

처음 시작하는



+



GitHub 활용하기



OSS 개발자 포럼

목차

1. Git, Github 소개
2. 기초 명령어 소개 / 쉬는 시간 ☕
3. 기초 실습
4. Github 기능 소개 및 실습 / 쉬는 시간 ☕
5. Github Pages 로 개발 블로그 만들기 (Today I Learned)
6. 오픈소스 프로젝트에 기여하는 법
7. Git 관련 도구 소개

이번 강의는 Git 과 Github 을 처음 접하시는 분들을 대상으로 준비를 하였습니다.
총 3시간 동안 위와 같은 순서로 진행할 예정인데요.

목차

1. Git, Github 소개

2. 기초 명령어 소개 ★★

3. 기초 실습 ★★★

4. Github 기능 소개 및 실습 ★

5. Github Pages 로 개발 블로그 만들기 (Today I Learned)

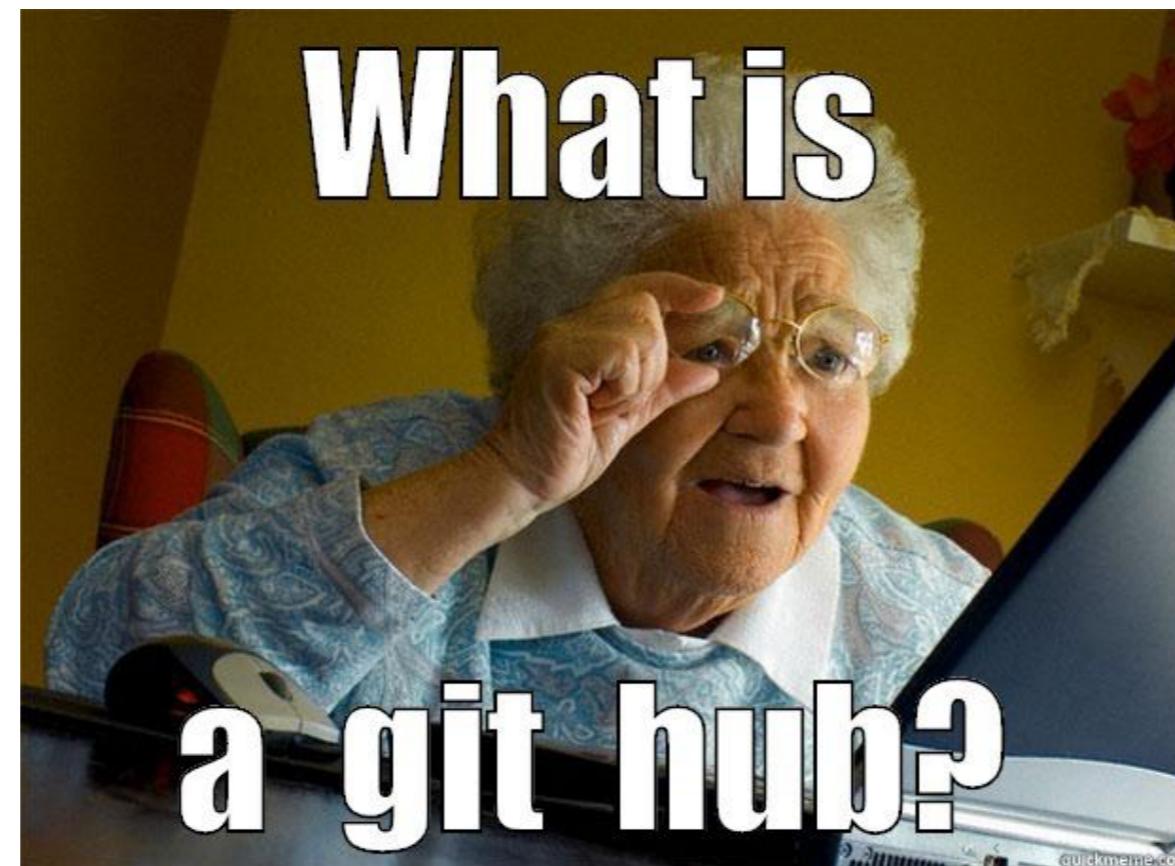
6. 오픈소스 프로젝트에 기여하는 법

7. Git 관련 도구 소개

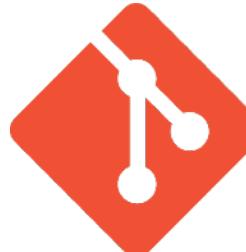
****Hands on** 강의라서 실습 중심으로 진행할 예정이며,
Github Pages 로 개발 블로그를 만드는 과정까지 다룰 예정입니다.**

Git and Github

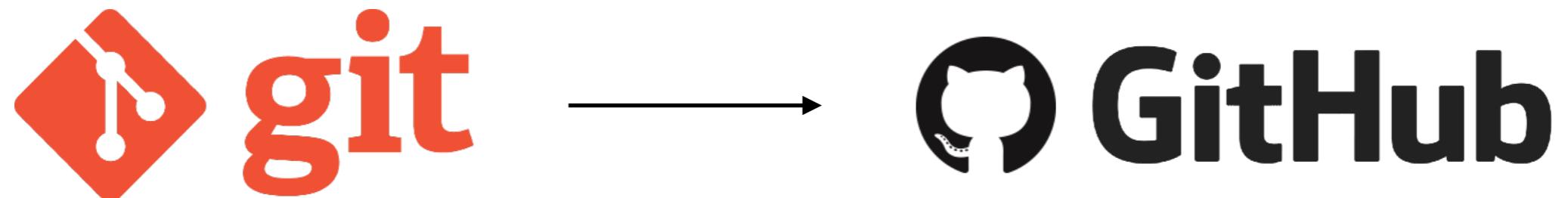
우선, Git 과 Github 에 대해 먼저 소개를 해드리겠습니다.



Git 과 Github 의 개념에 대해서는 충분히 헷갈릴 수 있어요. 저도 옛날엔 그랬으니까요.
결론부터 말씀드리자면, Git 과 Github 은 다릅니다.

 **git** 은 분산 버전관리 시스템(DVCS) 도구, **GitHub** 은 Git 을 제공하는 서비스

Git 은 Linux 의 창조주인 리누스 토발즈가 만든 분산 버전관리 시스템 도구이고,
Github 은 이것을 제공해주는 서비스입니다. 가장 유명해서 Git 하면 Github 을 떠올릴 수 있는 것 뿐이죠.



• || •

 Download WordPress 5.2.2

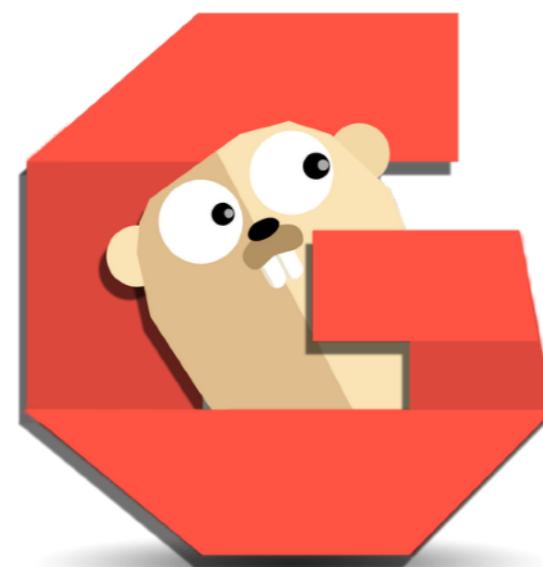
[Download .tar.gz](#)



WordPress Hosting

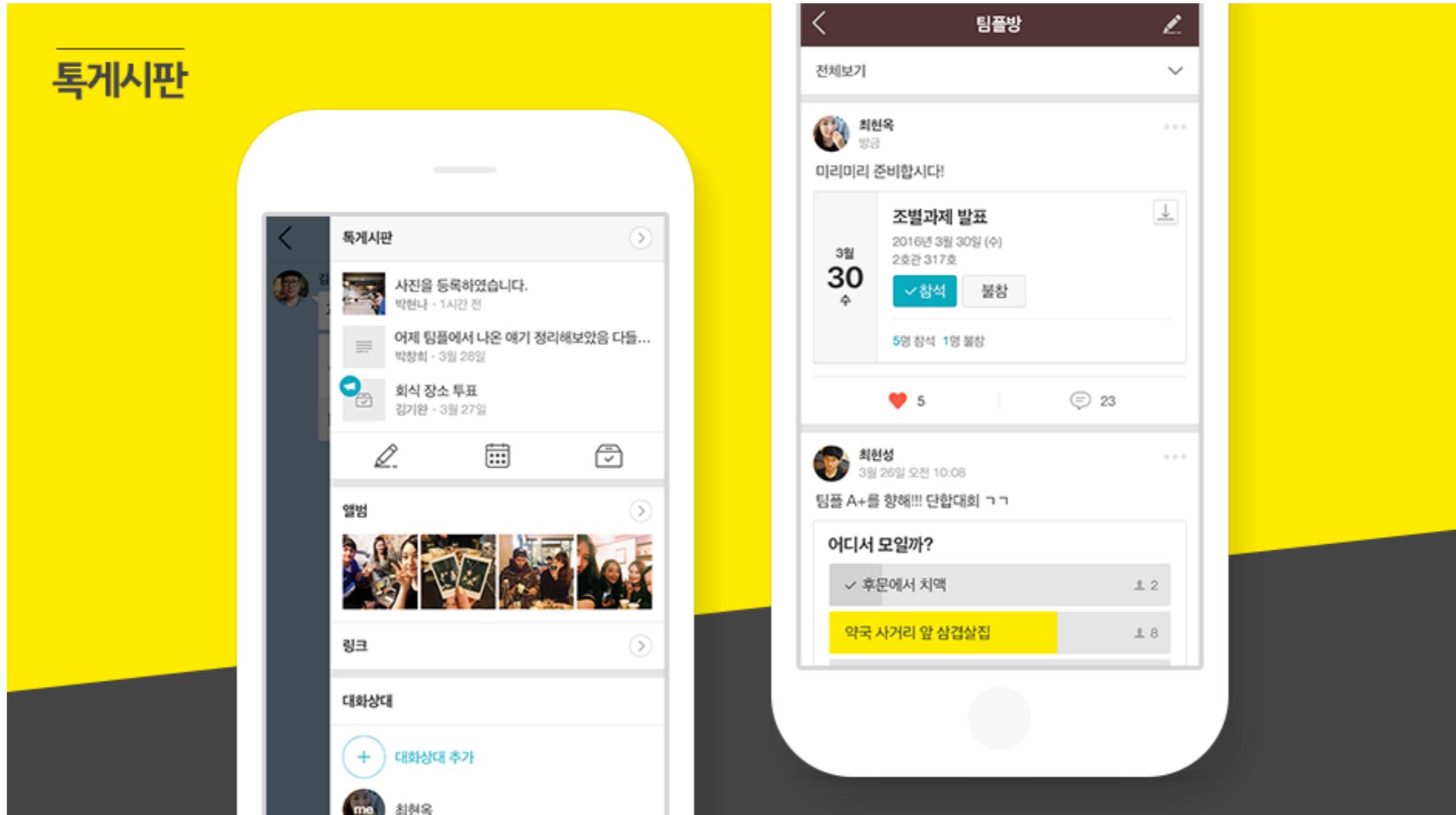
Choosing a hosting provider can be difficult, so we have selected a few of the best to get you started.

쉽게 말해서, 오픈소스로 받아서 직접 설치해서 사용하는 설치형 워드프레스와 설치하지 않고 바로 사용하는 호스팅형 워드프레스와 비슷하다고 볼 수 있어요.



그리고 Github 이 가장 유명하고 널리 쓰이고 있지만,
Git 을 서비스하는 곳은 Bitbucket, Gitlab, Gogs 등 여러가지가 있습니다.

..근데, 왜 써야 하죠..?



그런데… 도대체 분산 버전관리 시스템이 뭐길래 우리는 왜 이것을 써야 하는 걸까요?
번거롭게 굳이 안 써도 될 것 같지 않나요? 사실 카톡으로 주고받아서 합쳐도 되는데 말이에요.

..다른 것도 있는데요?



그리고… Git 이외에도 CVS, SVN, Mercurial 등
여러가지로 다양한 도구들이 있는데 왜 Git 을 많이 쓰는 걸까요?

버전 관리 도구 ?

우선, 버전관리 도구를 왜 쓰는지에 대해 먼저 말씀드릴게요.

<p>히스토리</p> <p>음.. 우리가 개발을 체계적으로 잘 하고 있군</p> <p>언제 어떤 기능이 추가되었고</p> <p>이 기능은 저 사람이 만들었군</p>	<p> 4월 1일 - 메인페이지 레이아웃 수정</p> <p> 4월 2일 - 로그인 기능 추가</p> <p> 4월 3일 - 페북 아이디로 가입하기 추가</p> <p> 4월 4일 - 회원정보 수정하기 추가</p> <p> 4월 5일 - 글쓰기 기능 수정</p> <p style="text-align: center;">⋮</p> <p> 8월 8일 - 프로모션 페이지 수정</p> <p> 8월 9일 - 메인페이지 개선</p>	<p>되돌리기</p> <p>문제가 심각하니 이전 버전으로 되돌려서 복구해야해</p> <p>Ctrl + Z !!</p>
		<p>...심각한 문제 발생!</p> 

개발을 하다 보면

- 1) 여태까지 개발해 온 이력을 확인해야 할 때가 있을 수 있고,
- 2) 예전에 작업했던 상태로 다시 되돌려야 할 때도 있어요.

공지사항 올리기

송금하기

이모티콘 보내기

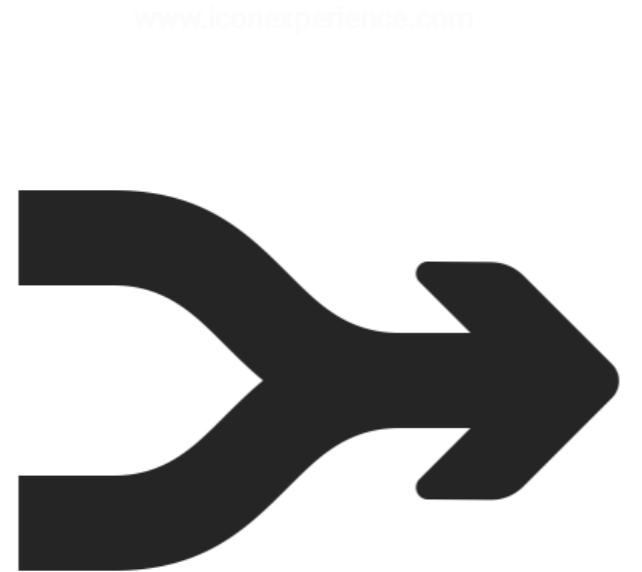
메시지 보내기

사진 보내기

사용자 차단하기

프로필 꾸미기

⋮
⋮
⋮



최종 결과물

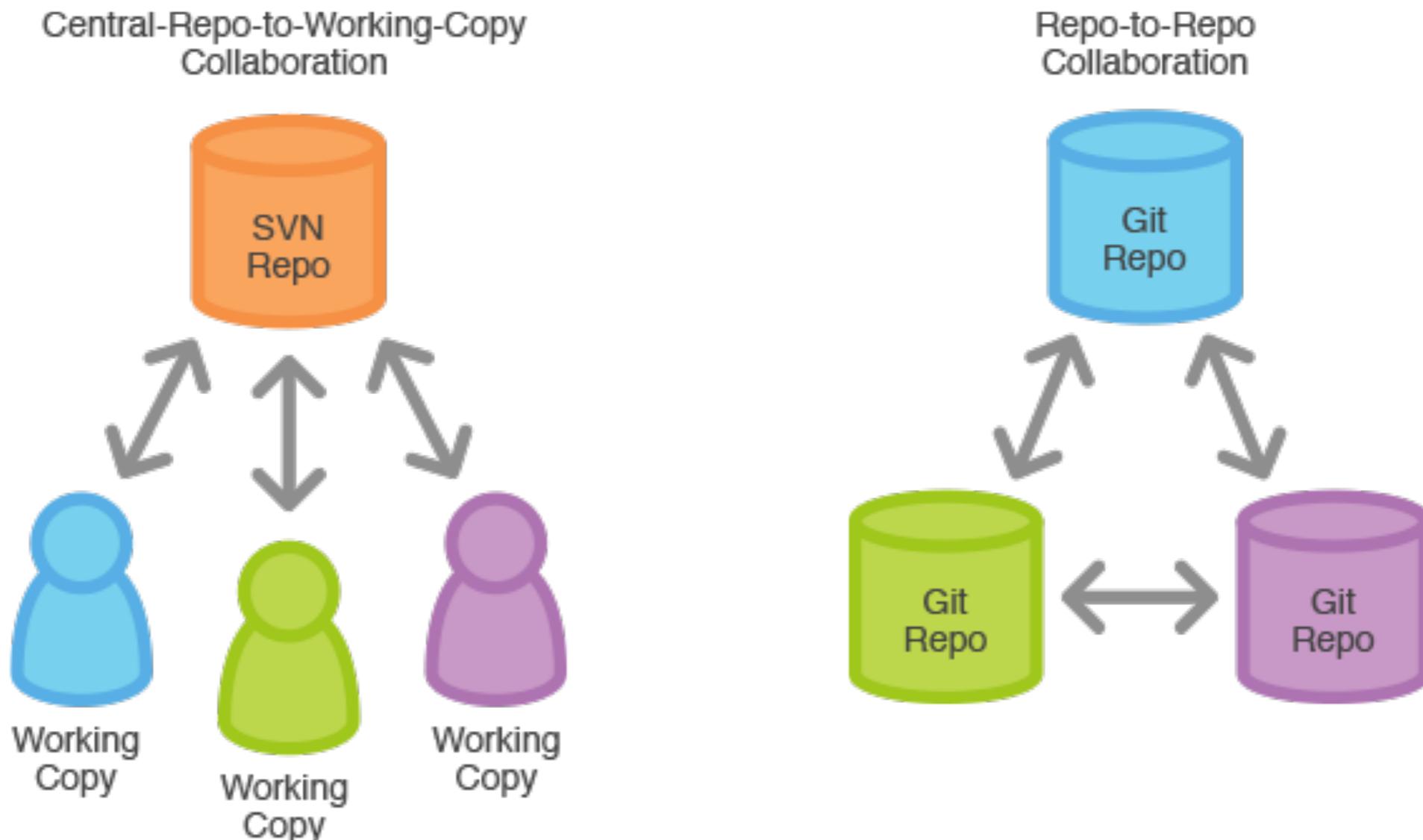
그리고 여러 개발자들이 작업을 나눠서 동시에 개발을 진행하면
개발 완료 후 합치는 과정에서 서로 겹친 부분 때문에 3) 충돌이 일어날 수도 있고요.



이러한 불편함과 문제점들을 보다 효율적으로 극복하기 위해
버전관리도구(Version Control System)를 사용하는 것입니다.

