Git 활용(1)

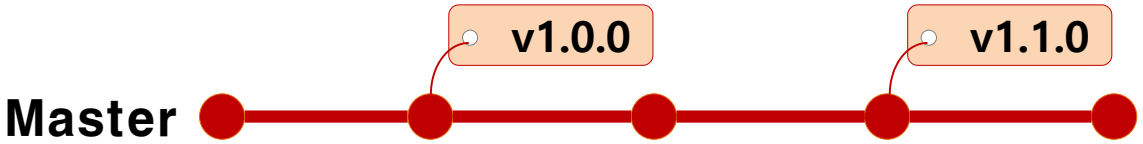
- Git 태그 활용
- Git 브랜치 활용
- Git 원격 저장소(GitHub) 활용

Computer & Software Engineering
Lee Jong Yeol
soyeum@kumoh.ac.kr

Git 태그 활용

❑ 태그(tag)(1)

- 커밋을 참조하기 쉽도록 알기 쉬운 이름을 붙이는 것
- 주로 새 버전이 릴리즈 될 때마다 해당 커밋에 태그를 남김




```
linuxer@linuxer-PC:~/libgit2$ git tag
v0.1.0
...
v1.1.0
linuxer@linuxer-PC:~/libgit2$ git tag -l "v0.28*"
v0.28.0
v0.28.0-rc1
...
linuxer@linuxer-PC:~/libgit2$ ls .git/refs/tags                (태그 확인 방법-1)
linuxer@linuxer-PC:~/libgit2$ find .git/refs -type f           (태그 확인 방법-2)
.git/refs/heads/master
.git/refs/remotes/origin/HEAD
...
```

Git 태그 활용

❑ 태그(tag)(2)

- 일반 태그(lightweight tag): 이름만 붙이는 태그
- 주석 태그(annotated tag): 태그에 대한 설명, 서명, 생성자 정보 포함

```
<lightweight tag 붙이기>
linuxer@linuxer-PC:~/libgit2$ git tag v9.9-lw

<annotated tag 붙이기>
linuxer@linuxer-PC:~/libgit2$ git tag -a v9.10 -m "my test version 9.10"
linuxer@linuxer-PC:~/libgit2$ git tag
...
v9.9-lw
v9.10

<태그 정보 확인>
linuxer@linuxer-PC:~/libgit2$ git show v9.10
tag v9.10
Tagger: soyeum <soyeum@gmail.com>
Date:   Sun Oct 25 01:19:48 2020 +0900

my test version 9.10
...
```

Git 태그 활용

❑ 태그(tag)(3)

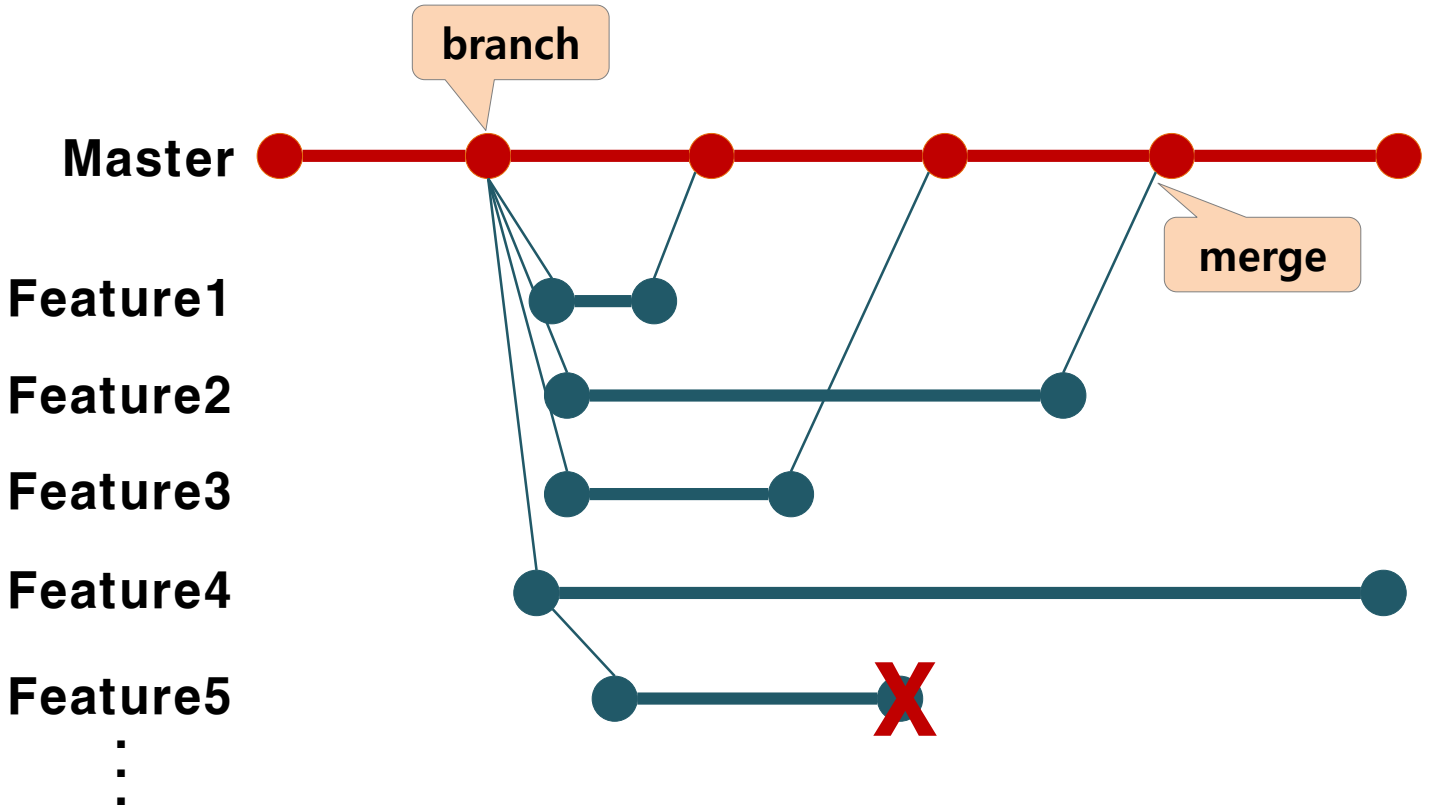
- 이미 커밋된 것에 대해 나중에 태그를 부착/제거 가능

```
linuxer@linuxer-PC:~/libgit2$ git log --pretty=oneline -4
aaf3e18fbe2f6e337cb03d44c0280f2cd30de350 (HEAD -> master, tag: v9.9-lw, tag:
v9.9, tag: v9.10-lw, tag: v10.1-lw) commit test...
2a51679005285e4e5f8642dd86dc35b46de772c1 (origin/master, origin/HEAD) Merge
pull request #5659 from libgit2/ethomson/name_is_valid
8b0c7d7cdf4fd5ad30efa66251733d85b9c8b641 changelog: include new reference
validity functions
0caa4655ebdb7bf028df970d0651378d121fab3e Add git_tag_name_is_valid

linuxer@linuxer-PC:~/libgit2$ git tag -a v11.1 aaf3e1          (태그 부착)
linuxer@linuxer-PC:~/libgit2$ git log --pretty=oneline -1
aaf3e18fbe2f6e337cb03d44c0280f2cd30de350 (HEAD -> master, tag: v9.9-lw, tag:
v9.9, tag: v9.10-lw, tag: v11.1, tag: v10.1-lw) commit test...

inuxer@linuxer-PC:~/libgit2$ git tag -d v11.1                  (태그 삭제)
'v11.1' 태그 삭제함 (과거 76c795bb9)
linuxer@linuxer-PC:~/libgit2$ git log --pretty=oneline -1
aaf3e18fbe2f6e337cb03d44c0280f2cd30de350 (HEAD -> master, tag: v9.9-lw, tag:
v9.9, tag: v9.10-lw, tag: v10.1-lw) commit test...
```

Git 브랜치 활용

❑ 브랜치(branch)(1)

