady(innerContent);
419 → → → → → → → }
420 → → → → → → → this.scrollbar.scanDomNode();
421 → → → → → → → this.loadTextEditorViewState(input.getResource());
422 → → → → → → → this.updatedScrollPosition();
423 → → → → → → → });
424 → → → }
```

The conflict is resolved by accepting the incoming change, which adds a new block of code (lines 417-423) to the existing code (lines 411-413). The conflict is resolved by accepting the incoming change, which adds a new block of code (lines 417-423) to the existing code (lines 411-413).

- ▶ merge 하기 전에 워킹 디렉토리를 깔끔히 정리하는 것이 좋다.
  - 브랜치에 커밋하거나 Stash 해둔다. 그래야 어떤 일이 일어나도 다시 되돌릴 수 있다
- ▶ Merge 취소
  - `git merge --abort`

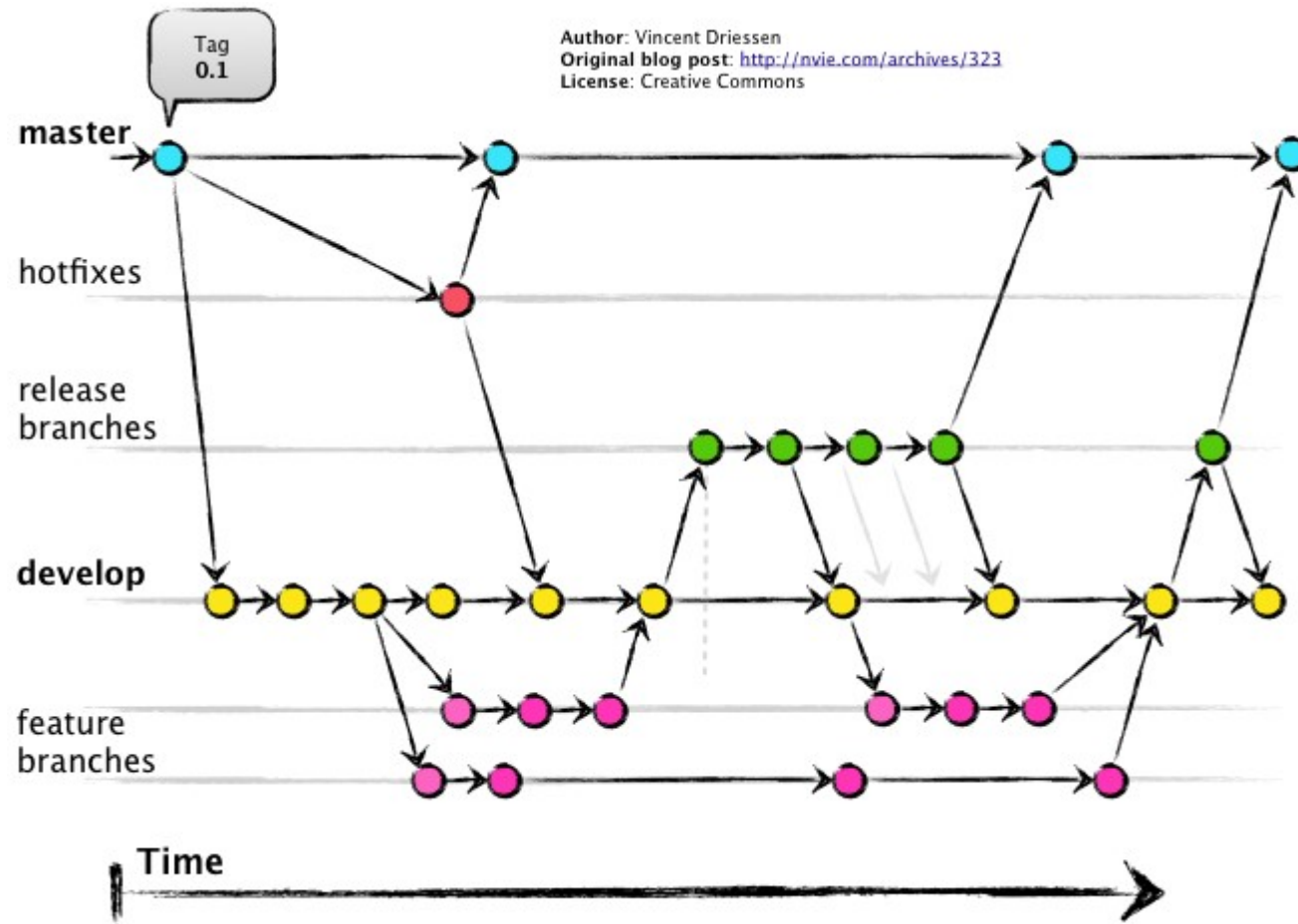
# 버전 변경시 Working directory 정리

- ❖ 수정된 working files 은 버전이 바뀔 때 merge가 발생할 수 있다.
  - ❖ checkout branch → branch 로 이동하고 해당 버전 file들을 working director 로 가져올 때
  - ❖ pull origin branch → origin/branch 를 가져와 branch 와 merge → merge된 local branch 를 checkout
- ❖ 해결 방안
  - 변경 사항을 commit 로 저장한다
  - 변경 사항을 별도 장소에 저장 - stash
- ▶ Stash: 하던 일을 잠시 치웠다 복구하기
  - git stash
    - 변경된 working files과 index(staged) 잠시 stack에 보관
  - git stash list
    - list stashes saved on the stack
  - git stash apply
    - 복원 (꼭 예전 브랜치로 돌아갈 필요는 없다.) stash는 남아 있음
  - git stash drop
    - 해당 stash 제거
  - git stash pop
    - 복원되고 해당 stash 제거

# Undo

- ▶ Latest commit 수정하기 (이미 push한 것은 안됨)
  - 수정한 파일을 staging area로 올리고 (commit log만 수정하려면 이 행동은 생략)  
`$ git commit -amend`
- ▶ Undo latest commit
  - 완전히 되돌리기  
`git reset --hard HEAD^`
  - Index, working directory는 유지한 채 되돌리기  
`git reset -soft HEAD^`
  - ❖ HEAD^ 대신에 <commit ID>를 적으면 그 commit 으로 HEAD가 reset됨 (매우 주의 필요)
- ▶ 충돌했을 때 merge 전으로 돌리기  
`git merge --abort`

# Git 협업 workflow



# Pull Request

원격 저장소의 어떤 branch에 내가 만든 changes(commit)을 push 했으니

- 변경사항을 검토해 주고 (토론)
- 문제 없다면 주 branch로 merge해 주기를 요청함

# Pull Request와 협동작업

- ▶ [local repository] 내가 작업한 branch featureA를 push한다.
- ▶ [GitHub remote repository] Pull Request (featureA → master) 를 만든다(open pull request).
  - 내가 작업한 branch 를 review 하고, 이 branch 를 master 로 merge 해 줄 것을 요청하는 내용이다.
  - 동료의 GitHub에서 PR이 보이며 토론에 참가한다.
  - 재 수정할 것을 요구할 수 있다.
  - 승인할 만 하면 (저장소 관리자가) merge한다.
  - Pull request를 close한다.
  - PR은 e-mail로도 전달된다.
- ▶ 모든 팀원은 수정된 master branch를 pull한다.