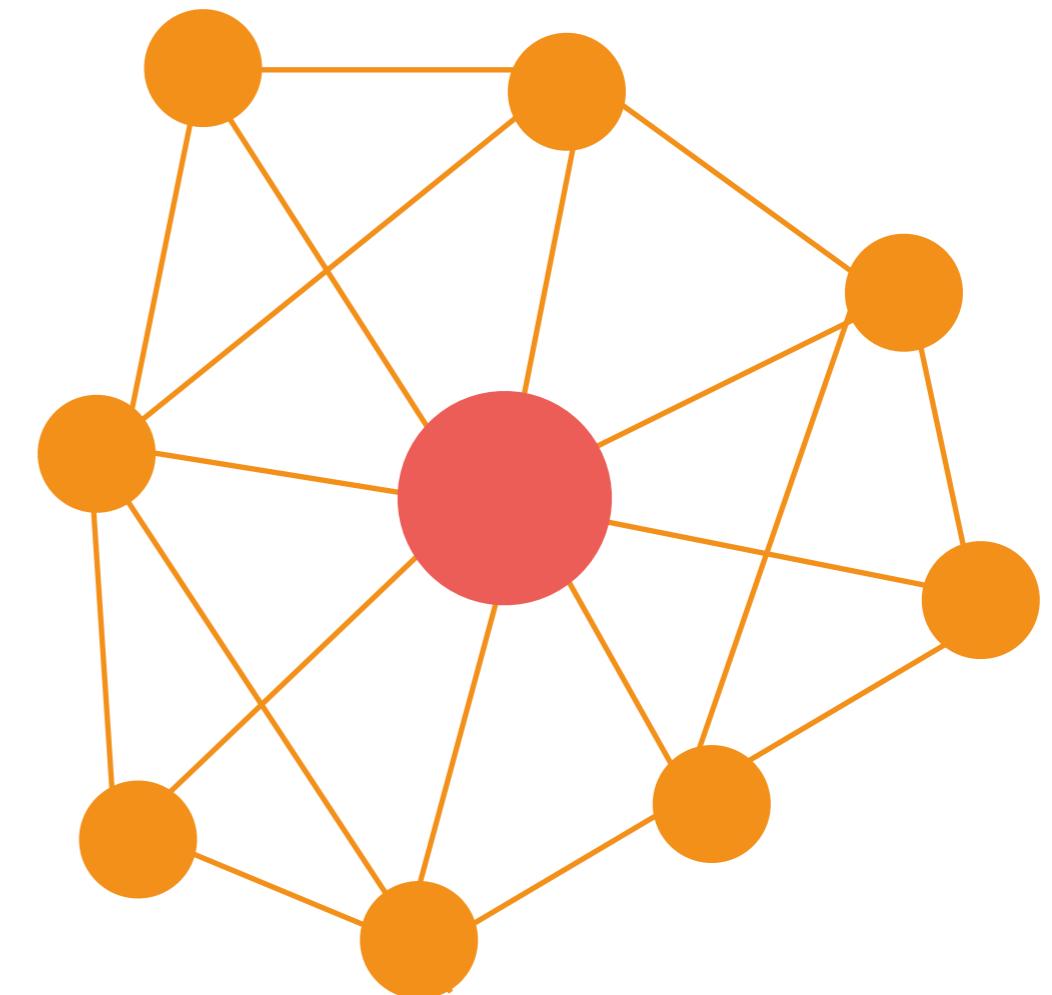
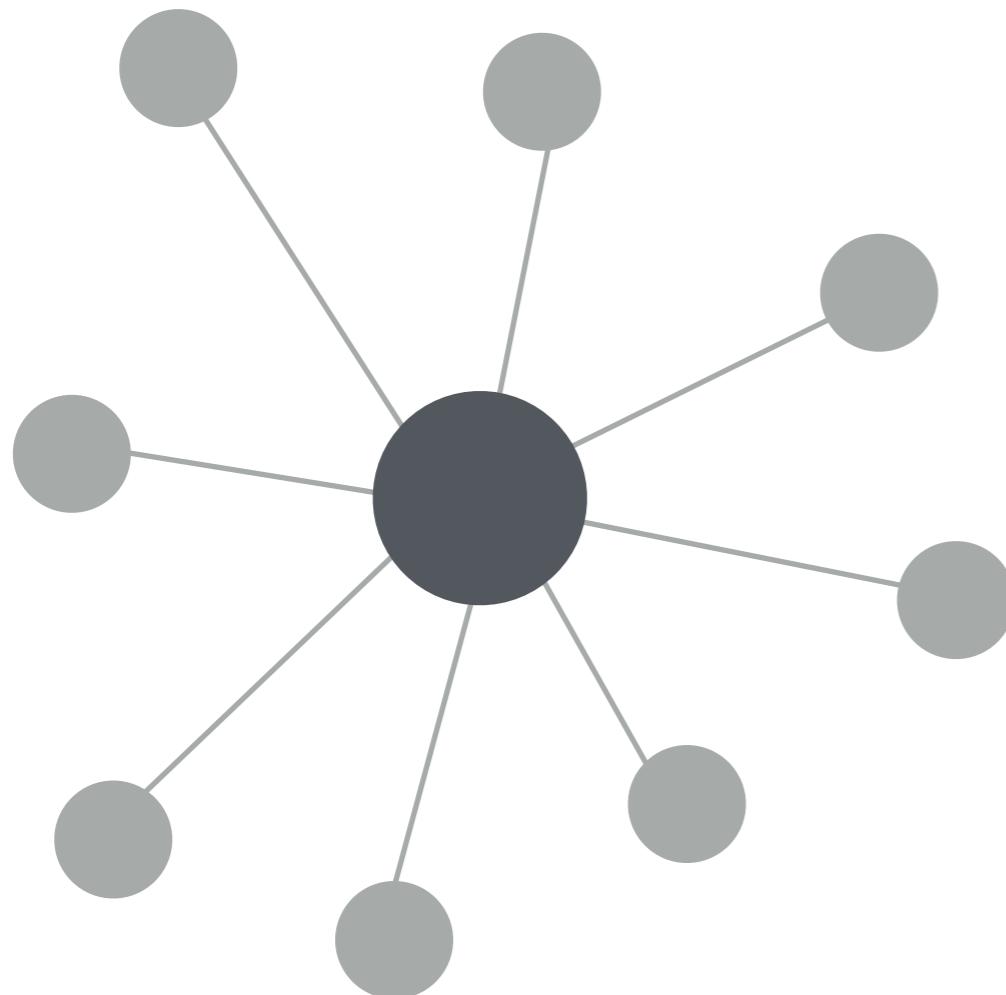
Why Git ?

근데, 그 중에서 왜 Git 을 가장 많이 쓰게 된 걸까요?



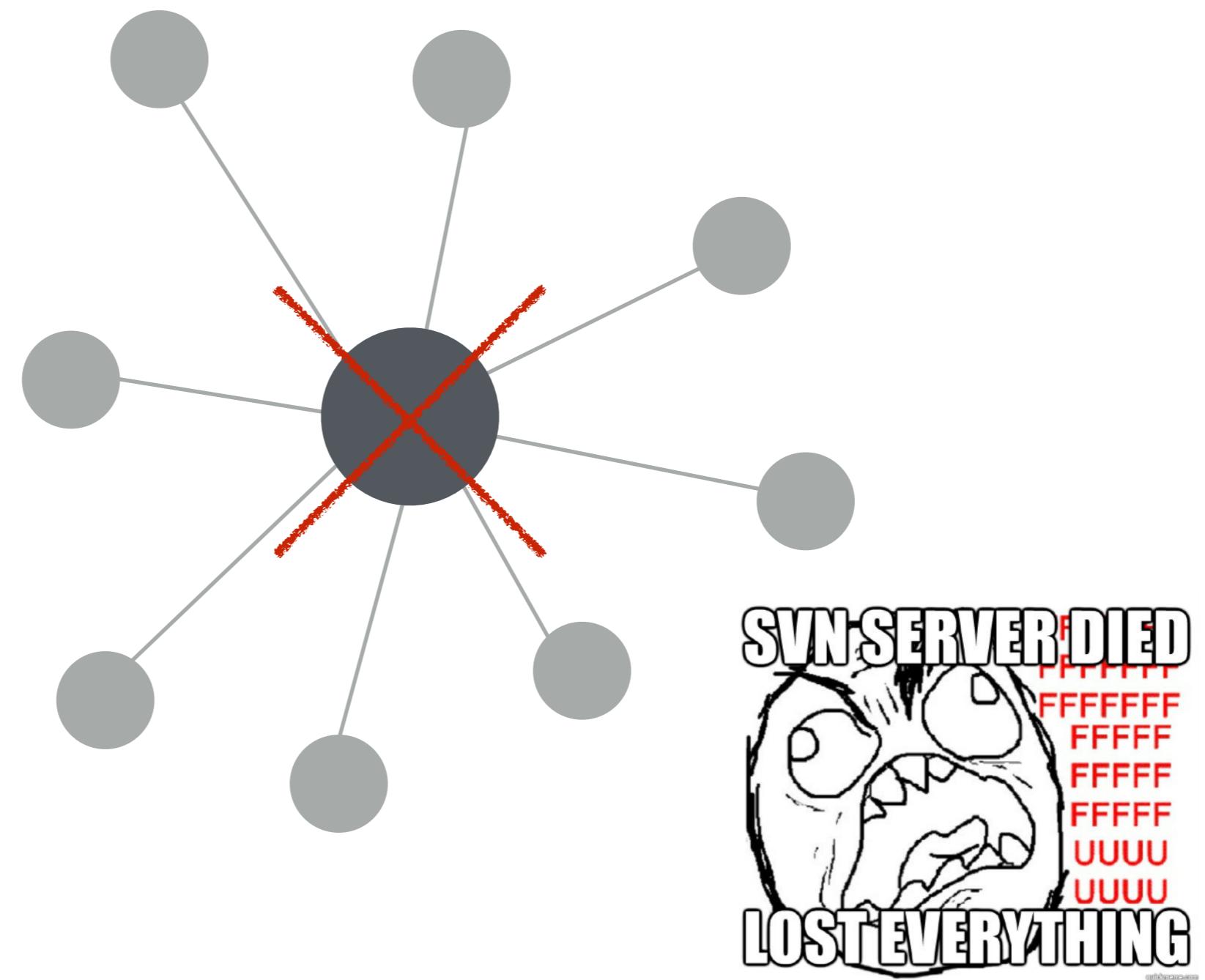
우선, SVN(Subversion) 과 간단히 비교를 해볼게요.

서브버전의 경우 하나의 중앙 저장소가 있고, Git 의 경우에는 원격 저장소와 로컬 저장소가 있어요.

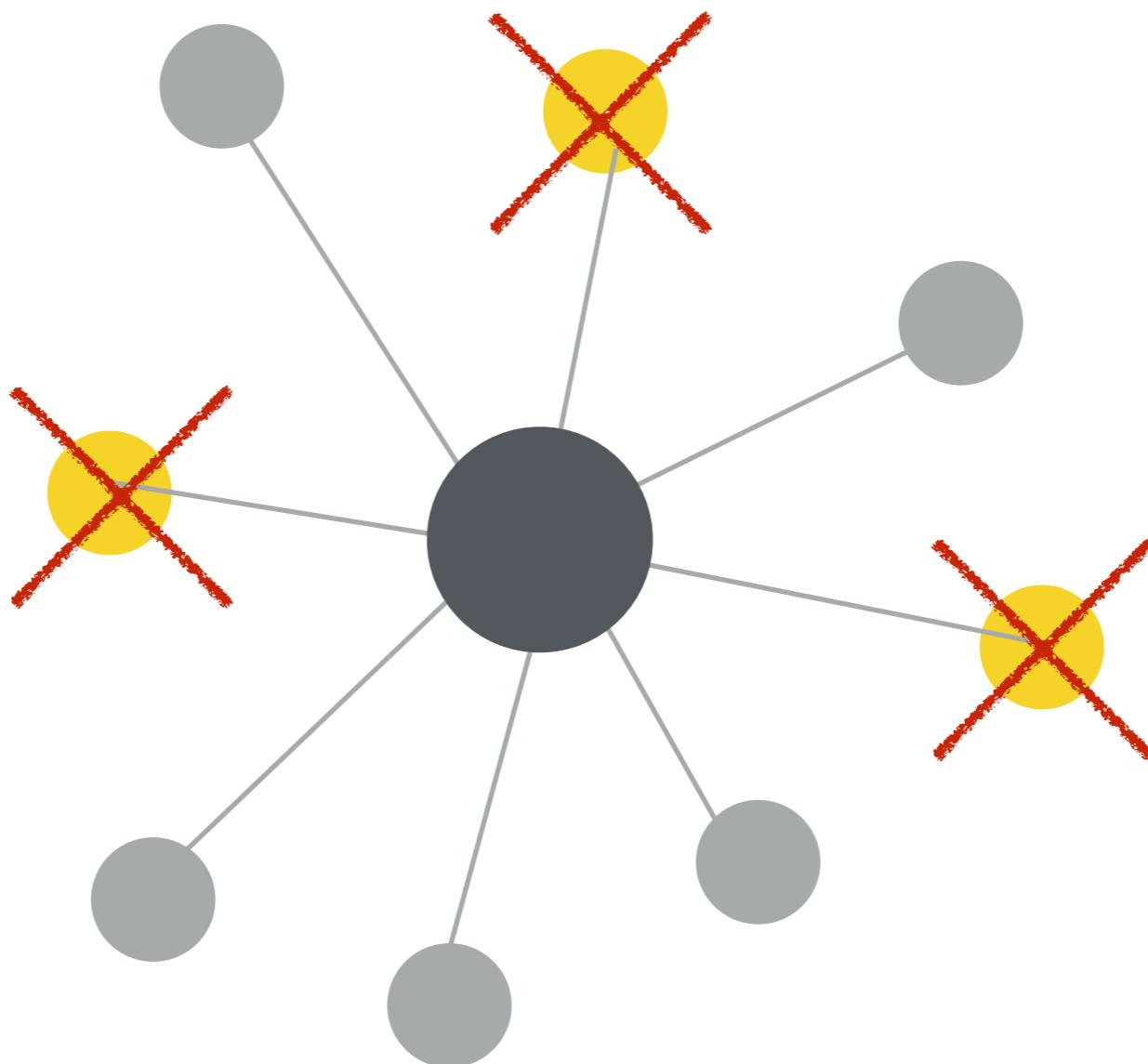


간단히 그래프로 비유를 해볼게요.

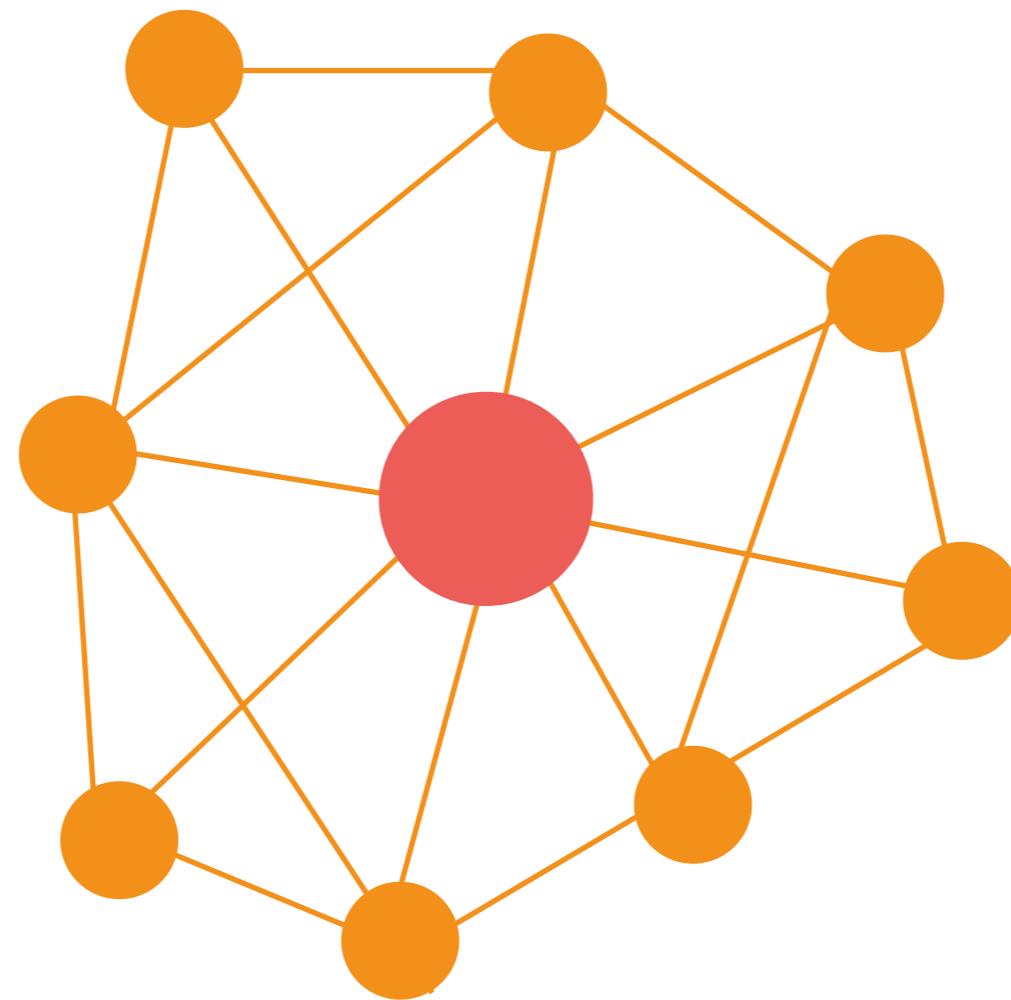
SVN 은 왼쪽처럼 중앙 집중형 구조를 가지고, Git 은 오른쪽처럼 그물망 모양을 가진다고 할 수 있는데요.



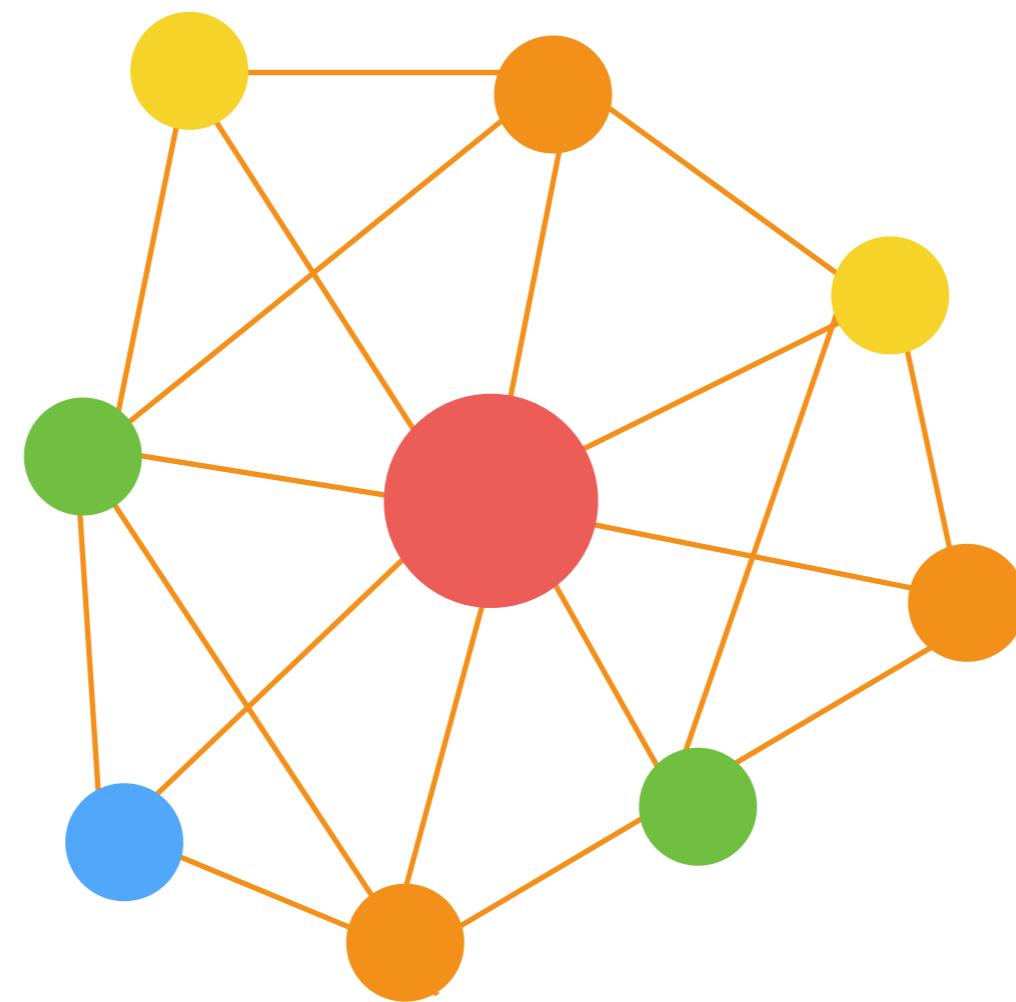
그런데 SVN은 중앙 저장소에 집중되어 있는 구조라
중앙저장소에 문제가 생겼을 경우, 따로 백업을 해두지 않는 이상 복구를 하기가 좀 힘들어요.



그리고 저장소를 공유하여 사용하기 때문에, 동시에 여러명이서 개발을 하기에 어려움이 있습니다.



그러나 Git 은 리모트 저장소를 로컬 저장소로 복사해와서 작업하는 구조로,
로컬과 리모트가 분산되어 있기 때문에 더욱 안전하면서도 여러명이서 효율적으로 개발할 수 있어요.