- 원래 코드와는 상관없이 독립적으로 개발을 진행하는 것
- 대부분의 VCS에서 제공하고 있는 기능임



Git 브랜치 활용

❑ 브랜치(branch)(2)

▪ branch의 목적

- 병행 개발을 통해 지속적이고 안정적인 소프트웨어 개발 가능
  (기본적으로 master 브랜치와 topic 브랜치를 분리 진행)
- 목적에 맞는 제품 개발이 가능
  (제품 출시 일정, 기능, 고객에 맞는 제품 개발을 병행)

▪ branch의 전략

- 신규 기능, 버그 수정, 실험 적인 작업이 요구될 때 사용
- 같은 기능 개발 중에도 다양한 실험의 목적으로 브랜치에서
  또 다른 새로운 브랜치 생성
- 브랜치에서 작업 중에 수정된 내용을 취소하고 싶다면
  해당 브랜치만 삭제

Git 브랜치 활용

❑ 브랜치(branch)(3)

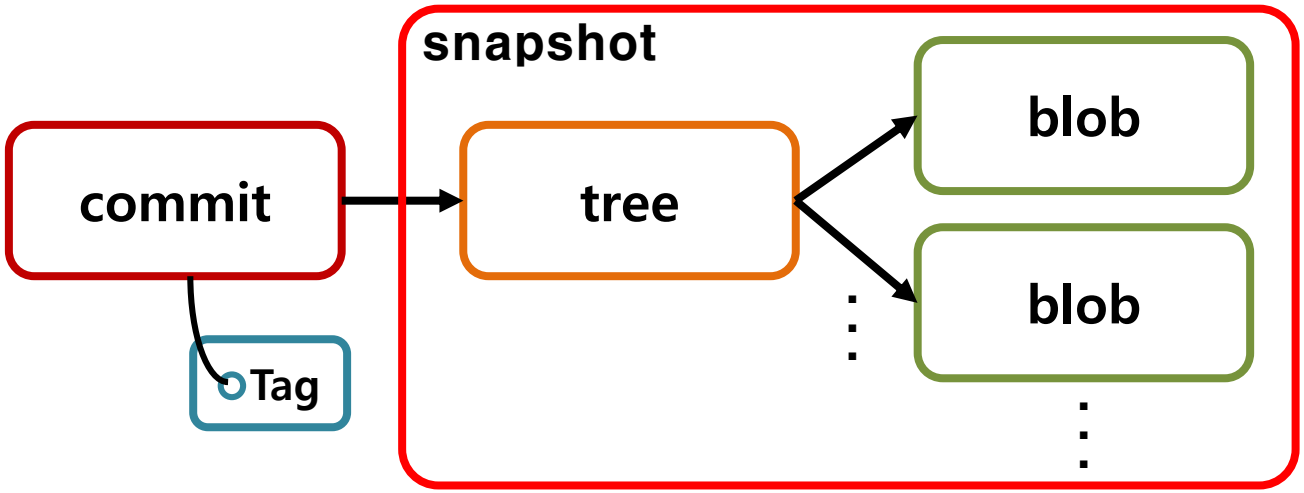
▪ Git branch의 장점

- 규모가 큰 프로젝트에서도 빠르고 안정적으로 동작
- CVCS와 비교해서 브랜치 이동(swith)이 매우 간편함
- 브랜치 생성 비용이 상대적으로 낮음 (vs. Mercurial)
- 브랜치의 이력도 merge시 모두 포함되어 이력관리에 유리함

Git 브랜치 활용

❑ 브랜치(branch)(4)

- 커밋되면 Git 객체들이 생성되고 저장: .git/objects 에서 확인 가능
  - Blob: 파일이 저장되는 객체(.git/objects, .git/index)
  - Tree: Blob 또는 Tree를 포인팅하는 디렉터리 객체 (.git/objects)
  - Commit: 커밋하면 생성되는 객체(.git/objects)
  - Tag: 특정 commit 객체에 대한 alias (.git/refs/tags)



Git 브랜치 활용

❑ 브랜치(branch)(5)

▪ commit object 확인(1)

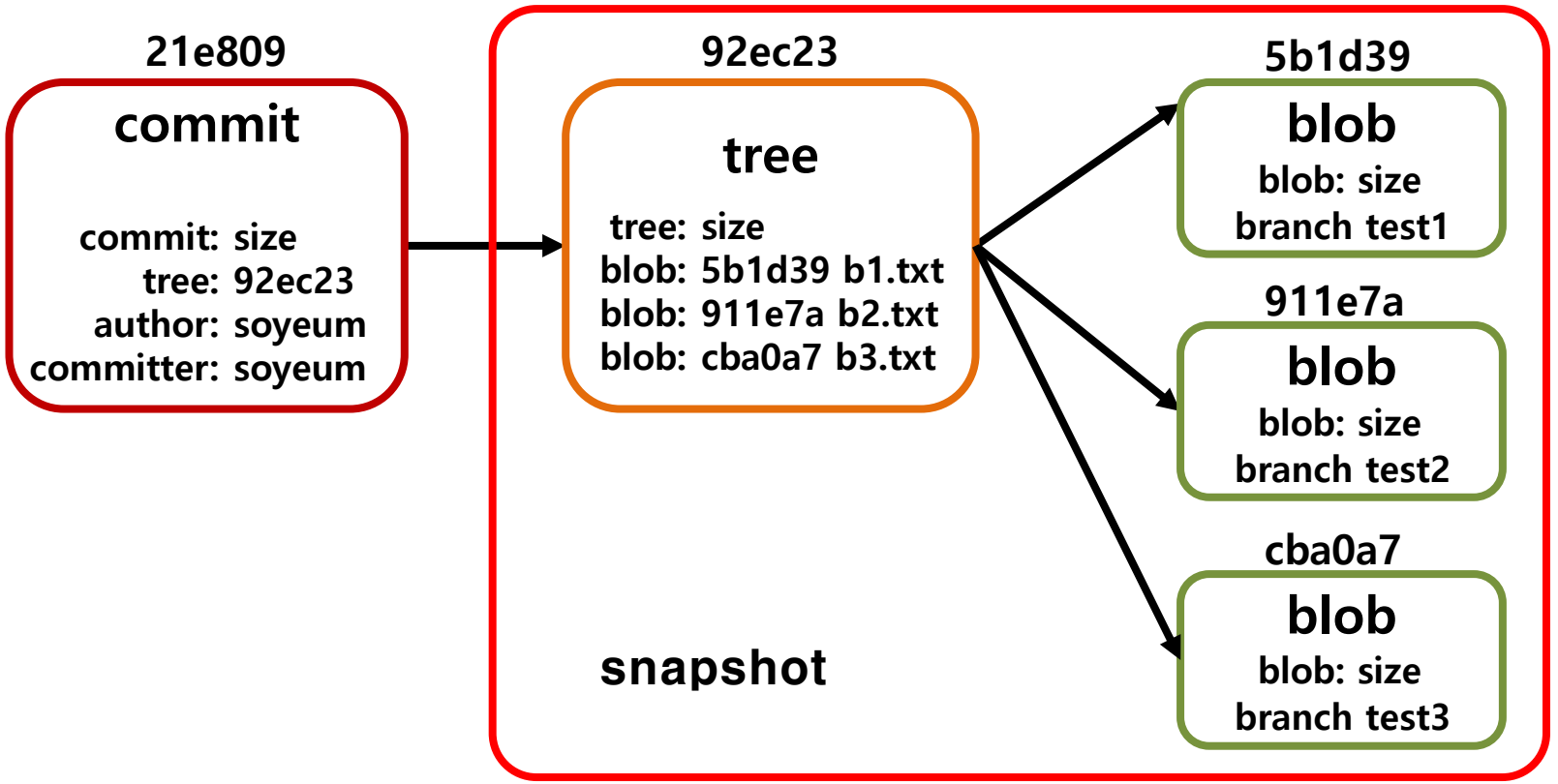
```
linuxer@linuxer-PC:~/libgit2$ echo 'branch test1' > b1.txt
linuxer@linuxer-PC:~/libgit2$ echo 'branch test2' > b2.txt

linuxer@linuxer-PC:~/libgit2$ git add b1.txt b2.txt
linuxer@linuxer-PC:~/libgit2$ git commit -m 'branch test'
[master 21e8095b8] branch test
 3 files changed, 3 insertions(+)
 create mode 100644 b1.txt
 create mode 100644 b2.txt

linuxer@linuxer-PC:~/libgit2$ git log --pretty=oneline -1
21e8095b85740776805c42940aab769f46dd8037 (HEAD -> master) branch
test

linuxer@linuxer-PC:~/libgit2/$ git log --pretty=format:"%h %s" -1 --graph
* 21e8095b8 branch test
```