덕분에 불특정 다수가 동시에 작업하기 용이한 환경을 지니고 있어서
오픈소스 진영에서 주로 많이 쓰이고 있습니다.



mercurial

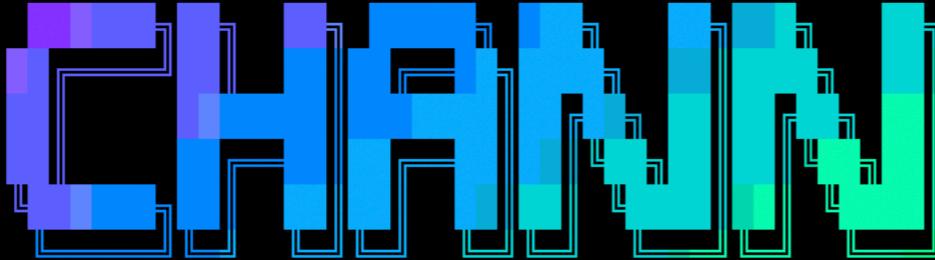
<http://d2.naver.com/helloworld/1011>

Mercurial 과 Git 은 둘 다 분산형 버전관리 도구라 비슷한 점이 많습니다.

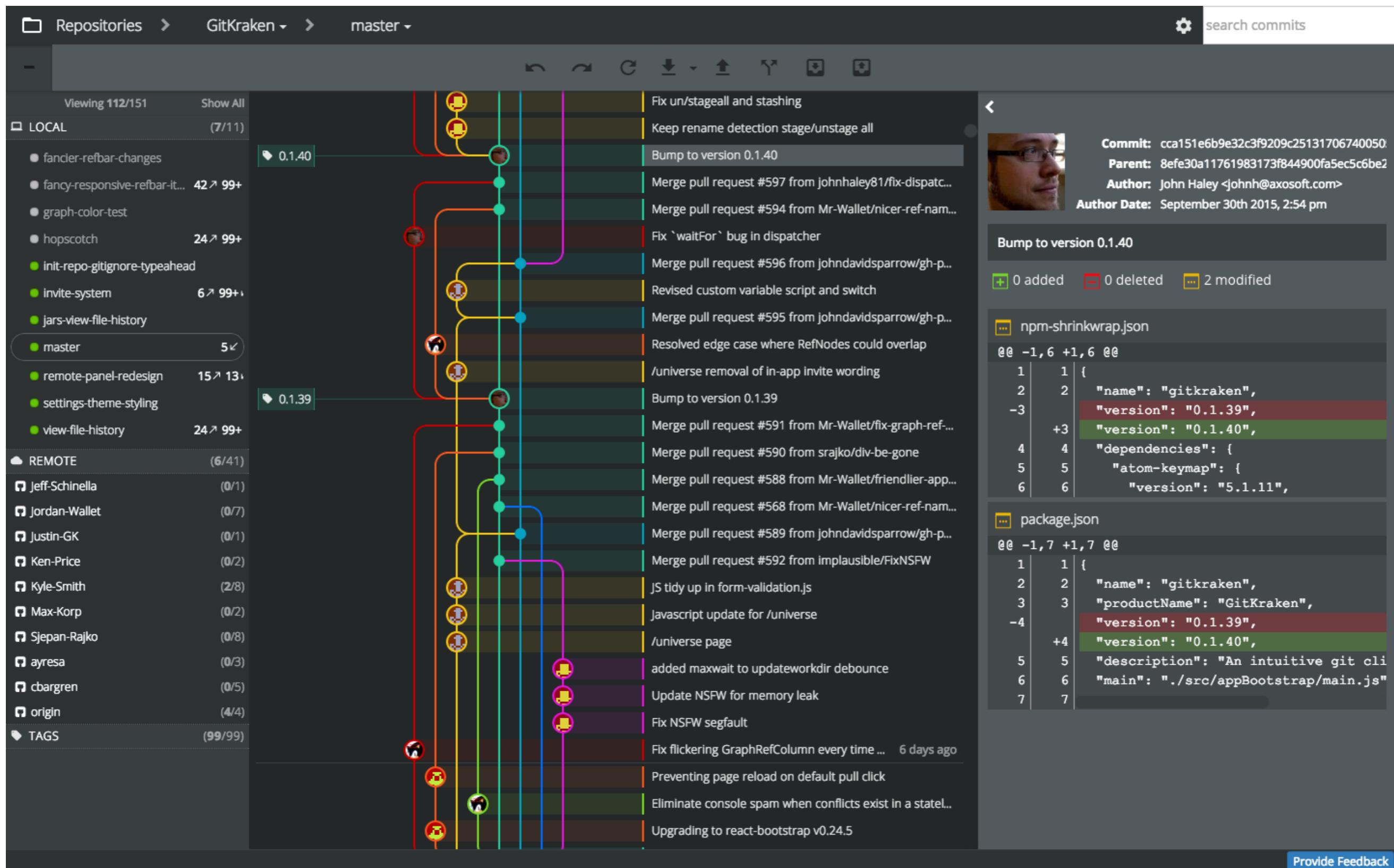
차이를 간단히 요약하자면 Branch 중심적(Mercurial) / Commit 중심적(Git) 이라 할 수 있을 것 같아요.

보다 자세한 사항은 따로 질문을 주시거나 위의 링크를 참고해주시면 좋을 것 같습니다.

```
1. chann@CHANN-MacBook: ~/git/kmu-swcamp-tutorial (zsh)
Last login: Thu Jul 25 14:19:27 on ttys002

  
...with MacBook Pro 2017  
-----  
14:20  up 57 mins, 3 users, load averages: 1.62 1.36 1.42  
USER     TTY      FROM          LOGIN@  IDLE WHAT  
chann    console  -           13:23    56 -  
chann    s000     -           14:20      - w  
chann    s002     -           14:19      - -zsh  
-----  
▶14:20:07 chann@CHANN-MacBook:~  
$ cd ~/git/kmu-swcamp-tutorial  
▶14:20:11 chann@CHANN-MacBook:~/git/kmu-swcamp-tutorial  
$ █
```

그리고 저희는 오늘 CLI (Command Line Interface) 환경에서 실습을 할 예정입니다.
공부를 위해서는 터미널에서 명령어를 하나하나 직접 입력해 보는 것이 가장 효과적이라고요.



Provide Feedback

이렇게 멋진 GUI 도구도 있긴 하지만, 버튼을 누르면 미리 설정된 CLI 명령어가 실행되는 것에 불과하고 GUI 특성 상 세부적인 명령을 실행하기 어렵습니다. 제공해주는 버튼과 기능만 써야 하고 속도가 많이 느려요.

1. cmatrix (cmatrix)

```

? X B \ \ y M L : θ K m q F 5 # p A " | w : - 0 a i 0 o a q
Q * i + 7 Y B ^ p ? q F ? Y # X $ ' c ] N - o J a j & i N y p ?
f . g f T 5 G C N ? ` h M z . F h ; R + P R - . F ! <
E l C 0 8 M 3 * 0 N W Y I 4 W v ^ : # . b : 6 ? . \ 2 L
: u s G j J 1 " 4 J G , / 6 d - # u b 8 a K z w Q = Y 1
M % } ( 0 R w , 0 U j Y @ > a Z & H x W G I
r U ! w b W r * > X = g t V R r { g z % 4 S T G d 0
r s ( ) $ " I A 7 0 D a l & = G ) x [ ! # r q - Y J $
K ' K ` ) z " + G 7 & y 3 k Y K @ 1 C + w y 2 7 i p 9 l
, ' ] S C & T N 2 > 2 L o B - 3 < 1 F p u l , . V S 6 K
4 ^ ? D L " Q 3 V < F 0 4 \ * $ y p g P u k = M Q c b
( I i 5 0 u = A l y s 6 q e h v h { Y 0 2 ; m K v = b ( 4 m
x G y ; K + ! C $ } f \ q r J 9 k K $ E 1 c v ) + "
k N l V ) Q e | r ; > 2 ` p g f " # e # : G z m f 0
p Z 1 D ) Z F / b 0 9 } @ F 7 B c 0 A . % - Q
t f ! t m x ; F ] a H & b G W w s ` E : 8 j ' u
$ w \ C ? | ! 5 H y & T 0 F & Y K ] ( < :
? s L C , 6 I t y M V X H B C u V | A $ 8 f ] m
t ( L K D 9 ] E ) h . e u I e F ^ @ 6 v % [ ) , Z
y _ k o + { r h . k p ; \ d - I r F m ) L F 0 0 7
- B * 8 ) [ s J \ k . f P t f * C { 3 b i , e y I <
} H 7 _ k ; , k . f P t f * C { 3 b i , e y I <
@ O > Z e R E @ W . | j k G @ [ u i 0 < m " r r 7 t |
] # ? S Q ^ ( X 0 | z | : / l ] . ; j ' B 7 K b x s Q x
+ ; 6 K ? p b } W y } 0 I < o > d & W V Z K #
A " Z ! N x F ] ) u n 4 h ` ; 5 J . g 6 0 4 & o { w ^
l - 4 7 3 2 l " > 2 & h W A Z p ; Z s m z 3 v G 7 : n 6 | X
V | [ R M Y ? 0 { & h W A Z p ; Z s m z 3 v G 7 : n 6 | X
h ' z p + H / X 0 l U c f { 7 " e s 0 = G . 4 * p ^
4 : F h g ) a ! o X H : Y @ % . o 7 - 3 o 0 } P H ^ V 7
C + = C L y t - J 5 K P @ M / 9 , | d { , ' 8 + e W =
f F N W , r Y v - 1 q 5 T r / ! x . l , _ g ( ^ x
E B > f o : V c r 5 ^ ^ Z 1 { h j o b G g 0 v ; d
J f 5 T @ r 7 t y J 0 A " 8 Q 8 = g K 0 ( e g
I 5 A : [ M D Y j A X 0 u P [ ' v x : ` @ M / b ,
^ # r X 7 ^ X # . L 0 I o c ! ^ u } Y M / j X j
C > 9 ` l @ % U n b J k R p - h s d P A ^ @ ! t ! E S
7 & d J 0 m 2 a . p G n V Z S S 0 T > F 6 e X 7 b S ^ ( -
K w 1 0 2 < k @ ? 2 ^ ( n > P ^ ? - : u Q Z < . > p 0 ] 0 "
d 8 H # e e b 2 | K l ' B l j _ ? - : u Q Z < . > p 0 ] 0 "

```

… 그리고 터미널 화면에서 명령어 치고 하는 게 좀 더 힘하고, 있어보이고, 멋지잖아요..?
이번 강의시간 만큼은 터미널의 멋짐을 충분히 느끼고 가셨으면 좋겠습니다. 😎

기초 명령어 소개

그럼 이제부터 Git 의 기초 명령어들을 먼저 소개해드릴게요.

기초 명령어 소개 이후에는 잠깐 휴식시간을 가지고 본격적인 실습을 진행할 예정입니다.

\$ git <command>
명령어

<option>
옵션

<args>
입력값

다음은 CLI 환경에서의 git 기본 사용 명령어 틀입니다.
앞으로 다루게 될 기본 명령어들은 위와 같은 틀에서 크게 벗어나지 않을거에요.

옵션 (인라인 메시지)

```
$ git commit -m “커밋 메시지 작성”
```

명령어 (커밋)

입력값 (커밋 메시지)

```
$ git push origin master --force
```

명령어 (푸시)

입력값
(저장소 이름)

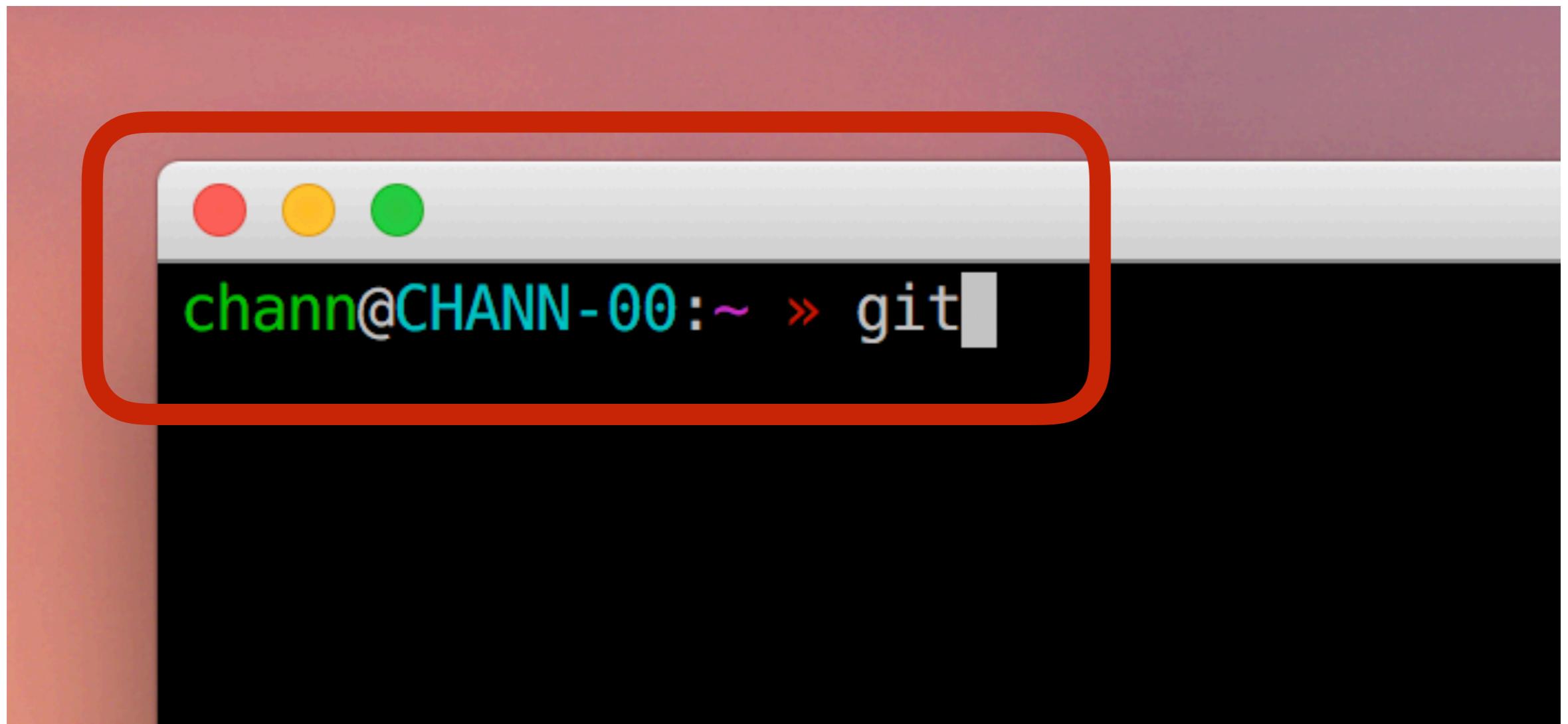
master

--force

옵션
제로 하기)

이렇게 말이죠.

옵션의 경우 보통 명령어 바로 뒤에 쓰지만 간혹 입력값 맨 뒤에 붙을 수도 있습니다.



먼저, git 명령을 입력해보도록 하겠습니다.

```

chann@CHANN-00:~ » git
usage: git [--version] [--help] [-C <path>] [-c name=value]
          [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
          [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
          [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
          <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  reset     Reset current HEAD to the specified state
  rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect    Use binary search to find the commit that introduced a bug
  grep      Print lines matching a pattern
  log       Show commit logs
  show      Show various types of objects
  status    Show the working tree status

grow, mark and tweak your common history
  branch   List, create, or delete branches
  checkout Switch branches or restore working tree files
  commit   Record changes to the repository
  diff     Show changes between commits, commit and working tree, etc
  merge    Join two or more development histories together
  rebase   Reapply commits on top of another base tip
  tag      Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch    Download objects and refs from another repository
  pull     Fetch from and integrate with another repository or a local branch
  push     Update remote refs along with associated objects

  git help -a and git help -g list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
chann@CHANN-00:~ »

```

위와 같이 터미널에서 `git` 이라고만 입력을 하면 다음과 같이 도움말이 뜨는데요.

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

reset Reset current HEAD to the specified state

rm Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)

bisect Use binary search to find the commit that introduced a bug

grep Print lines matching a pattern

log Show commit logs

show Show various types of objects

status Show the working tree status

grow, mark and tweak your common history

branch List, create, or delete branches

checkout Switch branches or restore working tree files

commit Record changes to the repository

diff Show changes between commits, commit and working tree, etc

merge Join two or more development histories together

rebase Reapply commits on top of another base tip

tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

가장 대표적이고 많이 쓰이는 일반적인 명령어 리스트를 보여줍니다.

명령어도 영어고, 도움말도 영어입니다. 앞으로는 영어에 익숙해지시는 것이 좋겠죠?

다양한 상황에서 사용할 수 있는 기본적인 Git 명령어들을 소개합니다:

작업 공간 시작하기 (자세히 알아보기: git help tutorial)

`clone` 새로운 디렉토리에 저장소를 복제하기

`init` 빈 저장소를 새롭게 만들거나 기존의 저장소를 다시 초기화하기

현재 변동사항 다루기 (자세히 알아보기: git help everyday)

`add` 스테이지 영역에 수정된 파일 추가하기

`mv` 파일, 폴더, 심볼릭 링크의 위치를 옮기거나 이름을 바꾸기

`reset` 현재 HEAD 를 특정 위치로 이동(초기화) 하기

`rm` 현재 작업 트리에서 파일을 제거하고 스테이지 영역에서도 지우기

히스토리와 상태 검토하기 (자세히 알아보기: git help revisions)

`bisect` 바이너리 서치로 버그 찾기

`grep` 패턴과 매칭된 줄 출력

`log` 커밋 로그 보기

`show` 다양한 타입의 오브젝트 보기

`status` 현재 작업중인 트리의 상태 보기

히스토리 수정, 마크, 추가하기

`branch` 브랜치 리스트 보기, 브랜치 만들기 / 지우기

`checkout` 브랜치 바꾸기 또는 작업중인 트리의 파일들 복구하기

`commit` 로컬 저장소에 수정사항 반영하기

`diff` 커밋 또는 작업중인 트리의 변동사항 보기 등

`merge` 2개 혹은 그 이상의 개발 히스토리 합치기

`rebase` 다른 베이스 브랜치의 최상단에서 커밋 재적용하기 (부모 브랜치 바꾸기)

`tag` GPG 사인과 함께 태그 만들기, 보기, 삭제하기 등

협업하기 (자세히 알아보기: git help workflows)

`fetch` 다른 브랜치로부터 오브젝트(object)와 레퍼런스(refs) 다운로드하기

`pull` 로컬 브랜치 또는 다른 브랜치로부터 fetch 받아서 적용하기

`push` 관련된 오브젝트(object)로 리모트 레퍼런스(refs)를 업데이트 하기

이번에는 편의상 한글로 번역을 해드렸어요. 먼저, 작업공간 시작하기라는 부분부터 소개해드릴게요.

`clone` 은 저장소를 복제해 오는 것이고, `init` 은 빈 저장소를 새로 만들거나, 기존의 것을 초기화하는 것입니다.

다양한 상황에서 사용할 수 있는 기본적인 Git 명령어들을 소개합니다:

작업 공간 시작하기 (자세히 알아보기: git help tutorial)

clone 새로운 디렉토리에 저장소를 복제하기

init 빈 저장소를 새롭게 만들거나 기존의 저장소를 다시 초기화하기

현재 변동사항 다루기 (자세히 알아보기: git help everyday)

add 스테이지 영역에 수정된 파일 추가하기

mv 파일, 폴더, 심볼릭 링크의 위치를 옮기거나 이름을 바꾸기

reset 현재 HEAD 를 특정 위치로 이동(초기화) 하기

rm 현재 작업 트리에서 파일을 제거하고 스테이지 영역에서도 지우기

히스토리와 상태 검토하기 (자세히 알아보기: git help revisions)

bisect 바이너리 서치로 버그 찾기

grep 패턴과 매칭된 줄 출력

log 커밋 로그 보기

show 다양한 타입의 오브젝트 보기

status 현재 작업중인 트리의 상태 보기

히스토리 수정, 마크, 추가하기

branch 브랜치 리스트 보기, 브랜치 만들기 / 지우기

checkout 브랜치 바꾸기 또는 작업중인 트리의 파일들 복구하기

commit 로컬 저장소에 수정사항 반영하기

diff 커밋 또는 작업중인 트리의 변동사항 보기 등

merge 2개 혹은 그 이상의 개발 히스토리 합치기

rebase 다른 베이스 브랜치의 최상단에서 커밋 재적용하기 (부모 브랜치 바꾸기)

tag GPG 사인과 함께 태그 만들기, 보기, 삭제하기 등

협업하기 (자세히 알아보기: git help workflows)

fetch 다른 브랜치로부터 오브젝트(object)와 레퍼런스(refs) 다운로드하기

pull 로컬 브랜치 또는 다른 브랜치로부터 fetch 받아서 적용하기

push 관련된 오브젝트(object)로 리모트 레퍼런스(refs)를 업데이트 하기

두번째는 변동사항을 다루는 것인데요.

커밋을 할 파일을 추가(add) 또는 삭제(reset, rm) 등을 합니다. 주로 add, reset 을 많이 사용합니다.

다양한 상황에서 사용할 수 있는 기본적인 Git 명령어들을 소개합니다:

작업 공간 시작하기 (자세히 알아보기: git help tutorial)

clone 새로운 디렉토리에 저장소를 복제하기

init 빈 저장소를 새롭게 만들거나 기존의 저장소를 다시 초기화하기

현재 변동사항 다루기 (자세히 알아보기: git help everyday)

add 스테이지 영역에 수정된 파일 추가하기

mv 파일, 폴더, 심볼릭 링크의 위치를 옮기거나 이름을 바꾸기

reset 현재 HEAD 를 특정 위치로 이동(초기화) 하기

rm 현재 작업 트리에서 파일을 제거하고 스테이지 영역에서도 지우기

히스토리와 상태 검토하기 (자세히 알아보기: git help revisions)

bisect 바이너리 서치로 버그 찾기

grep 패턴과 매칭된 줄 출력

log 커밋 로그 보기

show 다양한 타입의 오브젝트 보기

status 현재 작업중인 트리의 상태 보기

히스토리 수정, 마크, 추가하기

branch 브랜치 리스트 보기, 브랜치 만들기 / 지우기

checkout 브랜치 바꾸기 또는 작업중인 트리의 파일들 복구하기

commit 로컬 저장소에 수정사항 반영하기

diff 커밋 또는 작업중인 트리의 변동사항 보기 등

merge 2개 혹은 그 이상의 개발 히스토리 합치기

rebase 다른 베이스 브랜치의 최상단에서 커밋 재적용하기 (부모 브랜치 바꾸기)

tag GPG 사인과 함께 태그 만들기, 보기, 삭제하기 등

협업하기 (자세히 알아보기: git help workflows)

fetch 다른 브랜치로부터 오브젝트(object)와 레퍼런스(refs) 다운로드하기

pull 로컬 브랜치 또는 다른 브랜치로부터 fetch 받아서 적용하기

push 관련된 오브젝트(object)로 리모트 레퍼런스(refs)를 업데이트 하기

세번째는 히스토리 관련 명령어들입니다.

보통은 log , status, show 순으로 주로 사용을 하는데요. bisect 라는 명령어는 디버깅을 할 때 사용합니다.

다양한 상황에서 사용할 수 있는 기본적인 Git 명령어들을 소개합니다:

작업 공간 시작하기 (자세히 알아보기: git help tutorial)

clone 새로운 디렉토리에 저장소를 복제하기

init 빈 저장소를 새롭게 만들거나 기존의 저장소를 다시 초기화하기

현재 변동사항 다루기 (자세히 알아보기: git help everyday)

add 스테이지 영역에 수정된 파일 추가하기

mv 파일, 폴더, 심볼릭 링크의 위치를 옮기거나 이름을 바꾸기

reset 현재 HEAD 를 특정 위치로 이동(초기화) 하기

rm 현재 작업 트리에서 파일을 제거하고 스테이지 영역에서도 지우기

히스토리와 상태 검토하기 (자세히 알아보기: git help revisions)

bisect 바이너리 서치로 버그 찾기

grep 패턴과 매칭된 줄 출력

log 커밋 로그 보기

show 다양한 타입의 오브젝트 보기

status 현재 작업중인 트리의 상태 보기

히스토리 수정, 마크, 추가하기

branch 브랜치 리스트 보기, 브랜치 만들기 / 지우기

checkout 브랜치 바꾸기 또는 작업중인 트리의 파일들 복구하기

commit 로컬 저장소에 수정사항 반영하기

diff 커밋 또는 작업중인 트리의 변동사항 보기 등

merge 2개 혹은 그 이상의 개발 히스토리 합치기

rebase 다른 베이스 브랜치의 최상단에서 커밋 재적용하기 (부모 브랜치 바꾸기)

tag GPG 사인과 함께 태그 만들기, 보기, 삭제하기 등

협업하기 (자세히 알아보기: git help workflows)

fetch 다른 브랜치로부터 오브젝트(object)와 레퍼런스(refs) 다운로드하기

pull 로컬 브랜치 또는 다른 브랜치로부터 fetch 받아서 적용하기

push 관련된 오브젝트(object)로 리모트 레퍼런스(refs)를 업데이트 하기

네번째는 히스토리 수정에 대한 명령어들입니다.

브랜치 추가, 리베이스, 머지 등이 있고 diff로 변동사항을 비교해서 볼 수도 있죠.

다양한 상황에서 사용할 수 있는 기본적인 Git 명령어들을 소개합니다:

작업 공간 시작하기 (자세히 알아보기: git help tutorial)

clone 새로운 디렉토리에 저장소를 복제하기

init 빈 저장소를 새롭게 만들거나 기존의 저장소를 다시 초기화하기

현재 변동사항 다루기 (자세히 알아보기: git help everyday)

add 스테이지 영역에 수정된 파일 추가하기

mv 파일, 폴더, 심볼릭 링크의 위치를 옮기거나 이름을 바꾸기

reset 현재 HEAD 를 특정 위치로 이동(초기화) 하기

rm 현재 작업 트리에서 파일을 제거하고 스테이지 영역에서도 지우기

히스토리와 상태 검토하기 (자세히 알아보기: git help revisions)

bisect 바이너리 서치로 버그 찾기

grep 패턴과 매칭된 줄 출력

log 커밋 로그 보기

show 다양한 타입의 오브젝트 보기

status 현재 작업중인 트리의 상태 보기

히스토리 수정, 마크, 추가하기

branch 브랜치 리스트 보기, 브랜치 만들기 / 지우기

checkout 브랜치 바꾸기 또는 작업중인 트리의 파일들 복구하기

commit 로컬 저장소에 수정사항 반영하기

diff 커밋 또는 작업중인 트리의 변동사항 보기 등

merge 2개 혹은 그 이상의 개발 히스토리 합치기

rebase 다른 베이스 브랜치의 최상단에서 커밋 재적용하기 (부모 브랜치 바꾸기)

tag GPG 사인과 함께 태그 만들기, 보기, 삭제하기 등

협업하기 (자세히 알아보기: git help workflows)

fetch 다른 브랜치로부터 오브젝트(object)와 레퍼런스(refs) 다운로드하기

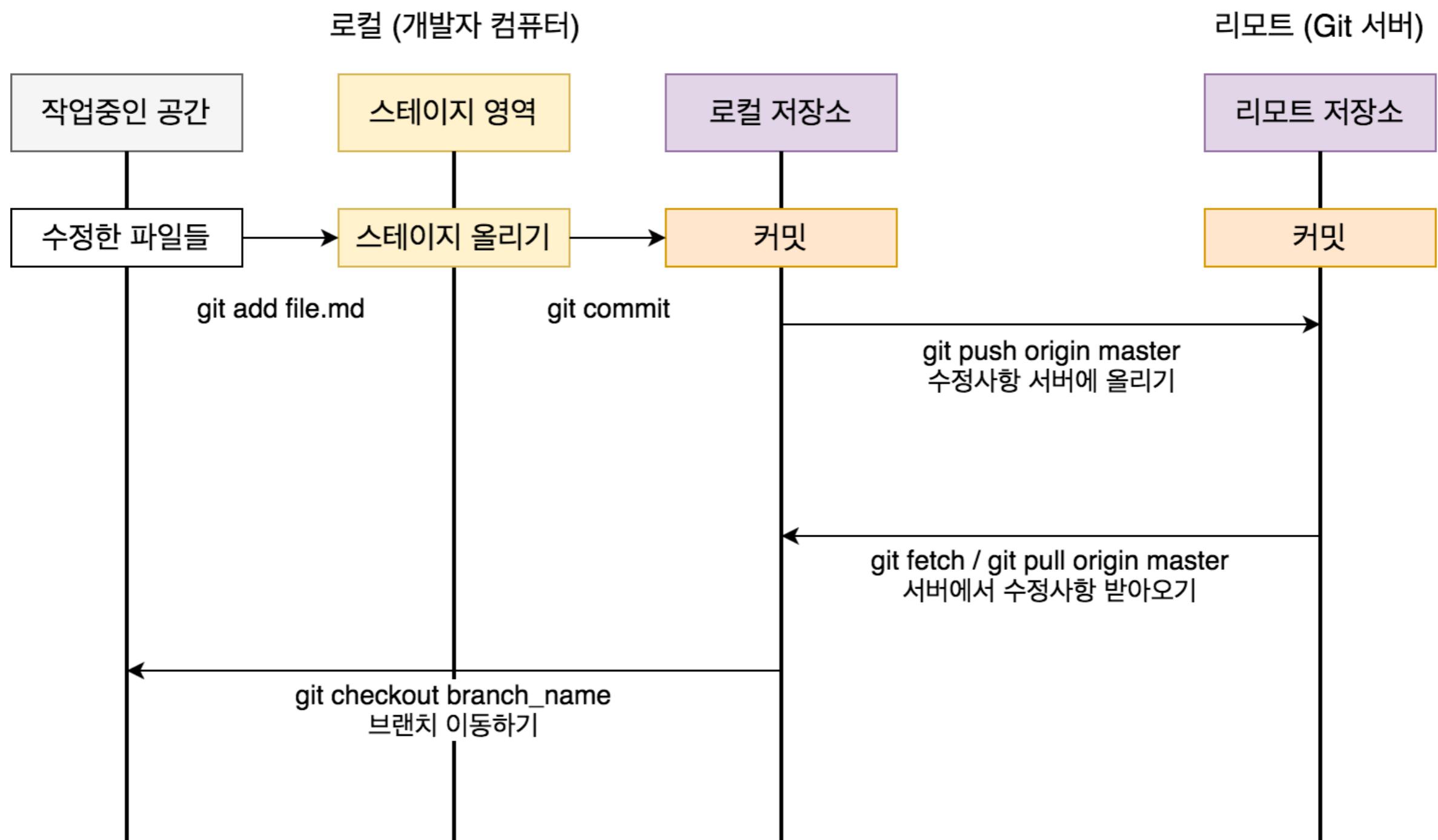
pull 로컬 브랜치 또는 다른 브랜치로부터 fetch 받아서 적용하기

push 관련된 오브젝트(object)로 리모트 레퍼런스(refs)를 업데이트 하기

마지막은 실제로 협업을 하기 위한 명령어입니다.

앞서 소개해 드린 명령어는 로컬 저장소에서만 의미가 있어요.

원격 저장소(리모트 저장소)에 push 하여 반영하기 전까지는 그 어떤 작업을 하든 원격 저장소에 반영이 되지 않습니다.



앞서 설명드린 기초 명령어들의 사용 흐름도를 그래프로 보여드리면 위와 같습니다.
자세한 흐름은 실습시간에 다시 설명드릴거에요.



```
function COFFEE.BREAK()
    for k, ply in pairs(GetAll()) do
        local drink = ply:CB()
        if not ply:Awake() then
            local strength = 1000

            if strength > k then
                local sugar = 2
                local milk = 1
                local extrashots = 2
                if postcoffee == not ply:Awake
                    local new = strength^2
                else return end
            end
        end
    end
```

그럼 잠시 10분간 쉬는시간을 가지고, 이후로는 본격적으로 실습을 해보도록 하겠습니다.

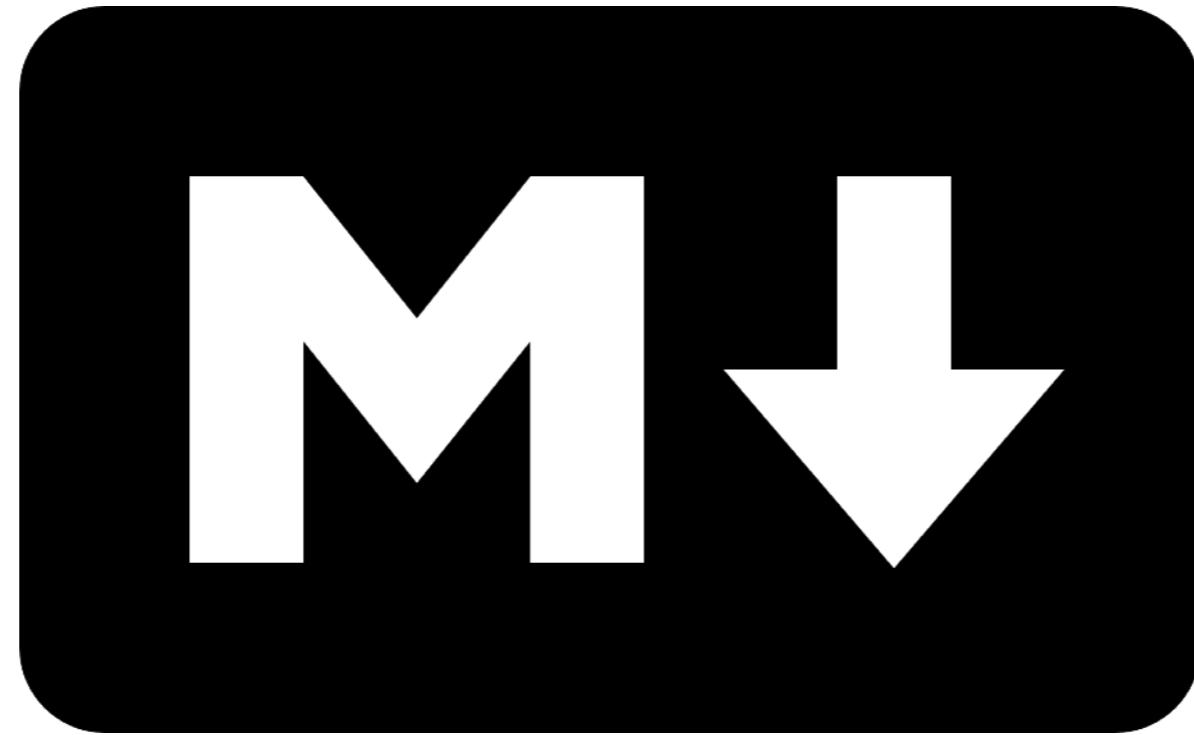
Learn by Doing

자, 그럼 이제 실습을 진행해볼까요?

사실 직접 해보기 전까지는 감이 잘 안 오실거에요. 뭐든 직접 해보는 게 가장 좋더라고요.

1. 저장소 초기화 / 환경설정
2. 브랜치 만들기
3. 커밋하기
4. 커밋 비교해보기 (diff)
5. 현재 작업 위치(HEAD) 이동하기
6. 다른 브랜치 만들기
7. 브랜치 비교해보기 (diff)
8. 충돌발생 시 해결방법
9. 풀 리퀘스트로 다른 사람의 저장소에 기여하기 (fork / pull-request)

실습은 다음과 같이 진행을 할 예정입니다.



마크다운 문서 작성

실습은 특정 언어의 코드 대신, 마크다운 문서로 진행을 할 예정입니다.
명령어를 직접 입력해보면서, 입력했던 명령어들을 마크다운으로 정리를 해볼거에요.

The screenshot shows a GitHub page for the repository `moby/moby`. The URL is <https://github.com/moby/moby/blob/master/README.md>. The page displays the `Code` tab of the README file. The file is titled `moby / README.md` and was last updated by `yongtang` on Oct 10, 2017. It contains 58 lines (39 sloc) and is 3.32 KB in size. The content of the README file is as follows:

```

The Moby Project


moby  

project

Moby is an open-source project created by Docker to enable and accelerate software containerization.

It provides a "Lego set" of toolkit components, the framework for assembling them into custom container-based systems, and a place for all container enthusiasts and professionals to experiment and exchange ideas. Components include container build tools, a container registry, orchestration tools, a runtime and more, and these can be used as building blocks in conjunction with other tools and projects.

Principles

```

가이드 : <https://guides.github.com/features/mastering-markdown/>

다음은 Github 에 있는 Ruby 라는 언어의 README.md 인데요.
 md 란 마크다운의 확장자입니다. rst 나 기타 포맷들도 많지만 요즘엔 md 를 많이 쓰는 추세입니다.

저장소 만들기

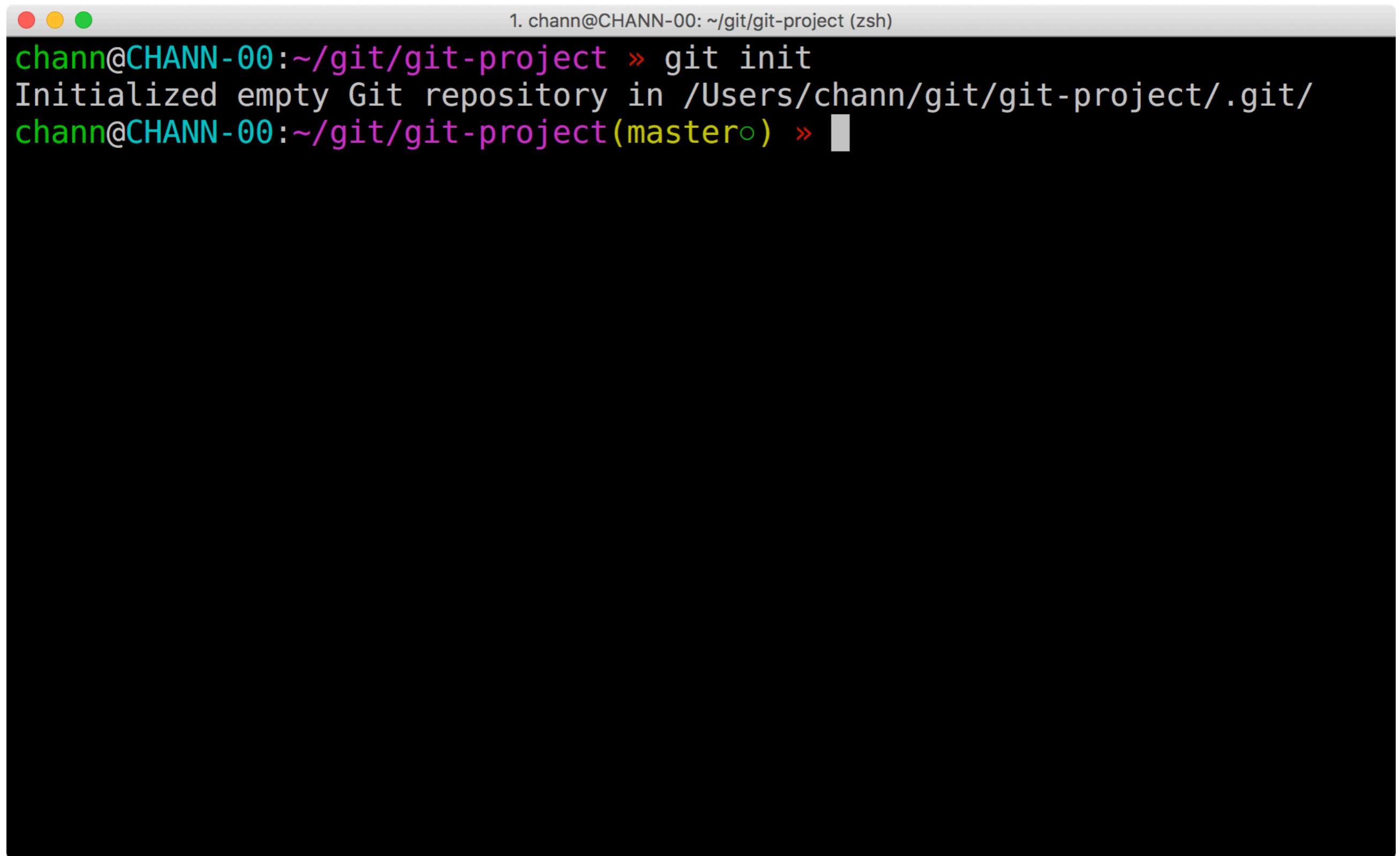
자, 그럼 본격적인 실습을 시작해볼게요. 먼저 저장소가 있어야겠죠?
저장소부터 만들어보도록 할게요.

Windows : C:\git

macOS, Linux : ~/git/

원활한 실습을 위해 작업환경의 위치를 통일하도록 하겠습니다.

위와 같은 경로로 통일해주시기를 부탁드릴게요. macOS, Linux 의 ~ 는 사용자 홈 폴더를 의미해요.