Git 브랜치 활용

❑ 브랜치(branch)(5)

▪ commit object 확인(2)

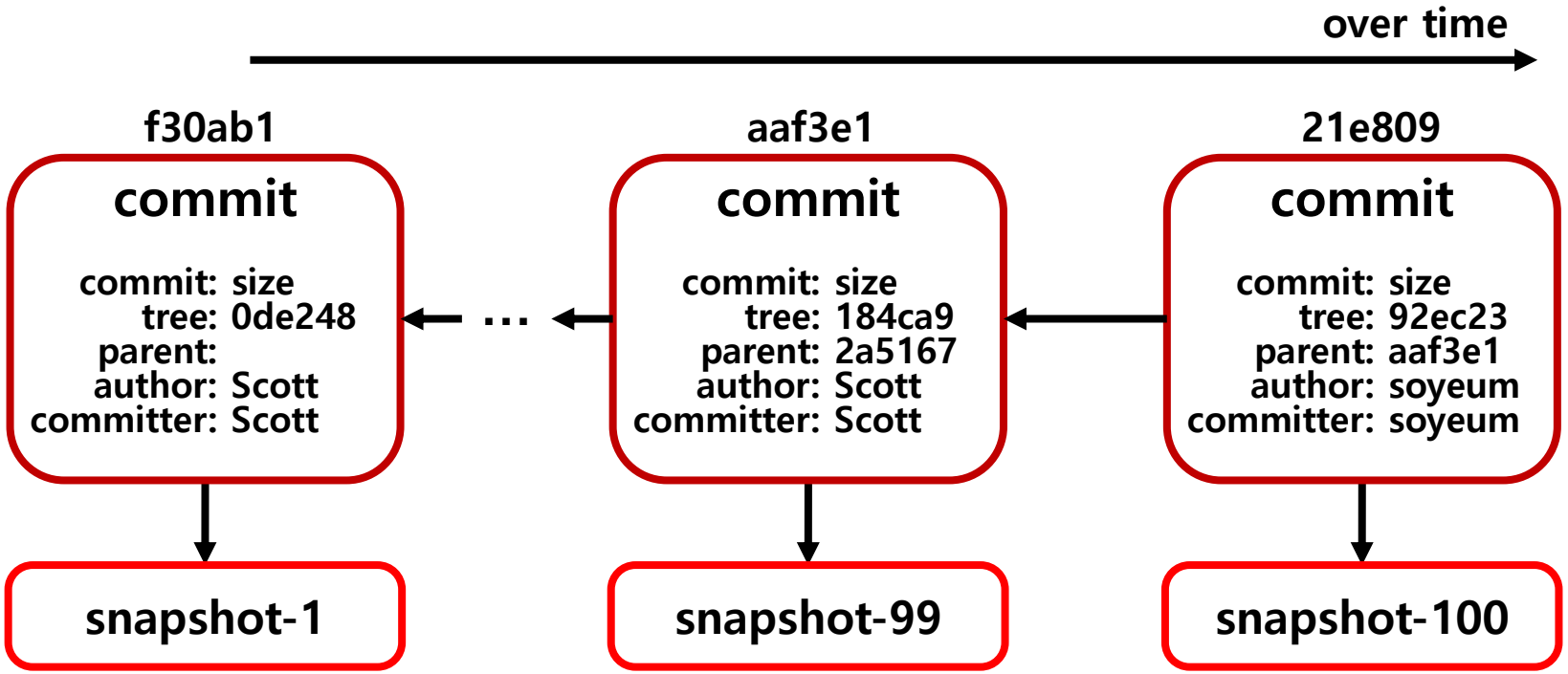
```
linuxer@linuxer-PC:~/libgit2$ ls .git/objects/21
b9c215947a53247ddd247267350bf92f29d201
e8095b85740776805c42940aab769f46dd8037
```



Git 브랜치 활용

❑ 브랜치(branch)(6)

▪ 이전 커밋과의 관계

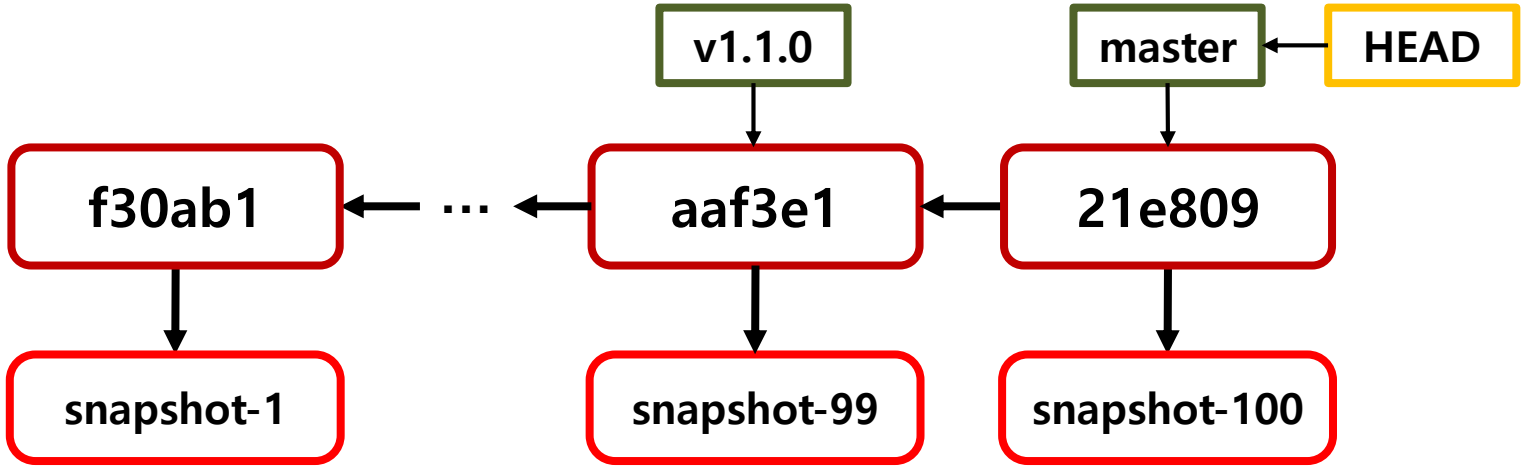


Git is a content-addressable filesystem

Git 브랜치 활용

❑ 브랜치(branch)(7)

- Git의 브랜치는 커밋 사이를 가볍게 이동할 수 있는 포인터
- git init 명령이 master 브랜치 생성
- 처음 커밋 명령이 실행되면, 생성된 커밋 객체를 master 브랜치가 포인팅
- 이후 커밋을 만들때마다 가장 마지막 커밋에 master 브랜치가 포인팅