A screenshot of a macOS terminal window. The title bar says "1. chann@CHANN-00: ~/git/git-project (zsh)". The terminal shows the command "git init" being run in a directory named "git-project". The output indicates that an empty Git repository was initialized in the specified path. The prompt then changes to "(master)".

```
chann@CHANN-00:~/git/git-project » git init
Initialized empty Git repository in /Users/chann/git/git-project/.git/
chann@CHANN-00:~/git/git-project(master) »
```

작업환경으로 사용하시는 위치에 git-project 이라는 이름의 폴더를 만드시고,
그 폴더에서 위와 같이 명령어를 입력해주세요.

```

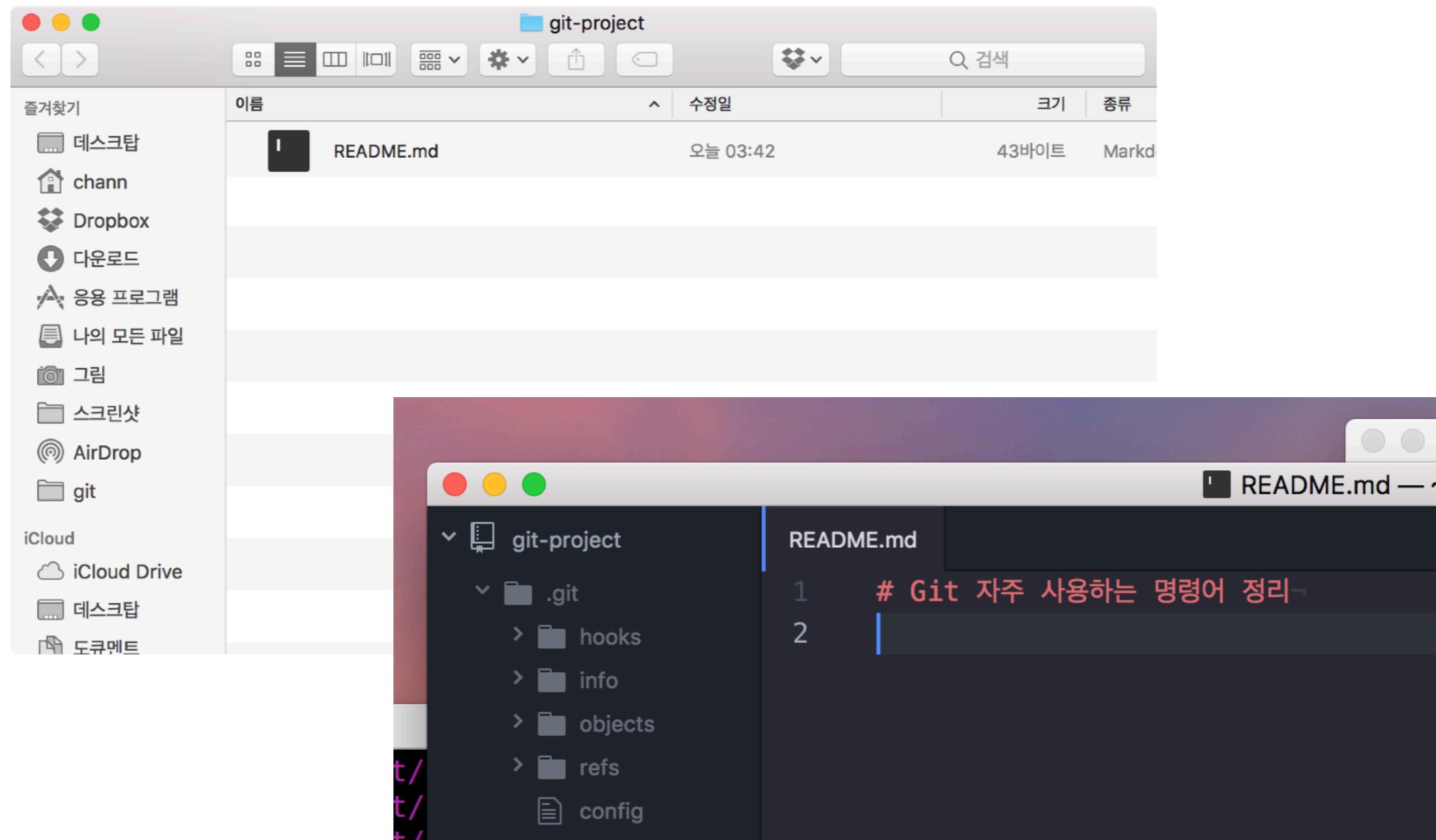
1. chann@CHANN-00: ~/git/git-project (zsh)
drwxr-xr-x  3 chann  staff   102B  4 15 02:44 .
drwxr-xr-x 10 chann  staff   340B  4 14 04:52 ..
drwxr-xr-x  9 chann  staff   306B  4 15 02:47 .git
chann@CHANN-00:~/git/git-project(master) » tree .git
.git
├── HEAD
├── config
├── description
└── hooks
    ├── applypatch-msg.sample
    ├── commit-msg.sample
    ├── post-update.sample
    ├── pre-applypatch.sample
    ├── pre-commit.sample
    ├── pre-push.sample
    ├── pre-rebase.sample
    ├── pre-receive.sample
    ├── prepare-commit-msg.sample
    └── update.sample
├── info
│   └── exclude
└── objects
    ├── info
    └── pack
└── refs
    ├── heads
    └── tags

8 directories, 14 files
chann@CHANN-00:~/git/git-project(master) »

```

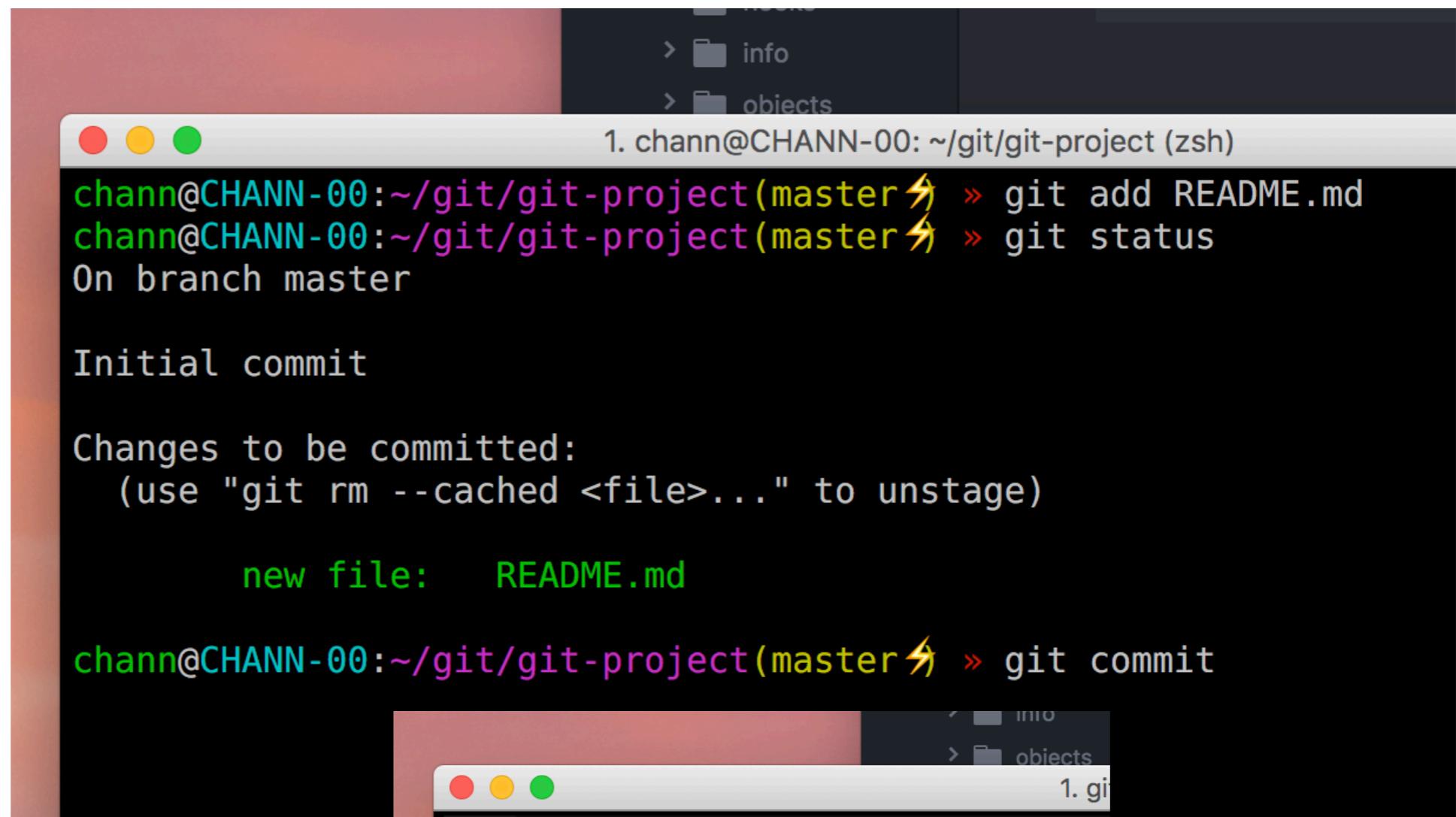
git init

.git 이라는 이름의 폴더가 생겼다면 정상적으로 저장소가 생긴거에요.
이 폴더는 로컬 저장소의 메타데이터 정보를 가지고 있어요.



git init

저장소가 생기긴 했지만, 사실 커밋이 하나도 없는 껍데기에 불과합니다.
 시작점(기준)이 될 만한 커밋을 하나 만들어주셔야 해요.
 README.md 파일을 추가하시고, 위와같이 제목을 입력해주세요.



```

> info
> objects
1. chann@CHANN-00: ~/git/git-project (zsh)
chann@CHANN-00:~/git/git-project(master⚡) » git add README.md
chann@CHANN-00:~/git/git-project(master⚡) » git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   README.md

chann@CHANN-00:~/git/git-project(master⚡) » git commit

```

git commit

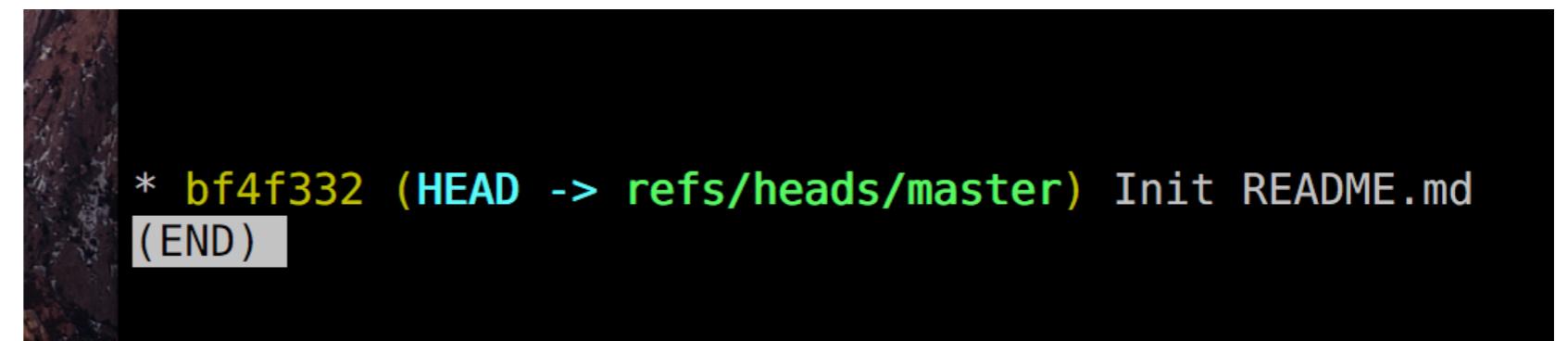
↑

git add README.md

↑

git init

커밋을 할 파일이 만들어졌으니 이제 첫 번째 커밋을 해볼까요!
 왼쪽의 명령어 순서대로 진행해보세요. status 명령어는 현재 상태를 보여줍니다.



```
* bf4f32 (HEAD -> refs/heads/master) Init README.md  
  (END)
```

```
git log --decorate=full --oneline --graph
```



```
git commit
```

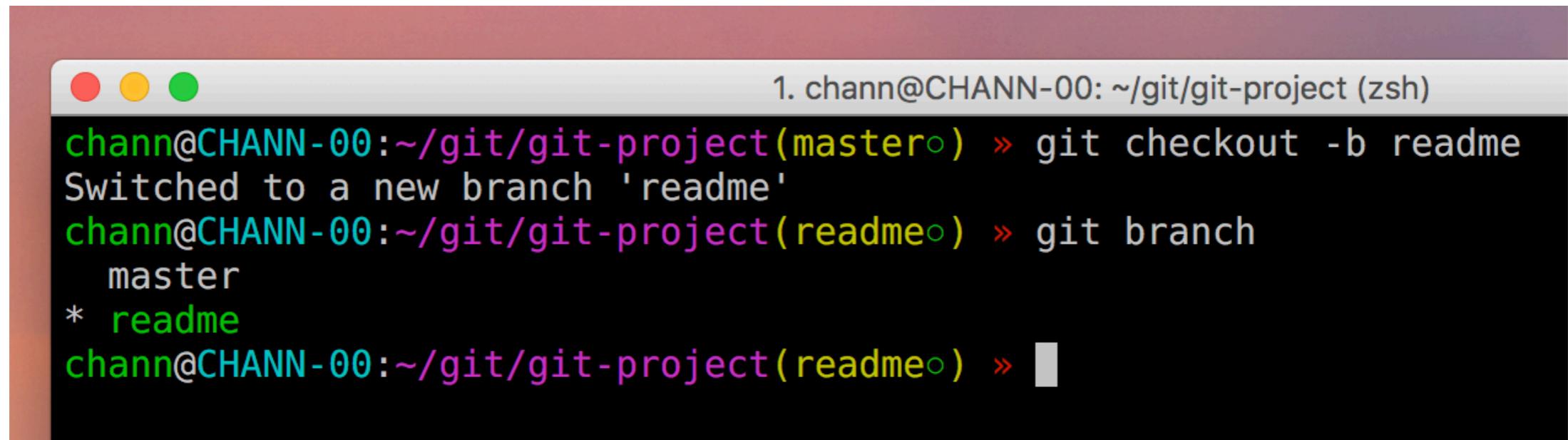


```
git add README.md
```



```
git init
```

커밋이 잘 되었는지 확인을 한번 하고 넘어갈까요?
위와 같이 명령어를 입력해서 확인하고 넘어가시면 됩니다.



```

1. chann@CHANN-00: ~/git/git-project (zsh)
chann@CHANN-00:~/git/git-project(master) » git checkout -b readme
Switched to a new branch 'readme'
chann@CHANN-00:~/git/git-project(readme) » git branch
  master
* readme
chann@CHANN-00:~/git/git-project(readme) »

```

git checkout -b readme



git log --decorate=full --oneline --graph



git commit



git add README.md



git init

이제 readme라는 브랜치를 만들어볼거에요.

브랜치 전략에 따라 다를 순 있지만, 브랜치는 대개 만들고자 하는 기능의 단위로 많이들 사용합니다.

왼쪽 노란색 박스를 참고해서 명령어를 입력해주시고 위의 스크린샷을 참고하여 생성된 브랜치로 이동이 된 상태인지 확인을 해주세요.

```
* bf4f332 - (HEAD -> readme, master) Init README.md (29 minutes ago) <CHANN>  
(END)
```

HEAD -> readme

master

같은..위치? HEA..D..??

readme라는 브랜치를 만들 때 master 브랜치로부터 checkout -b 옵션으로 만들었죠?

그러므로 master 브랜치는 readme의 부모 브랜치입니다.

그리고 HEAD 표식은 현재 위치를 나타내는 것입니다.

master 브랜치로부터 나왔으나 별다른 수정사항이 없으므로 같은 위치에 있는 거에요.

The screenshot shows a terminal window with a dark theme. On the left is a file tree for a project named 'git-project'. The 'README.md' file is selected. The main pane displays the contents of 'README.md' which contains a list of git commands in Korean. The bottom status bar shows the file name 'README.md', line count '0 0 0', and time '20:4'. Other status indicators include 'LF', '78 W | 372 C', 'UTF-8', 'GitHub Markdown', 'readme', '+19', 'git+', '2 updates', and a gear icon.

```

# Git 자주 사용하는 명령어 정리
자주 사용하는 기본적인 명령어들을 정리했습니다.

## 저장소 생성(초기화) 하기
```
bash
git init # 저장소 생성
```

## 커밋하기
```
bash
git add README.md # 커밋할 파일 추가
git status # 현재 스테이지 상태를 확인
git commit -m "init: README.md" # 인라인 커밋 메시지 작성
git log --decorate=full --oneline --graph # 커밋 로그 확인
```

## 브랜치 생성하기
```
bash
git checkout -b readme # readme라는 이름의 브랜치를 생성한 후, 생성된 브랜치로 체크아웃
```

```

git commit -m "add: 자주 사용하는 명령어들 추가"

master

HEAD -> readme

자.. 그럼 이어서 계속해볼까요?

아까와 동일하게 커밋을 추가할거에요. 브랜치가 바뀌었지만 커밋을 하는 방법은 동일합니다.

이번에는 인라인 메시지 옵션을 줘서 진행해볼게요. 커밋할 파일 스테이지 추가 하시고, 위와 같이 커밋해보세요

파일 내용은 위와 같이 방금 우리가 실습한 내용 중 기록하고 싶은 내용을 적어주세요.

```
# Git 자주 사용하는 명령어 정리
자주 사용하는 기본적인 명령어들을 정리했습니다.

## 저장소 생성(초기화) 하기
```bash
git init # 저장소 생성
```

## 커밋하기
```bash
git add README.md # 커밋할 파일 추가
git status # 현재 스테이지 상태를 확인
git commit -m "init: README.md" # 인라인 커밋 메시지 작성
git log --decorate=full --oneline --graph # 커밋 로그 확인
```

## 브랜치 생성하기
```bash
git checkout -b readme # readme라는 이름의 브랜치를 생성한 후, 생성된 브랜치로 체크아웃
```

```

git commit -m "add: 자주 사용하는 명령어들 추가"

master

HEAD -> readme

좋은 커밋메시지 작성법 : <http://www.haruar.com/blog/2738>

제가 입력한 내용은 위와 같습니다.

참고로, 커밋 메시지는 작업한 내용의 요약을 적는 것입니다.

나중에 누가 봤을 때 커밋 메시지만 봐도 '아 이런 작업 했구나' 하고 알 정도로 신경써서 작성해야 좋아요.

```

git-project
  .git
  README.md

# Git 자주 사용하는 명령어 정리
자주 사용하는 기본적인 명령어들을 정리했습니다.
- 최근 수정일 : 2017-04-15

## 저장소 생성(초기화) 하기
```bash
git init # 저장소 생성
```

```

```

chann@CHANN-00:~/git/git-project(readme⚡) » git cm "add: 최근 수정일 추가"
[readme 89cf54b] add: 최근 수정일 추가
 1 file changed, 1 insertion(+)
chann@CHANN-00:~/git/git-project(readme○) »

```

git commit -m "add: 최근 수정일 추가"

git commit -m "add: 자주 사용하는 명령어들 추가"

master

HEAD -> readme

이젠 커밋이 슬슬 익숙하시죠?
최근 수정일을 추가한 뒤 커밋을 한번 더 해보겠습니다.

```
* 89cf54b - (HEAD -> readme) add: 최근 수정일 추가 (22 minutes ago) <CHANN>
* ee22b56 - add: 자주 사용하는 명령어들 추가 (36 minutes ago) <CHANN>
* bf4f332 - (master) Init README.md (2 hours ago) <CHANN>
(END)
```

HEAD -> readme

```
git commit -m "add: 최근 수정일 추가"
```

```
git commit -m "add: 자주 사용하는 명령어들 추가"
```

master

```
git commit -m "Init README.md"
```

여기까지 순조롭게 잘 따라와주셨다면

다음과 같이 master로부터 뺌어나온 readme 브랜치에 커밋이 2개 추가된 모습을 보실 수 있습니다.

diff - 달라진 점만 골라서 보기

```
git diff README.md # README.md 파일의 최근 변경사항 비교
```

```
git diff master readme # master 브랜치를 기준으로 readme 브랜치와 비교
```

```
git diff bf4f332 89cf54b # 커밋의 hash 값으로 커밋 대 커밋 비교
```

```
git diff HEAD~2 # 현재 위치(HEAD)로부터 2번째 이전 커밋과 비교
```

자세히 정리된 글 : <http://dogfeet.github.io/articles/2011/git-diff.html>

HEAD -> readme

```
git commit -m "add: 최근 수정일 추가"
```

```
git commit -m "add: 자주 사용하는 명령어들 추가"
```

master

```
git commit -m "Init README.md"
```

이제 머지를 해볼건데요.