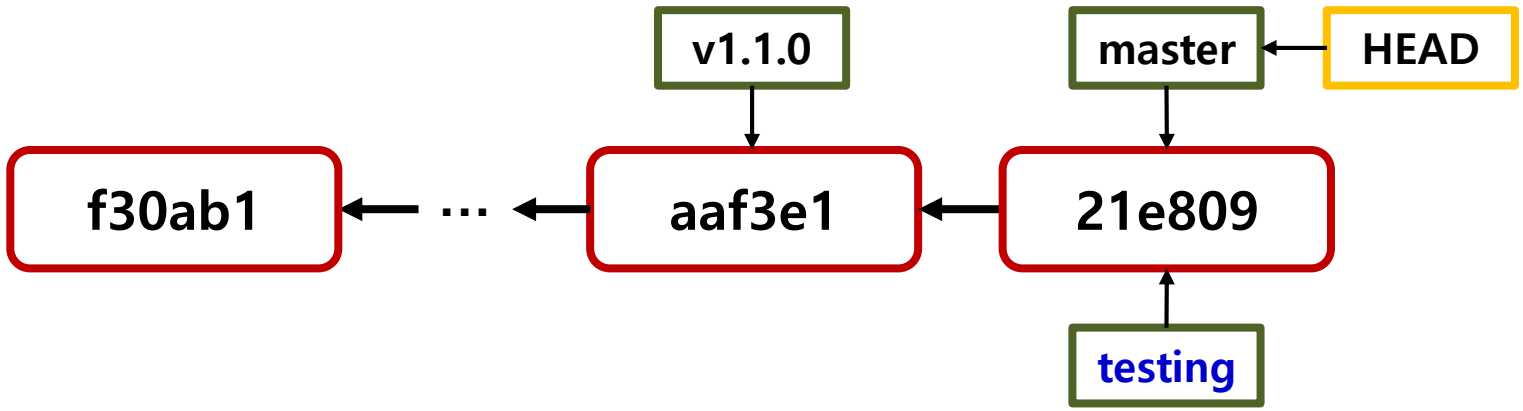


Git 브랜치 활용

❑ 브랜치(branch)(8)

▪ 새로운 브랜치 생성: git branch <브랜치 이름>

```
linuxer@linuxer-PC:~/libgit2$ git branch testing
linuxer@linuxer-PC:~/libgit2$ git branch
* master
  testing
linuxer@linuxer-PC:~/libgit2$ git log --decorate --oneline -2
21e8095b8 (HEAD -> master, testing) branch test
aaf3e18fb (tag: v9.9-lw, tag: v9.9, tag: v9.10-lw, tag: v10.1-lw) commit test...
```

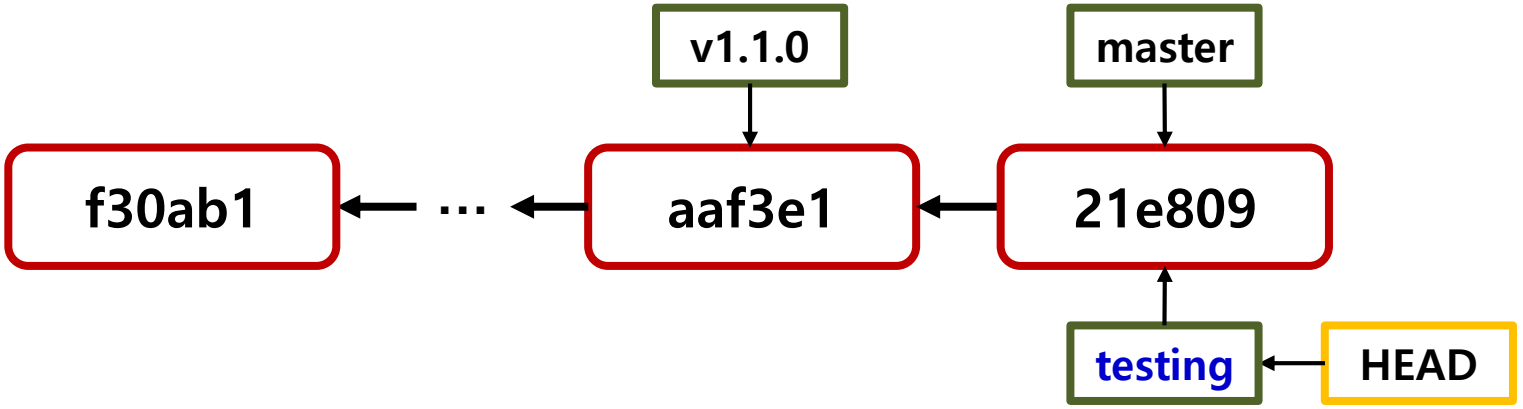


Git 브랜치 활용

❑ 브랜치(branch)(9)

- 브랜치 이동: git checkout <브랜치 이름>
- 브랜치를 이동하면 working 디렉터리의 파일도 변경됨
- 브랜치 생성과 이동을 한 번에 진행: git checkout -b testing

```
linuxer@linuxer-PC:~/libgit2$ git checkout testing
M       AUTHORS
'testing' 브랜치로 전환합니다
linuxer@linuxer-PC:~/libgit2$ git log --decorate --oneline -1
21e8095b8 (HEAD -> testing, master) branch test2
```

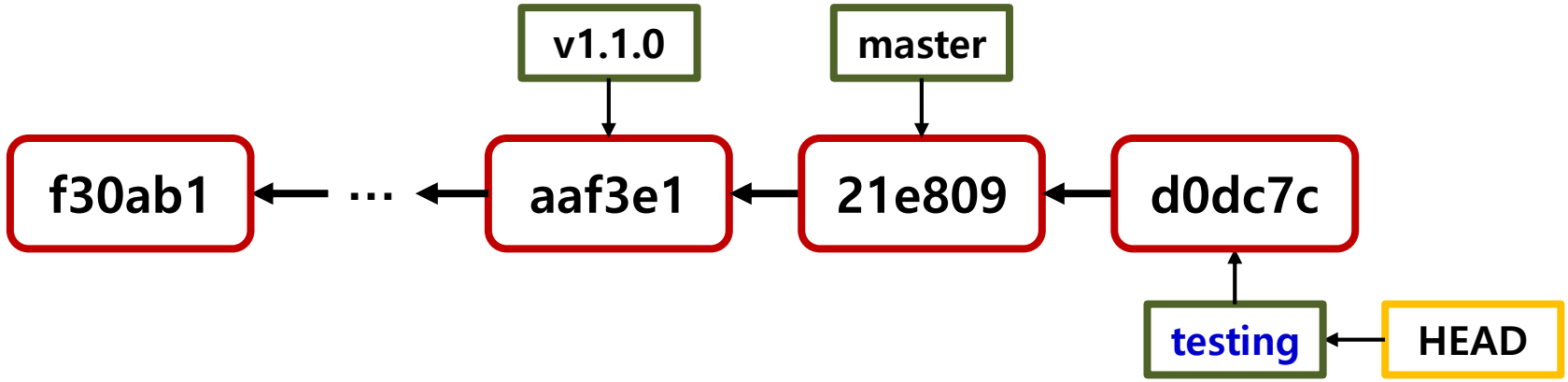


Git 브랜치 활용

❑ 브랜치(branch)(10)

▪ 새로운 커밋으로 브랜치 이동 확인

```
linuxer@linuxer-PC:~/libgit2$ echo 'branch test3' > b3.txt
linuxer@linuxer-PC:~/libgit2$ git commit -a -m 'testing branch test'
[testing d0dc7ca44] testing branch test
 1 file changed, 1 insertion(+)

linuxer@linuxer-PC:~/libgit2$ git log --decorate --oneline -2
d0dc7ca44 (HEAD -> testing) testing branch test
21e8095b8 branch test
```

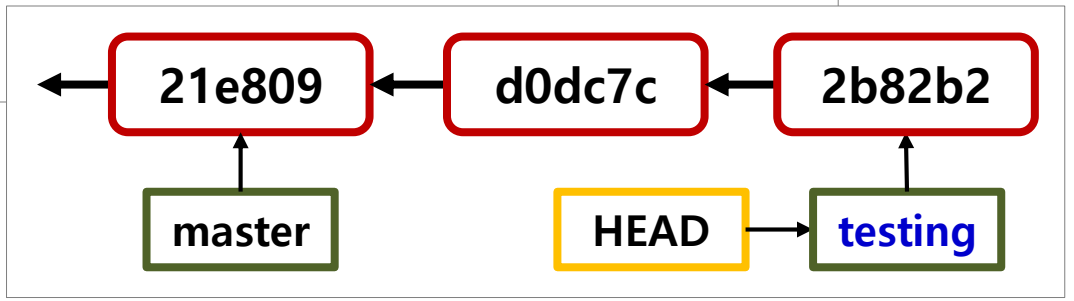


Git 브랜치 활용

❑ 브랜치(branch)(11)

▪ 새로운 브랜치에서 파일 수정후 커밋

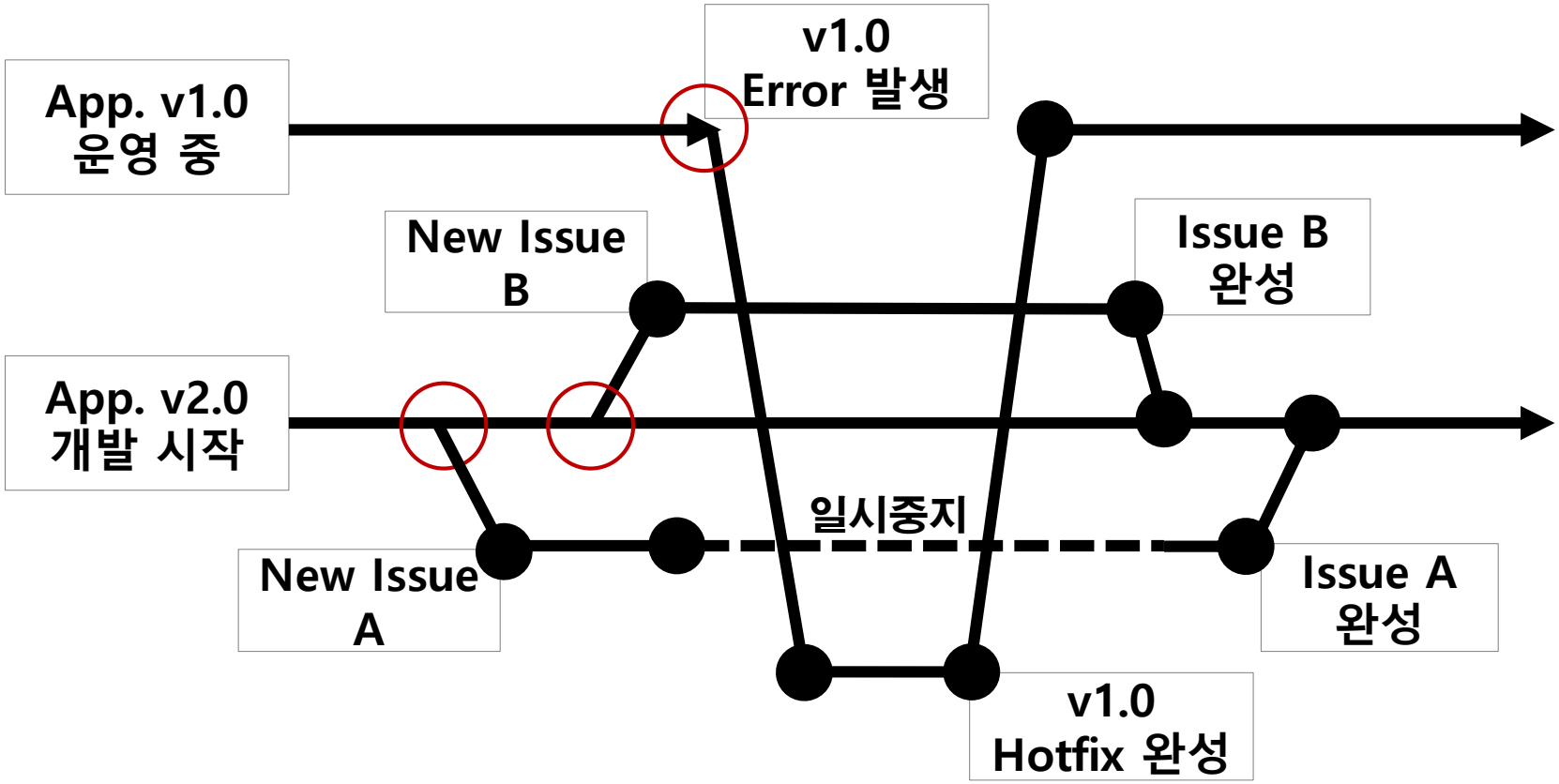
```
linuxer@linuxer-PC:~/libgit2$ echo 'testing branch test1' >> b3.txt
linuxer@linuxer-PC:~/libgit2$ cat b3.txt
branch test3
testing branch test1

linuxer@linuxer-PC:~/libgit2$ git commit -a -m 'b3.txt updated'
[testing 2b82b26b8] b3.txt updated
 1 file changed, 1 insertion(+)

linuxer@linuxer-PC:~/libgit2$ git log --oneline --decorate -3
* 2b82b26b8 (HEAD -> testing) b3.txt updated
* d0dc7ca44 testing branch test
* 21e8095b8 (master) branch test
```



Git 브랜치 활용

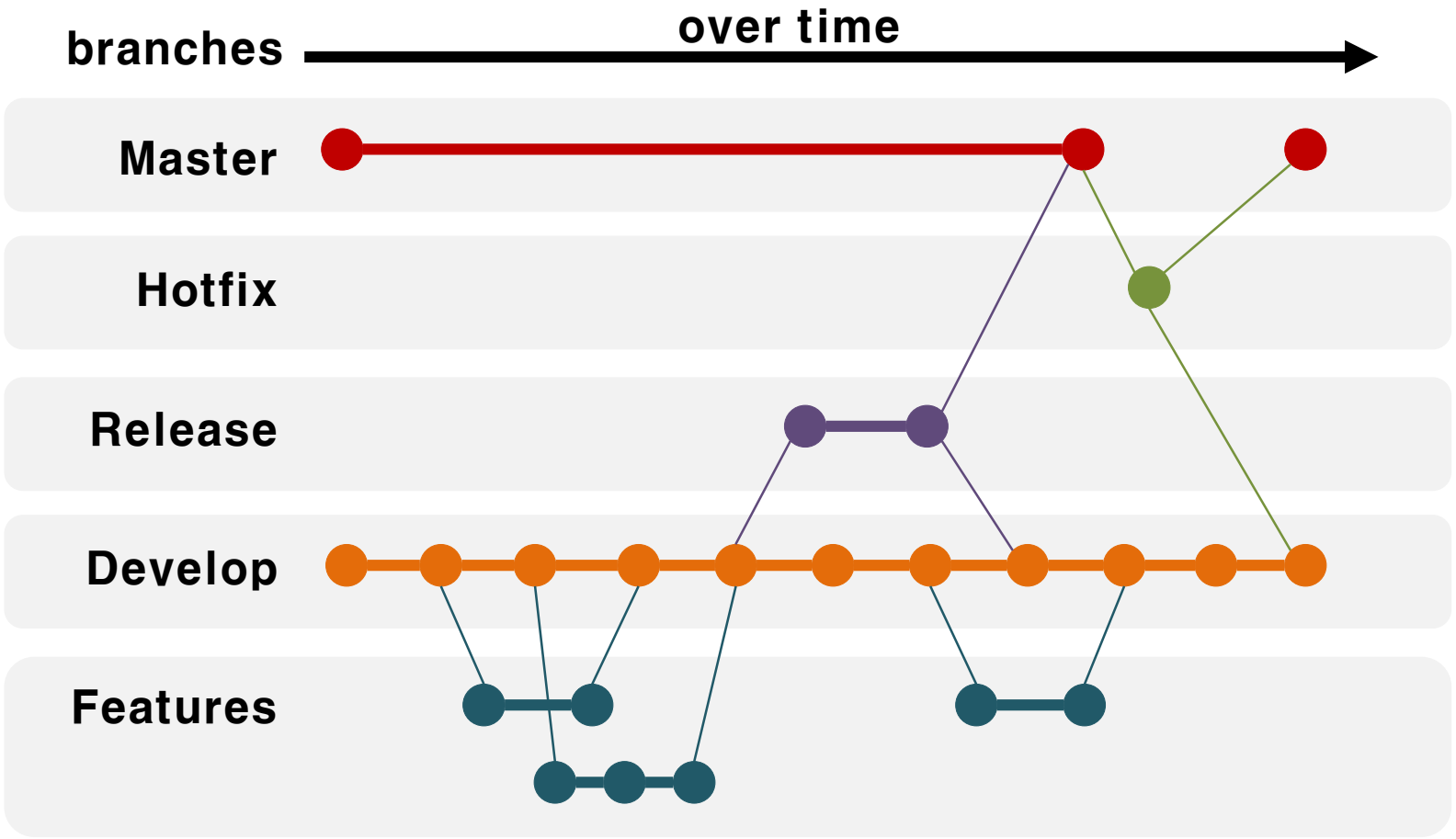
❑ 브랜치(branch)(12)