**그 전에 먼저 diff 의 대표적인 명령어들을 알려드리고 넘어가도록 하겠습니다.
머지해서 합치기 전에 합칠 대상으로부터 뭐가 다른지 비교는 해보고 넘어가야겠죠?**

1. chann@CHANN-00: ~/git/git-project (zsh)

```
chann@CHANN-00:~/git/git-project(readme) » git checkout master
Switched to branch 'master'
chann@CHANN-00:~/git/git-project(master) »
```

git checkout master

HEAD -> readme

git commit -m "add: 최근 수정일 추가"

git commit -m "add: 자주 사용하는 명령어들 추가"

master

git commit -m "Init README.md"

우선, 머지를 하기 위해서는 변경사항을 받아들일 브랜치로 이동을 해야 합니다.

우리는 readme 브랜치를 master 로 합칠 거니까 master 브랜치로 이동을 해야겠죠?

그럼 위와 같이 readme 에서 작업하고 커밋했던 내용들이 사라집니다. master 로 이동했기 때문이죠.

```

1. chann@CHANN-00:~/git/git-project (zsh)
chann@CHANN-00:~/git/git-project(readme) » git checkout master
Switched to branch 'master'
chann@CHANN-00:~/git/git-project(master) » git merge readme --no-ff
Merge made by the 'recursive' strategy.
 README.md | 20 ++++++-----+
 1 file changed, 20 insertions(+)
chann@CHANN-00:~/git/git-project(master) » git lg

```

git checkout master

readme

git commit -m "add: 최근 수정일 추가"

git commit -m "add: 자주 사용하는 명령어들 추가"

git merge readme --no-ff

HEAD → master

git commit -m "Init README.md"

그럼 머지를 진행해볼까요.

위의 --no-ff 옵션은 fast-forward라는 기능을 끄는 겁니다.

fast-forward는 줄을 자동으로 최대한 한 줄로 만들어주는 기능이라고 보시면 되는데,
그렇게 되면 공부하는 저희 입장에서는 대개 한 줄로만 보이기 때문에 일부러 끄는 거에요.

실제 개발할 때 저 기능을 끄면 엄청난 수의 라인을 보실 수 있습니다..

```
* 380c25a - (HEAD -> master) Merge branch 'readme' (2 minutes ago) <CHANN>
|\ 
* 89cf54b - (readme) add: 최근 수정일 추가 (51 minutes ago) <CHANN>
* ee22b56 - add: 자주 사용하는 명령어들 추가 (65 minutes ago) <CHANN>
|
* bf4f332 - Init README.md (2 hours ago) <CHANN>
(END)
```

HEAD -> master

Merge branch 'readme'

readme

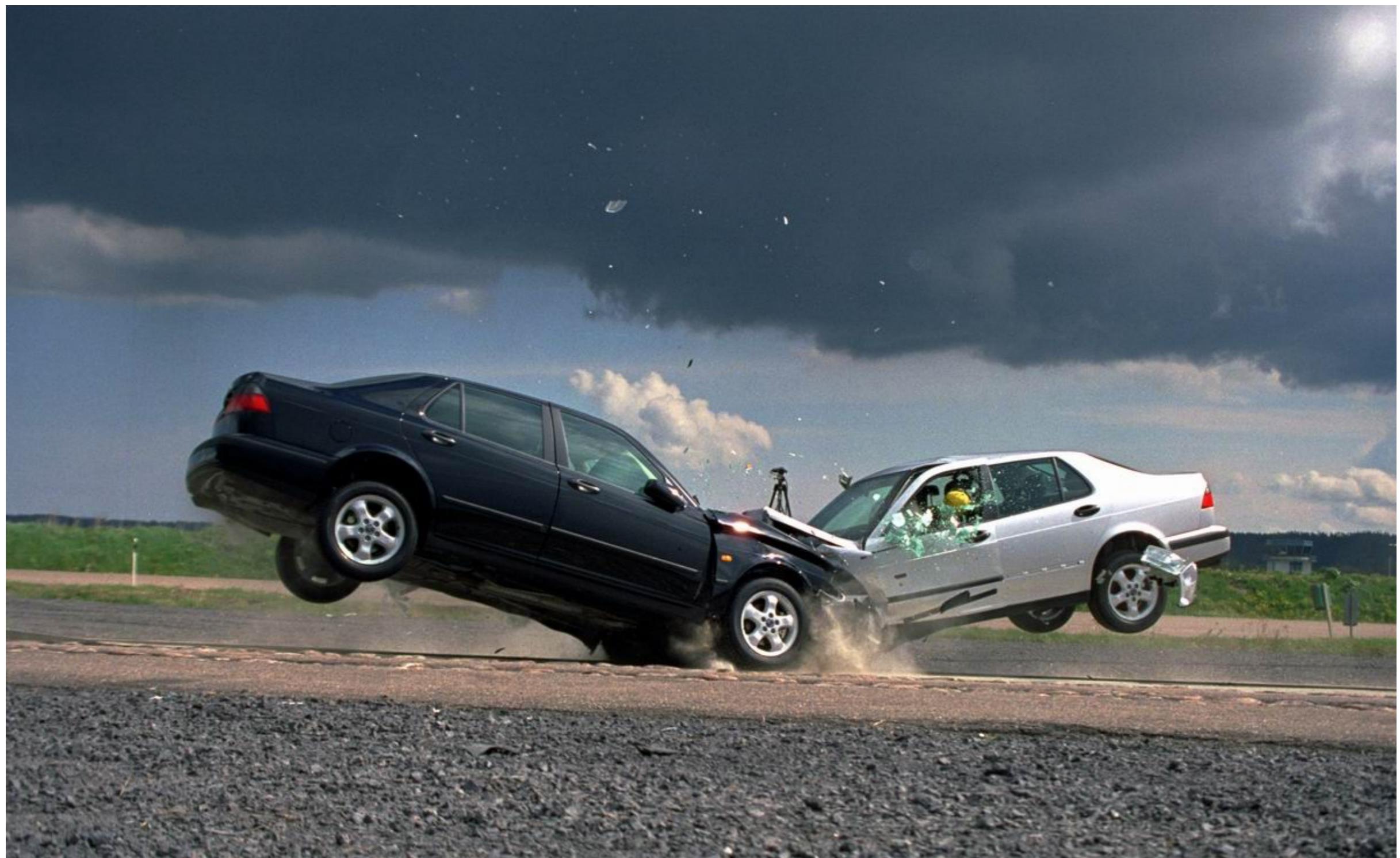
git commit -m "add: 최근 수정일 추가"

git commit -m "add: 자주 사용하는 명령어들 추가"

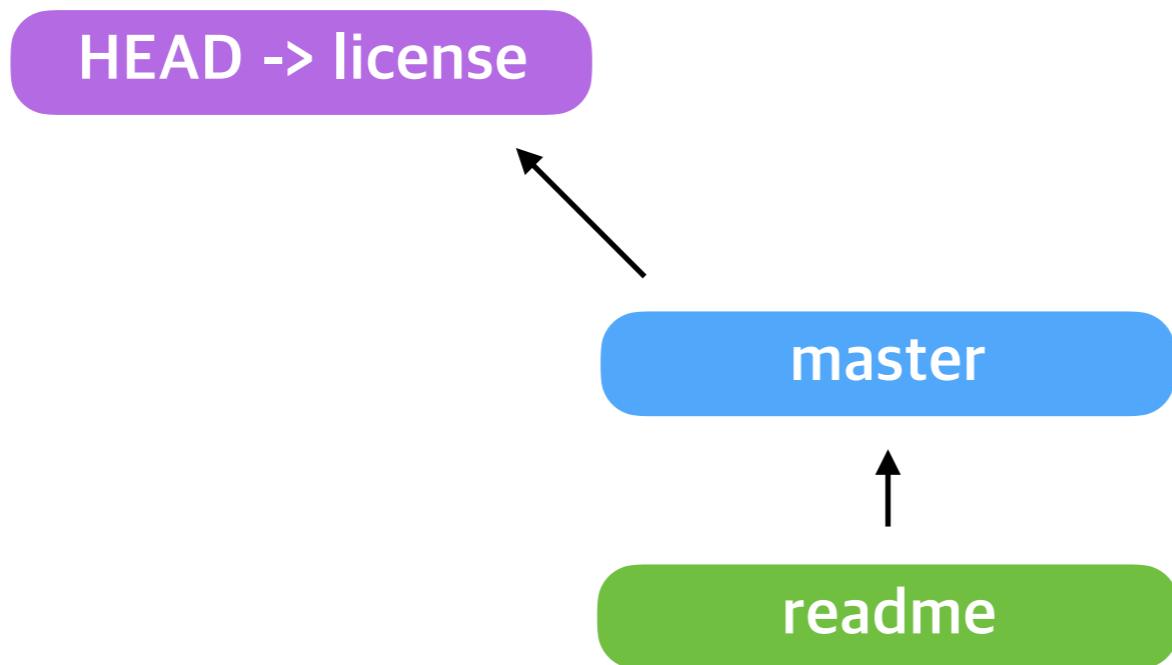
git merge readme --no-ff

git commit -m "Init README.md"

머지를 진행한 후 log 를 보시면 다음과 같이 바뀐 것을 확인하실 수 있습니다.



자, 그럼 이제 매번 터질 때마다 스릴이 넘치는 git 충돌실험을 진행해보겠습니다.



먼저, 충돌이 일어나는 간단한 시나리오를 먼저 설명해드릴게요.
현재 저희는 readme 위에 readme 를 머지하여 최신 상태가 된 master 가 있잖아요?
여기서 license 브랜치를 만들어서 뺀어나갈 겁니다.

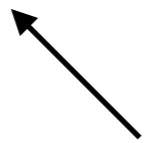
update: README.md

add: LICENSE

HEAD -> license

master

readme



그런 다음, 라이센스 정책 안내를 위해 LICENSE 파일을 추가할 것이고
라이센스가 추가되었다는 점을 README.md 에 반영할 거에요.

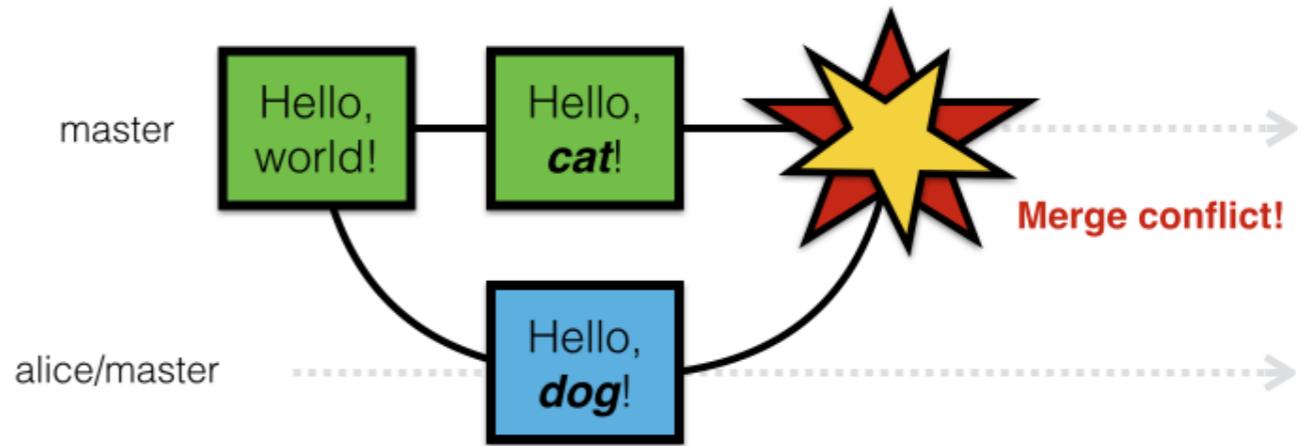
update: License 관련 안내
add: LICENSE 파일 추가

license

hotfix: 머지 관련 명령어 추가

HEAD -> master

readme



그런데 안타깝게도 master에 머지 관련 노트를 적어버렸어요.
머지를 하려고 보니까 master 브랜치가 달라져서 충돌이 나버렸네요.
… 사실 이런 경우는 여러 명이서 개발을 할 경우엔 흔하게 일어납니다.



골 아픈 상황이긴 한데.. 자.. 실습을 해볼까요?
(고라파덕!)

channprj / git-project Private

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Quick setup — if you've done this kind of thing before

저장소 주소

Set up in Desktop or HTTPS SSH git@github.com:channprj/git-project.git

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# git-project" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:channprj/git-project.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:channprj/git-project.git
git push -u origin master
```

그 전에… 미리 Github 저장소부터 연결해서 푸시를 해둘까요?

Github에서 다음과 같이 git-project라는 이름으로 저장소를 만들어주세요.

README.md, LICENSE 없음, 그리고 공개 저장소로 설정해서 만들어주시면 마우스 몇 번 클릭으로 끝납니다.

```
chann@CHANN-00:~/git/git-project(master) » git remote --help
chann@CHANN-00:~/git/git-project(master) » git remote add origin https://github.com/channprj/git-project.git
chann@CHANN-00:~/git/git-project(master) » git remote -v
origin  https://github.com/channprj/git-project.git (fetch)
origin  https://github.com/channprj/git-project.git (push)
chann@CHANN-00:~/git/git-project(master) » █
```

git remote add origin 저장소주소

명령어
(리모트)

명령어
(추가) (리모트 저장소 이름)

인자1
(저장소 이름)

인자2
(저장소 주소)

```
git remote add origin 저장소주소
```

```
git remote --help # git remote 명령어 도움말
```

```
git remote -v # 현재 연결된 저장소 확인
```

그런 다음.. 현재 연결된 저장소 확인을 해보면.. 당연히 있을리가 없죠.
저장소를 추가(연결) 해볼까요? 다음과 같이 명령어를 입력해보세요.

git push origin master

Git 밀어넣는다 오리진 마스터로

```
git push origin master
```

```
git checkout master
```

```
git remote add origin 저장소주소
```

```
git remote --help # git remote 명령어 도움말
```

```
git remote -v # 현재 연결된 저장소 확인
```

다 되었으면 마스터 브랜치로 이동하여 리모트 저장소에 마스터 브랜치로 푸시를 해보세요.

This repository Search Pull requests Issues Gist

channprj / git-project Private

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

No description, website, or topics provided. Edit Add topics

4 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

channprj Merge branch 'readme' Latest commit 380c25a an hour ago

README.md add: 최근 수정일 추가 2 hours ago

README.md

Git 자주 사용하는 명령어 정리

자주 사용하는 기본적인 명령어들을 정리했습니다.

- 최근 수정일 : 2017-04-15

저장소 생성(초기화) 하기

```
git init # 저장소 생성
```

커밋하기

예쁘게 잘 들어갔죠?
이 맛에 깃헙 하는 거죠. (...)

WTFPL : <http://www.wtfpl.net/about/>

HEAD -> license

```
git checkout -b license
```

master

```
Merge branch 'readme'
```

readme

```
git commit -m "add: 최근 수정일 추가"
```

```
git commit -m "add: 자주 사용하는 명령어들 추가"
```

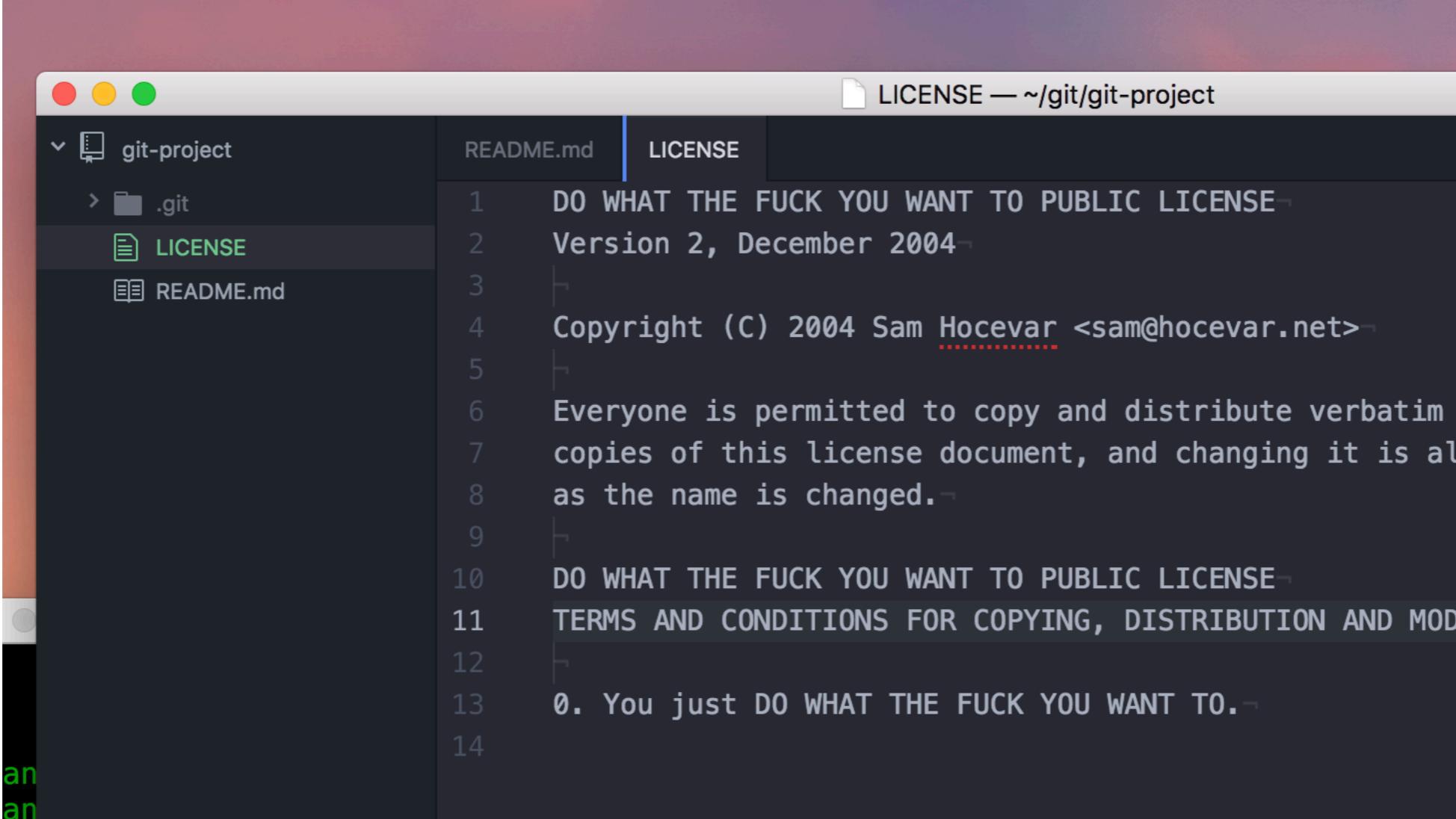
```
git merge readme --no-ff
```

```
git commit -m "Init README.md"
```

자.. 그럼 다시 돌아와서, 충돌 실험(?)을 계속 해볼까요?

우선, 다음과 같이 라이센스 브랜치를 따고, 라이센스 파일을 넣고, README.md 를 수정할거에요.

저는 MIT, Beerware, Coffeeware, WTFPL 라이센스를 좋아합니다.



LICENSE — ~/git/git-project

```
git-project
  ↘ .git
  LICENSE
  README.md
```

1 DO WHAT THE FUCK YOU WANT TO PUBLIC LICENSE
2 Version 2, December 2004
3
4 Copyright (C) 2004 Sam Hocevar <sam@hocevar.net>
5
6 Everyone is permitted to copy and distribute verbatim
7 copies of this license document, and changing it is al
8 as the name is changed.
9
10 DO WHAT THE FUCK YOU WANT TO PUBLIC LICENSE
11 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODI
12
13 0. You just DO WHAT THE FUCK YOU WANT TO.

이렇게 마음에 드는 라이센스 파일을 넣고,

The screenshot shows a terminal window with a dark theme. The title bar says "README.md — ~/git/git-project". The left sidebar shows a file tree with "git-project" expanded, containing ".git", "LICENSE", and "README.md". The main pane displays the content of "README.md". The content is a GitHub Markdown file with Korean annotations:

```
8  ````  
9  |  
10 | ## 커밋하기  
11 | ````bash  
12 | git add README.md # 커밋할 파일 추가  
13 | git status # 현재 스테이지 상태를 확인  
14 | git commit -m "init: README.md" # 인라인 커밋 메시지  
• 작성  
15 | git log --decorate=full --oneline --graph # 커밋  
• 로그 확인  
16 | ````  
17 | |  
18 | ## 브랜치 생성하기  
19 | ````bash  
20 | git checkout -b readme # readme라는 이름의 브랜치를  
• 생성한 후, 생성된 브랜치로 체크아웃  
21 | ````  
22 | |  
23 | # LICENSE  
24 | 이 저장소는 WTFPL 라이선스에 의해 보호를 받습니다.  
25
```

At the bottom, there is a status bar with "README.md", file statistics (0 0 0), the current time (24:32), and icons for LF, 92 W | 437 C, UTF-8, GitHub Markdown, license, +3, git+, and a settings icon.

README.md. 하단에 라이선스 보호 내용을 적어주세요.