▪ 실제 개발과정에서 브랜치 활용 방식



Git 브랜치 활용

❑ [참고] 실전 개발 프로세스에서 브랜치 관리

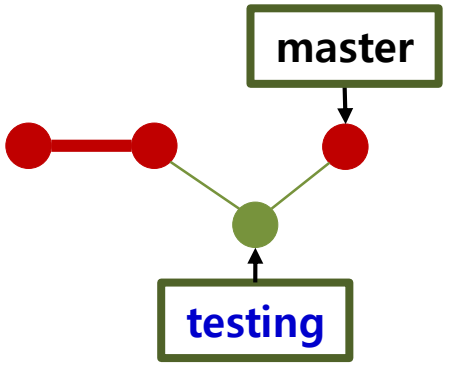


Git 브랜치 활용

❑ 머지(merge)(1)

- 둘 이상의 개발 이력을 결합: git merge <결합할 브랜치 이름>
  (주의) 반드시 결합될 브랜치로 checkout 후 진행할 것

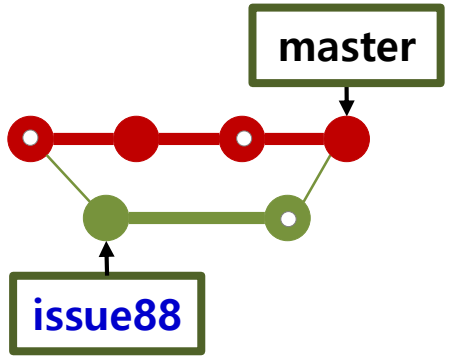
```
linuxer@linuxer-PC:~/libgit2$ git branch
  master
* testing
linuxer@linuxer-PC:~/libgit2$ git checkout master
linuxer@linuxer-PC:~/libgit2$ git branch
* master
  testing
linuxer@linuxer-PC:~/libgit2$ git merge testing
업데이트 중 21e8095b8.. 2b82b26b8
Fast-forward
 b3.txt | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 b3.txt

linuxer@linuxer-PC:~/libgit2$ git branch -d testing  (브랜치 삭제)
```



testing 브랜치가 master 브랜치 이후의 커밋을 가리키고 있기 때문에 그저 testing 브랜치는 master 브랜치와 동일한 커밋으로 간주됨

Git 브랜치 활용

❑ 머지(merge)(2)



▪ 커밋 시점이 다른 브랜치의 결합(1)

```
linuxer@linuxer-PC:~/libgit2$ git branch issue88
linuxer@linuxer-PC:~/libgit2$ git checkout issue88
linuxer@linuxer-PC:~/libgit2$ echo 'issue88 branch test' > issue88.txt
linuxer@linuxer-PC:~/libgit2$ git commit -a -m 'issue88.txt added'

linuxer@linuxer-PC:~/libgit2$ git checkout master
linuxer@linuxer-PC:~/libgit2$ echo 'master merge test' >> b3.txt
linuxer@linuxer-PC:~/libgit2$ git commit -a -m 'b3.txt updated'

linuxer@linuxer-PC:~/libgit2$ echo 'master merge test' > b4.txt
linuxer@linuxer-PC:~/libgit2$ git add b4.txt
linuxer@linuxer-PC:~/libgit2$ git commit -m 'b4.txt added'
```

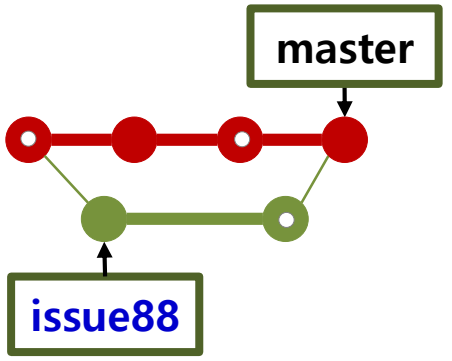
Git 브랜치 활용

❑ 머지(merge)(3)

▪ 커밋 시점이 다른 브랜치의 결합(2)

```
linuxer@linuxer-PC:~/libgit2$ git merge issue88
Merge made by the 'recursive' strategy.
 issue88.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 issue88.txt

linuxer@linuxer-PC:~/libgit2$ git log --oneline --decorate --graph -5
*   672a5eada (HEAD -> master) Merge branch 'issue88'
|\
| * 5166ca974 (issue88) issue88.txt added
* | b4c7fd8bf b4.txt added
* | 2874ef255 b3.txt updated
|/
* b66a30a18 b3.txt updated

linuxer@linuxer-PC:~/libgit2$ git branch -d issue88  (브랜치 삭제)
```

3-way merge
: 커밋 두 개와 공통 조상 하나를 이용해서 merge 수행



Git 브랜치 활용

❑ 머지(merge)(4)

- merge 할 수 없는 경우: 동일한 부분을 변경했다면 충돌 발생

```
linuxer@linuxer-PC:~/libgit2$ git branch conflict
linuxer@linuxer-PC:~/libgit2$ git checkout conflict
linuxer@linuxer-PC:~/libgit2$ echo 'conflict test1' > issue88.txt
linuxer@linuxer-PC:~/libgit2$ git commit -a -m 'issue88.txt updated'

linuxer@linuxer-PC:~/libgit2$ git checkout master
linuxer@linuxer-PC:~/libgit2$ echo 'conflict test2' > issue88.txt
linuxer@linuxer-PC:~/libgit2$ git commit -a -m 'issue88.txt updated'

linuxer@linuxer-PC:~/libgit2$ git merge conflict
자동 병합: issue88.txt
충돌 (내용): issue88.txt에 병합 충돌
자동 병합이 실패했습니다. 충돌을 바로잡고 결과물을 커밋하십시오.

linuxer@linuxer-PC:~/libgit2$ git status                (현재 상황 확인)
linuxer@linuxer-PC:~/libgit2$ git merge --abort         (병합 중단)
```

Git 브랜치 활용

❑ 머지(merge)(5)

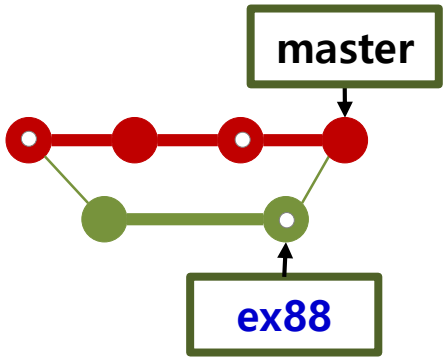
- 각 브랜치에서 동일한 파일을 작업할 때 충돌이 자주 발생
- 충돌이 발생하면 충돌을 없애야 merge 가능함
  - 상황 파악을 위해 해당 파일의 다른 점 식별
  - merge 도구를 활용하면 해결에 도움이 됨: git mergetool
  - 양쪽 브랜치에서 해당 파일을 동일한 내용으로 수정
  - 필요없는 파일은 삭제

Git 브랜치 활용

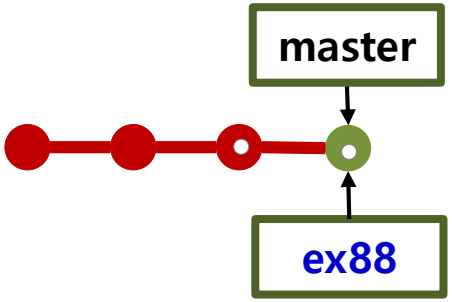
❑ 리베이스(rebase)(1)

- 브랜치를 다른 브랜치와 합치는 방법 중 하나
- merge는 브랜치에 대한 커밋 라인이 log에 남아 있음
- rebase는 커밋 라인을 재배치하여 log를 깔끔하게 볼 수 있게 함

<merge>



<rebase>



Git 브랜치 활용

❑ 리베이스(rebase)(2)

- 브랜치를 다른 브랜치와 합치는 방법 중 하나
  (주의) 반드시 결합할 브랜치로 checkout 후 진행할 것

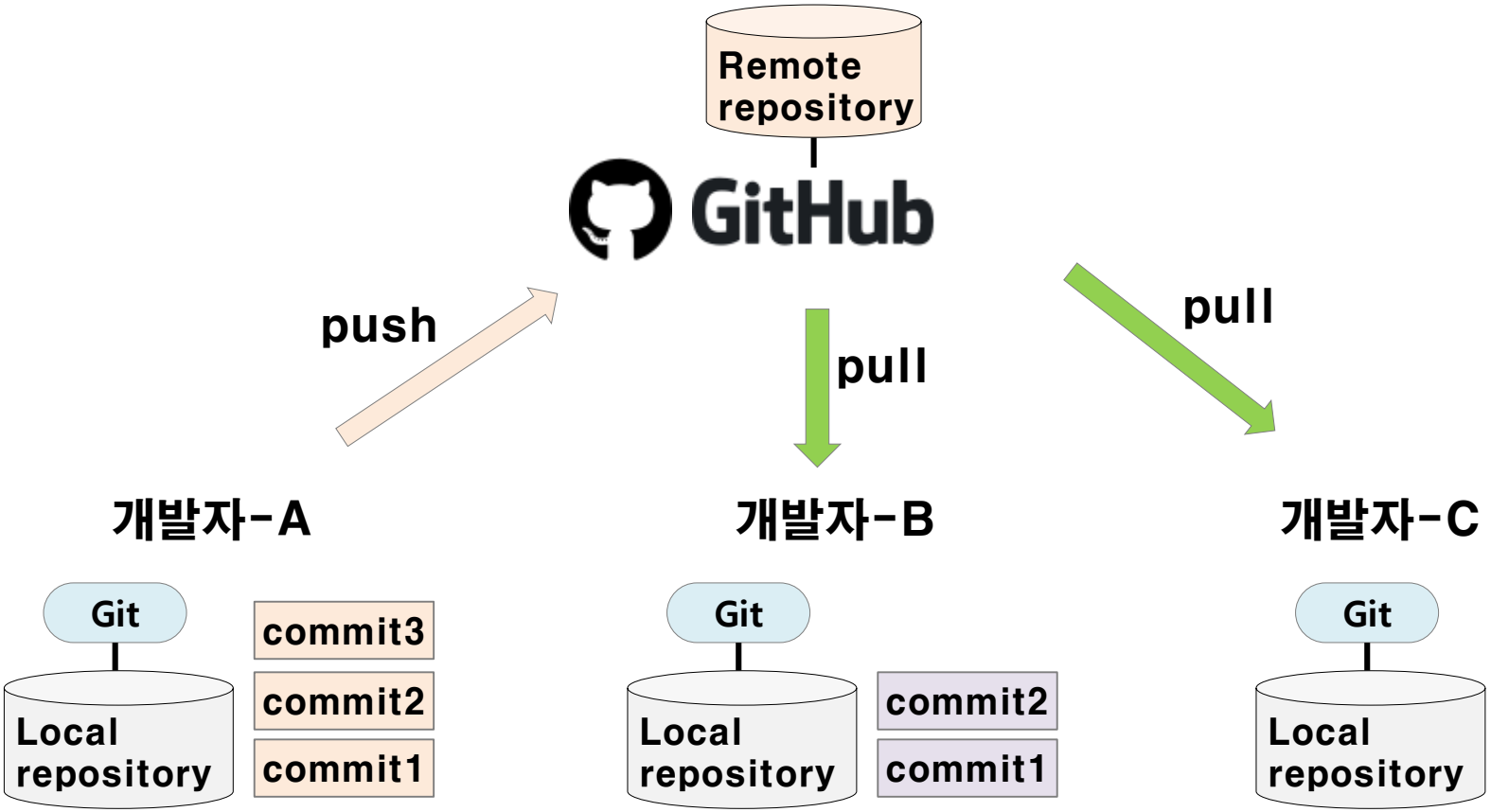
```
linuxer@linuxer-PC:~/libgit2$ git checkout -b ex88
linuxer@linuxer-PC:~/libgit2$ echo 'rebase test' > ex88.txt
linuxer@linuxer-PC:~/libgit2$ git add ex88.txt
linuxer@linuxer-PC:~/libgit2$ git commit -m 'ex88.txt added'

linuxer@linuxer-PC:~/libgit2$ git rebase master
Current branch ex88 is up to date.

linuxer@linuxer-PC:~/libgit2$ git log --oneline --decorate --graph -5
* 2ff460fa4 (HEAD -> ex88) ex88.txt added
*   672a5eada (master) Merge branch 'issue88'
|\
| * 5166ca974 (issue88) issue88.txt added
* | b4c7fd8bf b4.txt added
* | 2874ef255 b3.txt updated
|/
```

Git 원격 저장소 활용

- 원격 저장소(Remote Repository)
  - 내부 Git 서버 운영 (권장되지 않음)
    - Working 디렉터리가 없는 bare 저장소
    - Local, HTTP, SSH, Git 프로토콜 사용 가능
    - Git daemon으로 서버스 시작
  - Git 호스팅 서비스 (권장)
    : git.wiki.kernel.org/index.php/GitHosting
    - GitHub: https://github.com
    - Bitbucket: https://bitbucket.org
    - GitLab: https://about.gitlab.com

Git 원격 저장소 활용

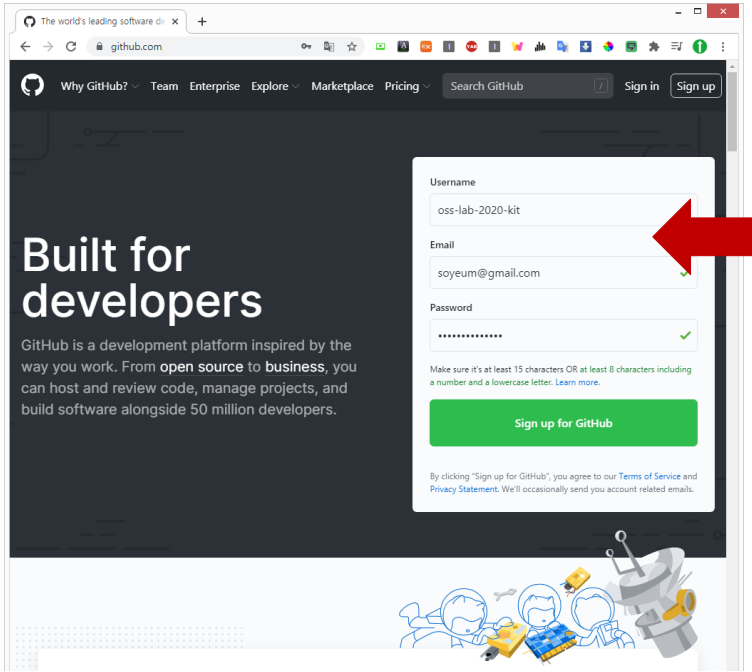
- GitHub(깃허브): https://github.com
  - 오픈 소스 활용과 오픈 소스 프로젝트 참가율 급상승



Git 원격 저장소 활용

❑ GitHub 등록 절차

- 아래 주소에 접근하여 계정 생성
  https://github.com
- 이름, 이메일, 비밀번호 입력 후
  "Sign up for GitHub" 클릭
- 직접 계정 등록하는 것인지를 확인하는
  이미지 선택 후 "Join a free plan" 클릭
- 각종 질문에 적절한 항목 선택 후
  "Complet setup" 클릭
- 등록된 메일로 실사용자 확인 메일 전송
  => 메일 열어서 "Verify email address" 클릭



Git 원격 저장소 활용

❑ GitHub 사용 흐름

① GitHub에 원격 저장소 생성

② 원격 저장소 복제(git clone)

③ 관리 대상 파일 작성 및 편집(vi)

④ 파일 생성, 변경, 삭제 내역을 Staging Area에 추가 (git add)