```

1. chann@CHANN-00: ~/git/git-project (zsh)
* [new branch]      master -> master
chann@CHANN-00:~/git/git-project(master) » git lg
chann@CHANN-00:~/git/git-project(master) » git co -b license
Switched to a new branch 'license'
chann@CHANN-00:~/git/git-project(license) »
chann@CHANN-00:~/git/git-project(license⚡) »
chann@CHANN-00:~/git/git-project(license⚡) » git st
On branch license
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    LICENSE

no changes added to commit (use "git add" and/or "git commit -a")
chann@CHANN-00:~/git/git-project(license⚡) » git add LICENSE
chann@CHANN-00:~/git/git-project(license⚡) » git commit -m "add: 라이선스 추가"
[license 3cbaf66] add: 라이선스 추가
  1 file changed, 13 insertions(+)
  create mode 100644 LICENSE
chann@CHANN-00:~/git/git-project(license⚡) » git add README.md
chann@CHANN-00:~/git/git-project(license⚡) » git commit -m "update: 라이선스 추가에 따른 처리"
[license 3e9e6c6] update: 라이선스 추가에 따른 처리
  1 file changed, 3 insertions(+)
chann@CHANN-00:~/git/git-project(license) »

```

git add LICENSE

git commit -m "add: 라이선스 추가"

git add README.md

git commit -m "update: 라이선스 추가에 따른 처리"

그리고 이렇게 커밋을 하시고요.

license

```
git checkout -b license
```

HEAD -> master

```
Merge branch 'readme'
```

readme

```
git commit -m "add: 최근 수정일 추가"
```

```
git commit -m "add: 자주 사용하는 명령어들 추가"
```

```
git merge readme --no-ff
```

```
git commit -m "Init README.md"
```

다시 돌아와서…
master 브랜치로 이동을 하세요.

```
          ↴  
## 머지하기  
```bash  
git merge readme --no-ff # readme 브랜치의 내용을 가져와서 머지함,
fast-forward 가능 끄기
```  
• LF 100 W | 488 C UTF-8 GitHub Markdown ⚡ master  
connect full online search ↵ git commit pull request history
```

```
## 머지하기  
```bash  
git merge readme --no-ff # readme 브랜치의 내용을 가져와서 머지함, fast-forward 가능 끄기
```
```

그리고 README.md 맨 하단에 위의 내용을 넣어주세요.
그리고 이젠 커밋하시는 건 아시죠?

```
* 1e718e9 - (HEAD -> master) hotfix: 누락된 머지 관련 내용 추가 (16 seconds ago) <CHANN>
| * 3e9e6c6 - (license) update: 라이선스 추가에 따른 처리 (8 minutes ago) <CHANN>
| * 3cbaf66 - add: 라이선스 추가 (8 minutes ago) <CHANN>
|
* 380c25a - (origin/master) Merge branch 'readme' (66 minutes ago) <CHANN>
```

그럼 이렇게 브랜치가 갈라지기 시작해요.
자, 이제 머지 들어가세요!

18 ## 브랜치 생성하기

1. chann@CHANN-00: ~/git/git-project (zsh)

```
chann@CHANN-00:~/git/git-project(master) » git merge license
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
chann@CHANN-00:~/git/git-project(master) »
```

CONFlict! CONFlict! CONFlict!



인내심의 한계를 경험 중

네, 당연히 충돌이 났습니다. (...)
이걸 어떻게 고쳐야 할까요...?

```

22
23 <<<<< HEAD
24 ## 머지하기
25 `` `bash
26 git merge readme --no-ff # re
fast-forward 가능 끄기
27 `` `
28 =====
29 # LICENSE
30 이 저장소는 WTFPL 라이선스에 의해 보호를 받습니다.
31 >>>>> license
32

```

```

chann@CHANN-00:~/git/git-project(master⚡) » git
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:

          new file:   LICENSE

Unmerged paths:
  (use "git add <file>..." to mark resolution)

          both modified: README.md

chann@CHANN-00:~/git/git-project(master⚡) »

```

• LF 114 W | 567 C UTF-8 GitHub Markdown

git status # 작업 디렉토리의 상태를 보여주기

**위의 명령어로 보니까 README.md 가 both modified 됐다고 하면서
머지가 되지 않았습니다… 수정사항이 겹친 파일을 누군가 합쳐서 넣으면 되지 않을까요..?**

```

*   871ca35 - (HEAD -> master) fix: 머지 오류 잡기 (4 minutes ago) <CHANN>
| \
| * 3e9e6c6 - (license) update: 라이선스 추가에 따른 처리 (26 minutes ago) <CHANN>
| * 3cbaf66 - add: 라이선스 추가 (26 minutes ago) <CHANN>
* | 1e718e9 - hotfix: 누락된 머지 관련 내용 추가 (18 minutes ago) <CHANN>
| /
* 380c25a - (origin/master) Merge branch 'readme' (84 minutes ago) <CHANN>
| \
| * 89cf54b - (readme) add: 최근 수정일 추가 (2 hours ago) <CHANN>
| * ee22b56 - add: 자주 사용하는 명령어들 추가 (2 hours ago) <CHANN>
| /
* bf4f332 - Init README.md (3 hours ago) <CHANN>
(END)

```

```

20      git checkout -b readme # readme라는 이름의 브랜치를 생성한 후, 생성된 브
      •   로 체크아웃
21      ````-
22      |
23      ## 머지하기-
24      ````bash-
25      git merge readme --no-ff # readme 브랜치의 내용을 가져와서 머지함,
      •   fast-forward 가능 끄기-
26      ````-
27      |
28      # LICENSE-
29      이 저장소는 WTFPL 라이선스에 의해 보호를 받습니다.-_
30

```

git cm "fix: 머지 오류 잡기" # 인라인 커밋

git add . # 전체 변경파일들 추가

**내용을 확인해보니까 다 필요한 내용들이어서
이렇게 코드를 합쳐서 커밋을 했더니 해결이 되는군요.**

이렇듯, 충돌 자체는 짜증나는 일이지만 원인만 파악이 되면 금방 해결할 수 있는 문제입니다.



이제 슬슬 실습도 끝나가네요.
하지만… 아직 저희에겐 풀 리퀘스트 하나가 남아 있습니다.

channprj / git-starter

Code Issues Pull requests Projects Wiki Pulse Graphs Settings

Unwatch 1 Unstar 1 Fork 0

Git, Github Hands on Lab for Starter @ OSS Dev Forum <http://bit.ly/git-starter>

study git starter newbie Manage topics

8 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

channprj committed on GitHub update: 함께 하신 분들 목록에 조교님들 추가 Latest commit c9aa9f7 6 minutes ago

README.md update: 함께 하신 분들 목록에 조교님들 추가 6 minutes ago

_config.yml Set theme jekyll-theme-cayman 22 days ago

README.md

처음 시작하는 Git + Github 활용하기

오픈소스와 협업에서 많이 쓰이는 Git에 대해 알아보고, 실습을 통해 익히는 과정을 다룹니다. 기초 사용법을 익히고 나서는 Github Pages를 이용하여 개발 관련 블로그를 만들고 운영하는 법을 소개합니다.

강의의 주요 대상은 Git을 처음 접하는 개발자 혹은 Git에 관심이 있는 학생입니다. 프로그래밍을 잘하지 않으셔도 따라오실 수는 있으나 원활한 실습을 위해 `Hello World` 정도는 해보신 분들을 대상으로 진행합니다.

1. Git, Github 소개

<https://github.com/channprj/git-starter>

여기 강의자료를 올린 곳에서 풀 리퀘스트 실습을 해보실 수 있어요.
우선, 저장소를 Fork 받으셔서 각자 개인 저장소로 복사를 해주세요.

https://github.com/Github_아이디/git-starter

**그럼 나의 저장소로 복사(fork) 가 됩니다.
복사된 저장소로 이동을 해주세요.**

```
39
40 ## 8. 알아두면 좋을 팁
41 - awesome-series
42 - .gitconfig
43 - 스스로 문제를 해결하는 법
44 - 도움을 요청하는 법
45
46 ## 9. Thanks to..
47 - [@jhoon2816] (https://github.com/jhoon2816)
48 - [@E-nuri] (https://github.com/E-nuri)
49
```



Commit changes

Update

Add an optional extended description...

- [@아이디] (<https://github.com/아이디>)

그리고 위와 같은 형태로 README.md 의 하단에 9. Thanks to.. 에서
마크다운 형식에 맞춰서 자신의 아이디를 적어주신 후 커밋을 하신 다음

This screenshot shows a GitHub repository page for 'channprj / git-starter'. The top navigation bar includes links for 'This repository', 'Search', 'Pull requests', 'Issues', and 'Gist'. Below the navigation, there are buttons for 'Unwatch' (1), 'Unstar' (1), and 'Fork' (0). The main menu has tabs for 'Code', 'Issues 0', 'Pull requests 0' (which is highlighted with an orange border), 'Projects 0', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. Below the menu, there are 'Filters' (set to 'is:pr is:open'), 'Labels', and 'Milestones' buttons. A green 'New pull request' button is located on the right side of the main content area. A large red circle highlights this button. The central content area features a 'Welcome to Pull Requests!' message with a gear icon.

<https://github.com/channprj/git-starter/pulls>

Pull Requests 항목에서 초록색 New pull request 를 누르시고

This screenshot shows a GitHub repository page for 'channprj / git-starter'. The top navigation bar includes links for 'This repository' and 'Search', and tabs for 'Pull requests', 'Issues', and 'Gist'. On the right side of the header are icons for 'Unwatch' (1), 'Unstar' (1), and 'Fork' (0). Below the header, there are links for 'Code', 'Issues (0)', 'Pull requests (0)', 'Projects (0)', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. The main content area is titled 'Compare changes' and contains instructions: 'Compare changes across branches, commits, tags, and more below. If you need to, you can also compare across forks.' It shows dropdown menus for 'base: master' and 'compare: master'. A prominent green button labeled 'Create pull request' is highlighted with a red circle. A note below it says 'Choose different branches or forks above to discuss and review changes.' At the bottom, there are icons for 'Compare', 'Merge', and 'Tag'. The text 'Compare and review just about anything' is displayed.

Compare pull request 를 고르셔서 요청을 주시면 됩니다.

- GUI 노트
- 외부 서비스 및 각종 연결도구(Integration)

8. 알아두면 좋을 팁

- awesome-series
- .gitconfig
- 스스로 문제를 해결하는 법
- 도움을 요청하는 법

9. Thanks to..

- [@jhoon2816](#)
- [@E-nuri](#)

**풀 리퀘스트 실습의 일환이긴 하지만,
오늘 저와 함께 해주신 분들에게 감사의 표시를 기록하고 싶었습니다. (부끄)**

A screenshot of a GitHub pull request interface. At the top, there's a green header bar with the number '1' and a blue plus sign icon, followed by '+theme: jekyll-theme-cayman' and a red 'diff' icon. Below this, a comment is shown from a user named 'channprj' with a small profile picture of a person working at a desk. The comment was posted 'just now' and reads: 'Jekyll Wiki 의 더 많은 테마와 확장기능들 참고.' (Refer to more themes and extensions on the Jekyll Wiki.) At the bottom of the comment area, there's a 'Reply...' button with a small profile picture next to it.

풀 리퀘스트는 이렇게 코드리뷰 및 라인별로 코멘트를 달 수도 있고,
풀 리퀘스트 승인 권한을 가진 메인테이너가 철저하게 코드 품질 등을 관리할 수 있다는 점에서
오픈소스 계열의 발전을 이룩하는 데 큰 도움이 되었다고 생각해요.

Docker - the open-source application container engine <https://www.docker.com>

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

32,001 commits 7 branches 193 releases 1,666 contributors Apache-2.0

vdeemeester committed on GitHub Merge pull request #32601 from dnephin/builder-shell-words-buffer ... Latest commit 41f4c3c a day ago

.github Update ISSUE_TEMPLATE.md 6 months ago

풀 리퀘스트 기능 등이 포함된, 잘 갖추어진 버전관리 서비스 덕분에

1,600 명이 넘는 컨트리뷰터들이 활동하는 오픈소스가 생겨났고

이러한 오픈소스들 덕분에 서로 공생하고 발전하는 선순환 구조의 문화가 정착된 것 같습니다.

The screenshot shows the GitHub repository for Docker (`docker / docker`). The top navigation bar includes links for 'Pull requests', 'Issues', and 'Gist'. Below the repository name, there are buttons for 'Watch' (3,209), 'Unstar' (41,806), 'Fork' (12,628), and a search bar. A secondary navigation bar below the repository name includes 'Code', 'Issues 2,377' (highlighted in orange), 'Pull requests 126', 'Projects 4', 'Wiki', 'Pulse', and 'Graphs'. A 'Filters' dropdown, a search bar ('is:issue is:open'), and buttons for 'Labels' and 'Milestones' are also present. A green 'New issue' button is located in the top right corner of the main content area. The main content area displays a list of 2,377 open issues, each with a title, labels, and a comment count. The first few issues are:

- docker engine fails to recognize managed plugin driver after uninstalling DEB package** (area/plugins, version/17.03) - 1 comment
- "Address already in use" error with attachable overlay network** (area/networking, version/17.03) - 1 comment
- Limited docker exec concurrency** - 1 comment
- Service Remains at 0/1 Replicas if Bind-mount Requested** (area/swarm, version/17.03) - 1 comment
- network-add during service update doesnt allocate VIP from the newly attached network** (area/networking, area/swarm) - 1 comment
- Zero timestamp in UpdateStatus JSON** (area/api) - 1 comment
- privilege escalation when starting the daemon with user namespaces enabled** (version/17.03) - 1 comment
- docker stack restart option is ignored** (area/bundles, area/docs, version/17.03) - 6 comments
- Container in swarm mode overlay network encounter dns resolve time out with embedded DNS** - 1 comment
- Containers not removed after docker stack rm** (area/bundles, area/swarm, status/more-info-needed) - 1 comment

Github은 풀 리퀘스트 이외에도 훌륭한 이슈관리 도구가 갖추어져 있어요.
 뭔가 문제가 생겼거나, 문제가 생길만한 이슈가 있다면 이렇게 등록을 해두고
 누군가 해당 이슈를 대신 풀어주거나 함께 의논할 수 있어서 좋더라고요.
 저는 좋은 이슈를 발굴해내서 등록하는 것 또한 오픈소스에 기여하는 것이라고 봐요.
 실제로 이슈만 등록하더라도 컨트리뷰션 기록이 남습니다.

This repository Search Pull requests Issues Gist

Watch 3,209 Unstar 41,806 Fork 12,628

Code Issues 2,377 Pull requests 126 Projects 4 Wiki Pulse Graphs

Labels Milestones

Sort ▾

| | 4 Open | 63 Closed | | | |
|---------|-------------|----------------------------|---------------|---------|------------|
| 17.06 | No due date | Last updated 3 minutes ago | 30% complete | 7 open | 3 closed |
| 17.05.0 | No due date | Last updated 5 minutes ago | 95% complete | 14 open | 272 closed |
| 17.03.2 | No due date | Last updated 2 days ago | 100% complete | 0 open | 3 closed |
| 17.04.1 | No due date | Last updated 4 days ago | 0% complete | 0 open | 0 closed |

NOTICE: Issues and PRs included in milestones are not a guarantee t...[\(more\)](#)

이슈들을 모아서 마일스톤으로 등록한 후
일정을 잡아 체계적으로 발전시킬 수 있는 기반도 잡혀있고요.

The screenshot shows the GitHub repository for Docker. At the top, there are navigation links for 'This repository' and 'Search', followed by 'Pull requests', 'Issues', and 'Gist'. Below the header, the repository name 'docker / docker' is displayed, along with a 'Watch' button and a count of 3,209. The main content area features a Kanban-style project board with four columns: 'backlog' (3 items), 'Needs review' (0 items), 'Active' (3 items), and 'Revisit' (4 items). Each column contains a list of issues with their titles, descriptions, and status indicators (e.g., 'status/1-design...', 'status/needs-...'). The 'Active' and 'Revisit' columns have small icons in the top right corner.

| Column | Issue Title | Description | Status |
|--------------|--|--------------------------------|---|
| backlog | Allow user to specify default address pools for docker networks | #29376 opened by aboch | status/1-design...
status/needs-... |
| | Proposal: Add "lock" or "no delete" flag to container | #9878 opened by ibuildthecloud | area/api
area/runtime
kind/feature |
| | Swarm mode with mode=host and port is already in use by service on another host. | #31249 opened by tlvenn | area/networking
area/swarm
kind/bug
version/1.13 |
| Needs review | | | |
| Active | "Save our build times" aka Add support for mounts during docker build | #31499 opened by rdsuhbas | status/1-design... |
| | Allows environment variables to be lazily expanded on container run | #31525 opened by simonferquel | status/1-design... |
| | daemon: allow directory creation in /run/secrets | #31632 opened by cyphar | status/1-design... |
| Revisit | | | |
| Revisit | Windows: Builder GETENV | #29048 opened by jhowardmsft | status/1-design... |
| | Swarm cli enhancement: "service logs" - support for individual tasks | #29223 opened by jmzwcn | status/1-design... |
| | Support INCLUDE in Dockerfiles | #12749 opened by duglin | status/1-design...
status/needs-... |
| | [experimental] Configuration files for services | #32336 opened by aaronlehmann | status/1-design-r... |

그리고 칸반 스타일의 프로젝트 보드가 들어가서
좀 더 고도화된 이슈 및 일정관리가 가능해졌어요.

This screenshot shows the GitHub homepage for the Docker repository. The repository name is "docker / docker". Key statistics displayed include 3,209 pull requests, 41,806 stars, 12,628 forks, and 2,377 issues. Navigation tabs at the top include Code, Issues, Pull requests, Projects, Wiki (which is selected), Pulse, and Graphs.

Home

Sebastiaan van Stijn edited this page 4 days ago · 45 revisions

Engine Project Planning

▶ Pages 23

The Docker follows a **time-based** release process, currently aiming at releasing a new major version every two months. We issue a **code freeze** several weeks before the scheduled release date during which nothing but bugfixes can be added to the release.

Even though the process is time-based, the Docker Engine sets short-term goals for the upcoming release(s). However, time will always win over features: the code freeze will happen regardless of our goals being met.

Roadmap

Docker Engine roadmap items are filed on the issue tracker with the [roadmap](#) label. They are open to community participation: join us on the [usual communication channels](#) to contribute!

| Milestone | Target date | Release captain | Goals |
|-----------|-------------|-----------------|-------|
|-----------|-------------|-----------------|-------|

Resources

- [Engine Roadmap](#)
- [Project Pages](#)
 - [Engine 1.11.0](#)
 - [Engine 1.10.0](#)
 - [Engine 1.9.0](#)
 - [Engine 1.8.0](#)
 - [Engine 1.7.0](#)

Process

- [Contributing guide](#)
- [Maintainers list](#)

Clone this wiki locally

게다가 위키 기능까지 내장되어 있어서
개발을 하면서 필요한 문서작업을 굳이 다른 곳에서 할 필요가 없죠.

 This repository Search Pull requests Issues Gist

 + 

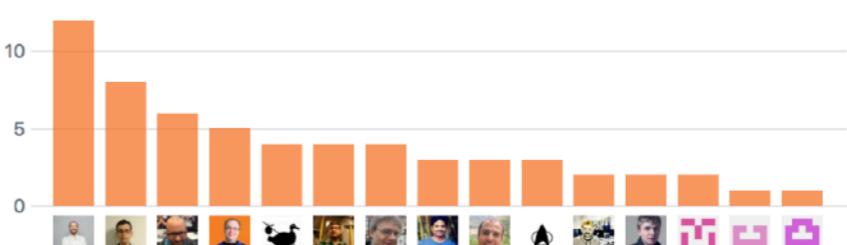
 docker / docker Watch 3,209 Unstar 41,806 Fork 12,628

Code Issues Pull requests Projects Wiki Pulse Graphs

April 7, 2017 – April 14, 2017 Period: 1 week

| Overview | |
|---|---|
|  108 Active Pull Requests |  154 Active Issues |
|  72 Merged Pull Requests |  36 Proposed Pull Requests |
|  79 Closed Issues |  75 New Issues |

Excluding merges, 36 authors have pushed 71 commits to master and 81 commits to all branches. On master, 277 files have changed and there have been 9,664 additions and 4,146 deletions.



1 Release published by 1 person

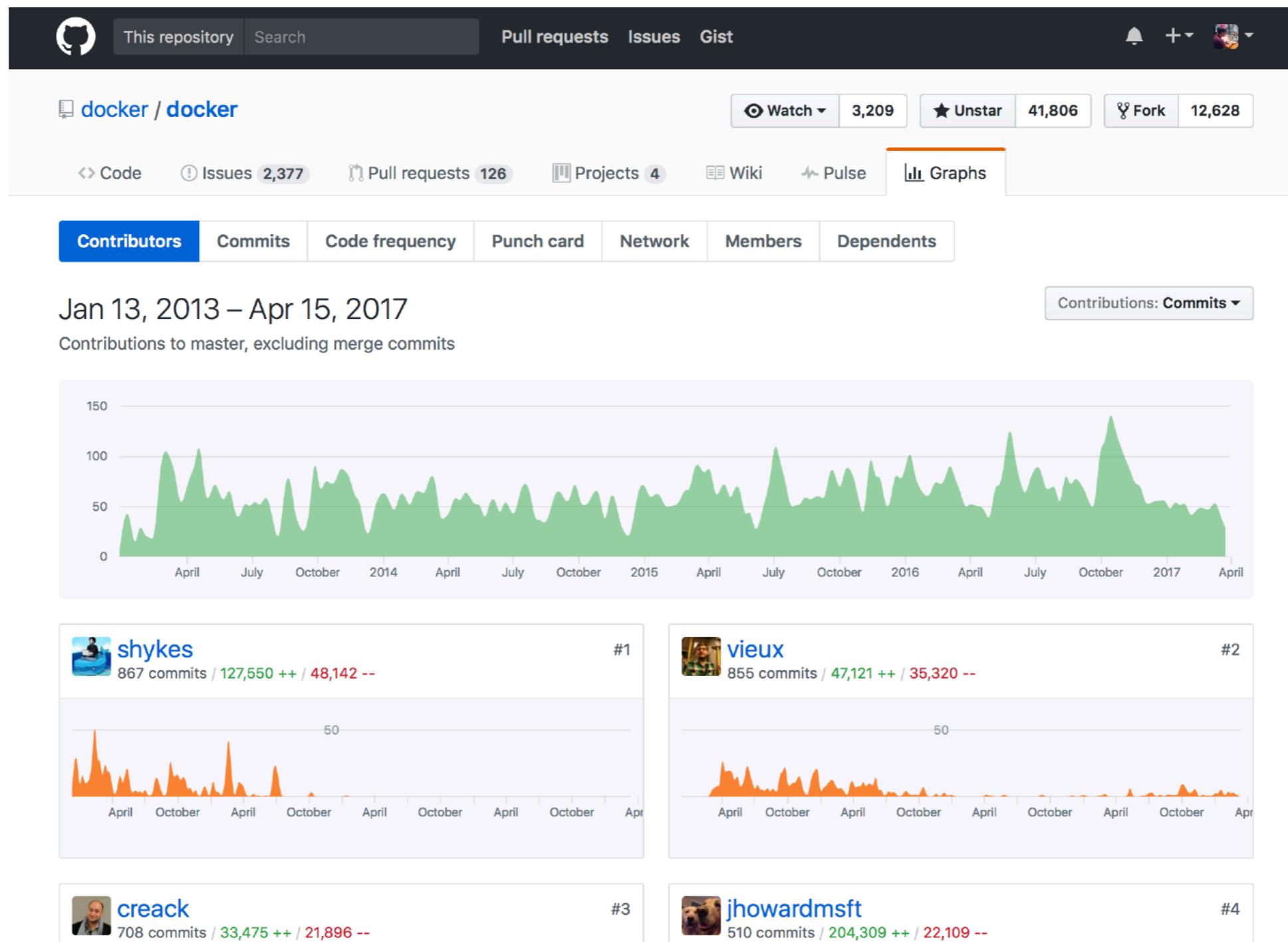
Published v17.05.0-ce-rc1 2 days ago

Merged #32601 Use a bytes.Buffer for shell_words string concat a day ago

Merged #32580 Refactor Dockerfile.parser and directive a day ago

Pulse 탭에서는 누가누가 얼마나 무엇을 열심히 했는가, 어떤 이슈가 해결되었는지에 대한
요약 대시보드까지 볼 수 있습니다.

일부 대학교에서는 컴퓨터 전공 학생들의 팀플을 Github 을 통해서 하도록 하는 곳도 있더라고요.
친구 등에 숨어서 물어가려는 암체 프리라이더를 원천 차단할 수 있는거죠.



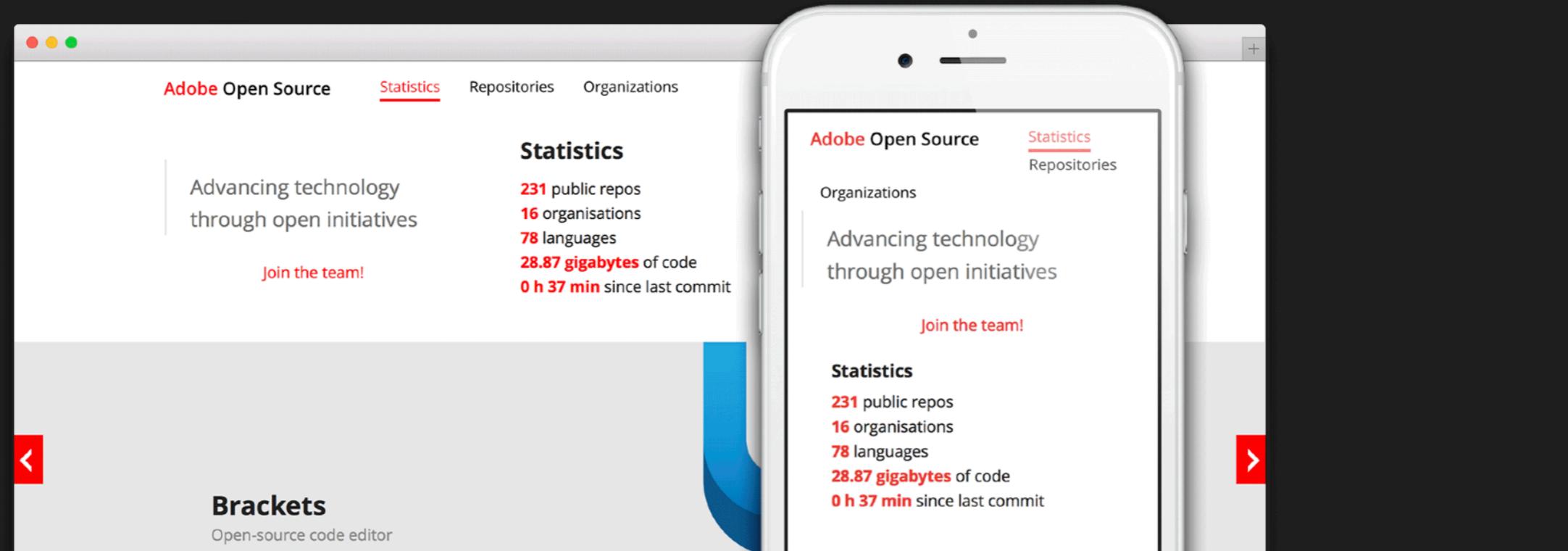
이렇게요.

내가 얼만큼 일했고 놀았는가가 한눈에 그래프로 보일 수 있어서
사실 100% 좋..다고는 못 할수도 있겠네요. 😭

GitHub Pages

Websites for you and your projects.

Hosted directly from your [GitHub repository](#). Just edit, push, and your changes are live.



게다가 개발자에게 가장 효율적이고 유용한 기능 중 하나는
바로 Github Pages 인데요.

Github 저장소를 일종의 무료 호스팅 서비스처럼 사용할 수 있도록 해서
개인 뿐만 아니라 여러 유명한 기업에서도 적극 활용하고 있어요.



처음 시작하는 Git + Github 활용하기

오픈소스와 협업에서 많이 쓰이는 Git에 대해 알아보고, 실습을 통해 익히는 과정을 다룹니다. 기초 사용법을 익히고 나서는 Github Pages를 이용하여 개발 관련 블로그를 만들고 운영하는 법을 소개합니다.

강의의 주요 대상은 Git을 처음 접하는 개발자 혹은 Git에 관심이 있는 학생입니다. 프로그래밍을 잘하지 않으셔도 따라오실 수는 있으나 원활한 실습을 위해 `Hello World` 정도는 해보신 분들을 대상으로 진행합니다.

1. Git, Github 소개

오픈소스 진영에서 주로 사용하는 분산 버전관리 도구인 Git과 Github에 대해 간단히 소개합니다. Git이 다른 버전관리 도구에 비해 어떤 점이 좋은지에 대해서도 다릅니다.

2. 기초 명령어 소개

그리고 예전엔 마크다운 문서를 정적 페이지 생성기(Static Site Generator)로 돌려서 직접 저장소에 올려줘야 했는데, 요즘엔 Github이 자동으로 변환해서 보여주더라고요.

This repository | Search | Pull requests | Issues | Gist | Unwatch | 1 | Unstar | 8 | Fork | 1 | Code | Issues 2 | Pull requests 0 | Projects 0 | Pulse | Graphs | Settings | Edit | Add topics | Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download | channprj committed on GitHub Create LICENSE | Latest commit e7ac2f0 on 4 Sep 2016

| Commit Details | Date |
|---|---------------|
| [CI]: TIL에 Travis-CI 적용하기(수정1) - 한글 태그 추가 | a year ago |
| [ETC]: GDG Webtech 160827 | 7 months ago |
| Update document name by rules. | a year ago |
| [Front-End]: Bower 사용법(수정1) - 태그추가 | a year ago |
| [Linux]: Run Level - Add tag | a year ago |
| [Python]: 단위 테스트(수정1) - 날짜 정정 | 9 months ago |
| [Shell]: ZSH 소유권 문제 해결 | 10 months ago |
| [Github]: Git 한글 깨짐문제(수정1) - 파일이름 수정 | a year ago |
| Update travis configuration. | a year ago |
| Create LICENSE | 7 months ago |
| Update Require.io Badge. | a year ago |
| Update README.md | 7 months ago |
| README.md | |

<https://github.com/channprj/TIL>

이 기능을 활용해서 TIL(Today I Learned) 저장소를 운영하니까
 Git 과 Github 사용법 연습도 되고, 오늘 배운 내용을 기록으로 계속 남기니까
 꾸준히 성장할 수 있는 좋은 계기가 되더라고요. (요즘엔 방치중…😭)

TIL-Driven Githubing

TIL 주도 Github 사용

그래서 여러분도 같이 하셨으면 좋겠습니다.

황금같은 시간을 내서 오신 만큼 손에 쥐고 가시는 게 있으면 좋잖아요.

저도 텀블이 정리를 하고 있어요.

This screenshot shows the GitHub repository `jekyll/jekyll` with the 'Themes' page selected. The top navigation bar includes 'Pull requests', 'Issues', 'Gist', 'Watch 1,317', 'Unstar 29,407', 'Fork 6,534', and tabs for 'Code', 'Issues 66', 'Pull requests 45', 'Projects 0', 'Wiki' (which is active), 'Pulse', and 'Graphs'. Below the tabs, it says 'Yehonal edited this page 12 hours ago · 182 revisions'. The main content area lists themes like 'Git-Wiki', 'Jalpc', 'Pixyll', 'Jekyll Metro', 'Midnight', 'Leap Day', 'Freelancer', 'HMFAYSAL Notepad Theme', 'Bitwiser', 'White Paper', 'Jalpc-A', 'Typewriter', 'block-log', 'Otter Pop', 'Noita', 'Simpleyyt', and 'architect', each with a link to 'source' and 'demo'. To the right is a sidebar titled 'Pages 20' with links to 'Home', 'Blog Migrations', 'Configuration', 'Contribute', 'Deployment', 'FAQ', 'How Jekyll works', 'Inspired By', 'Install', 'Liquid extensions', 'Markup Problems', 'Pagination', 'Permalinks', 'Sites', and 'Template Data'. A search bar at the top of the sidebar says 'Find a Page...'. At the bottom of the page is the URL <https://github.com/jekyll/jekyll/wiki/Themes>.

시간 관계 상 빠르게 만드는 방법을 소개해드리겠습니다.
위의 링크는 사전에 다른 분들이 만든 Github Pages 테마 저장소인데요.

GitHub repository page for **rosario / kasper**. Key statistics:

- 48 commits
- 1 branch
- 0 releases
- 10 contributors
- MIT license

Latest commit: 0d699c8 on 27 Jun 2016

| Commit Details | Date |
|---|---------------|
| _includes
Update to Casper v1.1.1 - Update jQuery - Adding partial templates - ... | 3 years ago |
| _layouts
Fix rss link in layouts to use the new feed.xml | a year ago |
| _posts
Small fix, using categories instead of tags | 3 years ago |
| _sass
Introduced SASS for syntax highlighting. | 3 years ago |
| assets
Update to Casper v1.1.1 - Update jQuery - Adding partial templates - ... | 3 years ago |
| .gitignore
Ignore .sass-cache. | 3 years ago |
| LICENSE.txt
first commit | 4 years ago |
| README.md
Update README - Add missing info about installing pygments.rb and add... | 10 months ago |
| _config.yml
Change redcarpet to kramdown (see https://help.github.com/articles/up... | a year ago |

일단 이번 실습에서는 kasper 를 사용해보도록 하겠습니다.

해당 저장소에 들어가셔서 Fork 를 해주세요.

HTML, CSS 를 조금 아신다면 테마를 직접 만드셔서 나만의 테마를 만드셔도 됩니다.

channprj / blog
forked from pythonkr/blog

Unwatch 1 | Star 0 | Fork 1

Code Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Options

- Collaborators
- Branches
- Webhooks
- Integrations & services
- Deploy keys

Settings

Repository name

blog Rename

Features

- Wikis
- Restrict editing to collaborators only
- Issues

Fork 를 받아오시면 자신의 저장소로 새로 복사가 되었을 거에요.
 자신의 저장소로 이동을 하셔서 Settings 에서 저장소의 이름을 **아이디.github.io** 로 바꿔주세요.
 Github 에서는 **아이디.github.io** 라는 이름으로 저장소를 만들게 되면
 자동으로 Github Pages 전용 저장소로 인식을 하게 됩니다.

GDG Webtech 160827

단순 참고용 메모, 개인 기록임.

- CPU 와 GPU 사이에 통신하면서 발생하는 이슈
- 메모리의 한계에 도달했을 때
- CPU 데이터 변경 시 GPU 데이터도 변경 필요

Vertex & Polygon

개념 숙지

렌더링

GPU를 효율적으로 활용하도록 잘 이용해야.

- 텍스쳐로 이미지 고속 출력
- 이미 가지고 있는 텍스쳐 재활용
- 회전, 확대, 축소, 기울임, 반투명 등
- 동시 ...

[more ...](#)

그리고 자신의 아이디.github.io. 로 접속을 하시면

Github Page 에 블로그가 뜨게 됩니다.

위의 스크린샷은 제가 따로 만든 Github Pages 기반의 TIL 블로그 입니다.

[rosario / kasper](#)

Code Issues 6 Pull requests 4 Projects 0 Wiki Pulse Graphs

Watch 17 Unstar 766 Fork 342

Ghost's default theme (Casper) on Jekyll

48 commits 1 branch 0 releases 10 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

rosario committed on GitHub Merge pull request #33 from pavelbinar/master ... Latest commit 0d699c8 on 27 Jun 2016

| | | |
|-----------------------------|---|---------------|
| _includes | Update to Casper v1.1.1 - Update jQuery - Adding partial templates - ... | 3 years ago |
| _layouts | Fix rss link in layouts to use the new feed.xml | a year ago |
| _posts | Small fix, using categories instead of tags | 3 years ago |
| _sass | Introduced SASS for syntax highlighting. | 3 years ago |
| assets | Update to Casper v1.1.1 - Update jQuery - Adding partial templates - ... | 3 years ago |
| .gitignore | Ignore .sass-cache. | 3 years ago |
| LICENSE.txt | first commit | 4 years ago |
| README.md | Update README - Add missing info about installing pygments.rb and add... | 10 months ago |
| _config.yml | Change redcarpet to kramdown (see https://help.github.com/articles/up...) | a year ago |
| about.md | Update to Casper v1.1.1 - Update jQuery - Adding partial templates - ... | 3 years ago |
| feed.xml | Add default feed.xml file | a year ago |
| index.html | Fix rss link in index.html | a year ago |

자, 이제 TIL 블로그가 만들어졌는데요. 오늘 배운 내용을 적어볼까요?
 _posts 폴더를 누르시면

Branch: master ▾ [kasper](#) / [_posts](#) /

 **rosario** Small fix, using categories instead of tags

..

 [2013-11-10>Welcome-to-jekyll.markdown](#) Small fix, using categories instead of tags

마크다운 문서가 하나 보이실거에요.

처음에 블로그가 생겼을 때 본 샘플로 작성된 글인데요.

이 글을 토대로 오늘 배운 내용을 참고해서 마크다운으로 작성하여 올리시면 자동으로 글이 등록됩니다.

Today I Learned

검색



검색결과 약 39개(0.54초)

[**zsh - Today I Learned**](#)
zsh 소유권 문제 해결. 가끔 터미널로 작업하다 보면 아래와 같은 상황을 직면한다. Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-66-generic x86_64) ...
<https://til.chann.kr/tag/zsh>
[Structured data](#)

[**zsh 소유권 문제 해결 - Today I Learned**](#)
2016년 6월 4일 ... 가끔 터미널로 작업하다 보면 아래와 같은 상황을 직면한다. Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-66-generic x86_64) ...
<https://til.chann.kr/shell/zsh-ownership-issue-fix>
[Structured data](#)

[**ZSH - CHANN**](#)
2015년 10월 15일 ... 터미널 실행시 zshell 로딩 빠르게 하기. 나는 구형 맥북을 사용한다. 정확한 모델명은 Macbook(13-inch, Late 2009)이지만, 흔동이라는 별칭으로 ...
<https://blog.chann.kr/tag/zsh/>
[Structured data](#)

 [**터미널 실행시 zshell 로딩 빠르게 하기**](#)
터미널 실행시 zshell 로딩 빠르게 하기. 2015년 10월 15일 on Dev, 개발, Terminal, 터미널, Shell, 쉘, ZSH, ZShell, CLI, 개발환경. 나는 구형 맥북을 사용한다. 정확한 ...

이렇게 Github Pages로 배운 내용들을 문서화해서 정리를 하다 보면,
분명 예전에 봤던 내용인데 까먹어서 다시 검색하게 되는 그런 내용들을
자신의 TIL 블로그 안에서 찾을 수 있기 때문에 탐색하는 시간이 줄어듭니다.
이렇게 나만의 지식 참고로 사용하실 수도 있어요.

오픈소스 프로젝트에 기여하는 법

오픈소스 프로젝트에 기여를 하는 법은 다양해요.
관점에 따라 다르지만, 꼭 기능 추가라던가 버그 패치를 하는 것만이 기여는 아니라고 생각합니다.
(라이브 화면)

Git 관련 도구 소개

<https://github.com/dictcp/awesome-git>

Git 과 관련된 다양한 도구들을 소개해드릴게요.

우선, awesome-git 이라는 저장소에서 큐레이션 하는 리스트들을 참고하시면 좋습니다.

지금은 아직 여기 소개되고 있지는 않은 몇가지 도구들을 소개해드릴게요.

(라이브 화면)

알아두면 좋을 팁

- awesome-series : <https://github.com/sindresorhus/awesome>
- .gitconfig : <https://github.com/channprj/dotfiles-macos>
- 스스로 문제를 해결하는 법
- 도움을 요청하는 법

감사합니다.

- e-mail : chann@chann.kr
- github : <https://git.chann.kr>
- twitter : @channprj
- facebook : <https://fb.com/channprj>
- blog : <https://blog.chann.kr>

혹시 따로 질문이 있으시거나,
따로 도움을 받고 싶으시다면 언제든지 연락을 주세요.
고맙습니다.