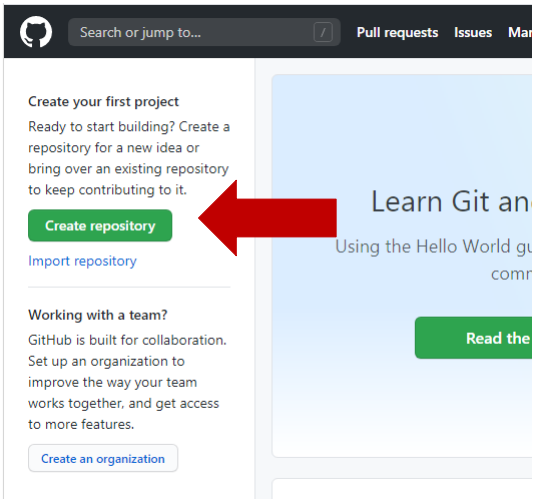
⑤ 변경 결과를 로컬 저장소에 commit (git commit)

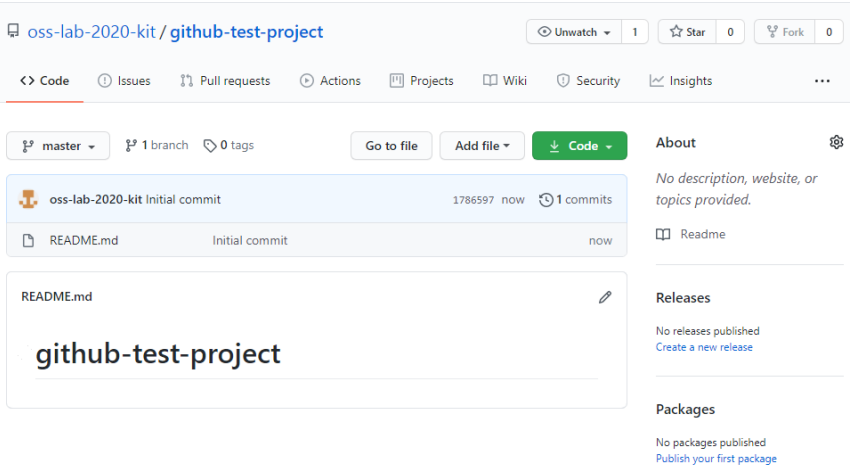
⑥ 로컬 저장소를 push해서 원격 저장소에 반영 (git push)

Git 원격 저장소 활용

① GitHub에 저장소 생성

- "Create Repository" 버튼 클릭
- Repository name: <임의 지정>
  ex> github-test-project
- Description: <none>
- 저장소 형태: Public (Private은 유료임)
- Add a README file : check
- Add .gitignore : uncheck
- Choose a license : uncheck
- "Create Repository.." 버튼 클릭

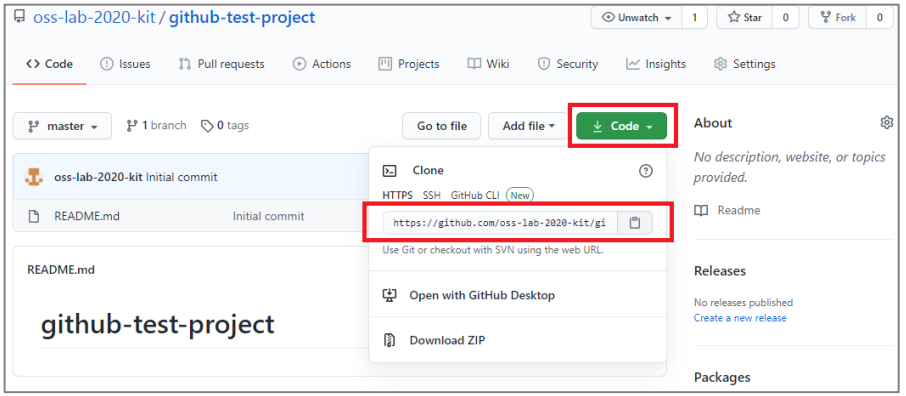




Git 원격 저장소 활용

② 원격 저장소 복제

- github.com에 로그인 후 원격 저장소 확인
- "Code" 클릭해서 HTTPS URL 복사
- 원격 저장소 복제 및 확인




```
linuxer@linuxer-PC:~$ git clone https://github.com/oss-lab-2020-kit/github-test-project.git
'github-test-project'에 복제합니다...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
오브젝트 묶음 푸는 중: 100% (3/3), 607 bytes | 607.00 KiB/s, 완료.

linuxer@linuxer-PC:~/github-test$ git remote -v
```

Git 원격 저장소 활용

③ 관리 대상 파일 작성 및 편집

- vi 를 이용해서 아래와 같은 코드 작성 후 hello.c 로 저장

```
linuxer@linuxer-PC:~/github-test$ vi hello.c

#include <stdio.h>

int main(int argc, char* argv[])
{
    int i;

    printf("Hello, World!₩n");
    printf("argc = %d₩n", argc);

    for( i = 0 ; i < argc ; i++ ) {
        printf("argv[%d] = %s₩n", i, argv[i]);
    }
    return 0;
}
:wq
```

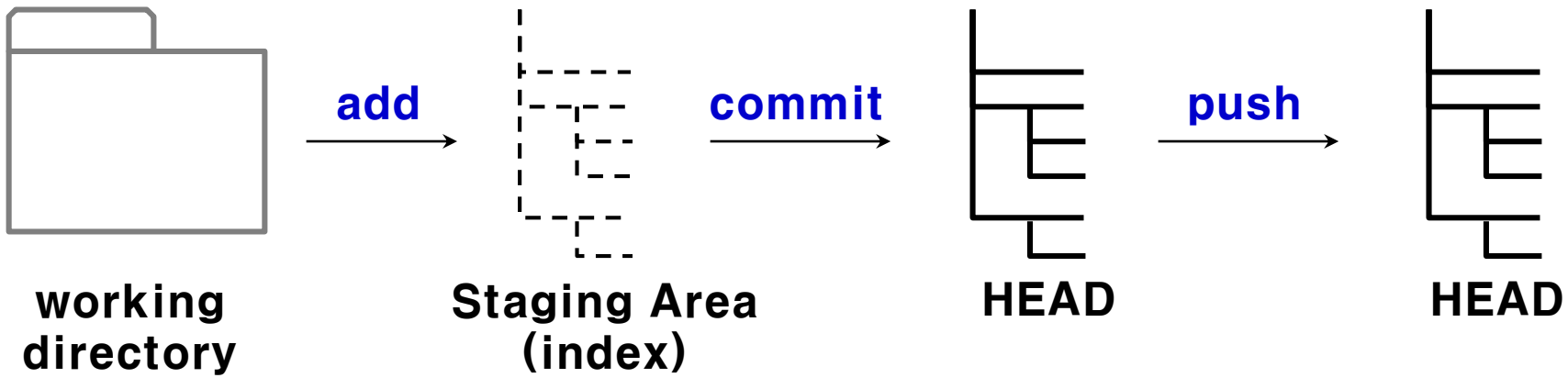
Git 원격 저장소 활용

④ 파일 생성, 변경, 삭제 내역을 Staging Area에 추가

```
linuxer@linuxer-PC:~/github-test$ git add hello.c
```

⑤ 변경 결과를 로컬 저장소에 commit

```
linuxer@linuxer-PC:~/github-test$ git commit -m 'new commit'
[master dda7872] new commit
 1 file changed, 15 insertions(+)
 create mode 100644 hello.c
```



Git 원격 저장소 활용

⑥ 로컬 저장소를 push해서 원격 저장소에 반영

```
linuxer@linuxer-PC:~/github-test$ git push origin master
Username for 'https://github.com': oss-lab-2020-kit
Password for 'https://oss-lab-2020-kit@github.com': *****
오브젝트 나열하는 중: 4, 완료.
오브젝트 개수 세는 중: 100% (4/4), 완료.
Delta compression using up to 2 threads
오브젝트 압축하는 중: 100% (3/3), 완료.
오브젝트 쓰는 중: 100% (3/3), 420 bytes | 420.00 KiB/s, 완료.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/oss-lab-2020-kit/github-test-project.git
   1786597..dda7872  master -> master
```

[복습] Git 활용

❑ Git command flow