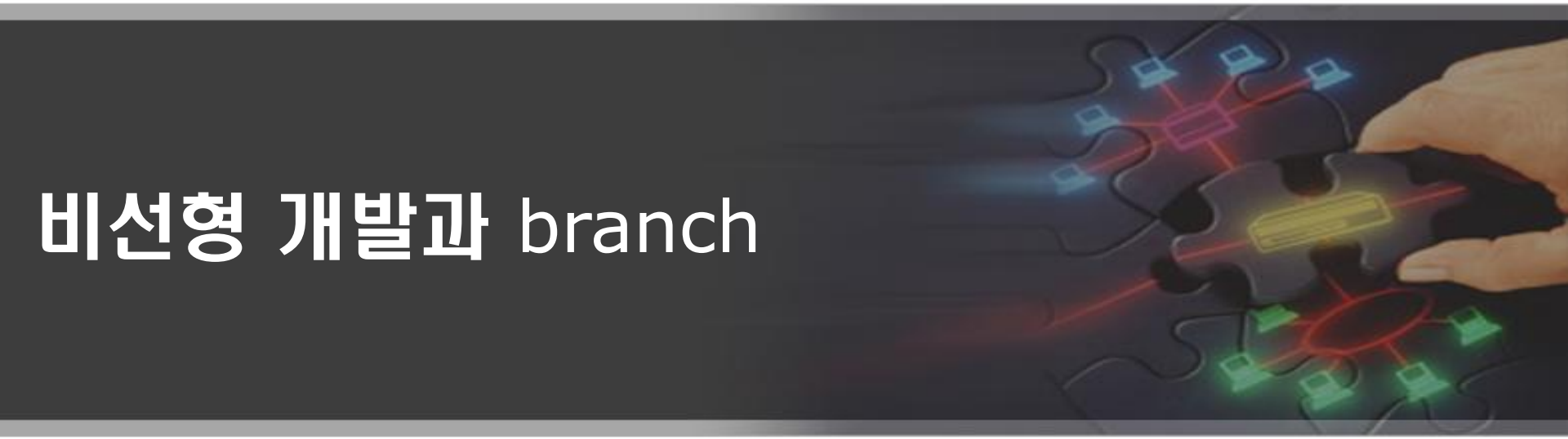
비선형 개발과 branch

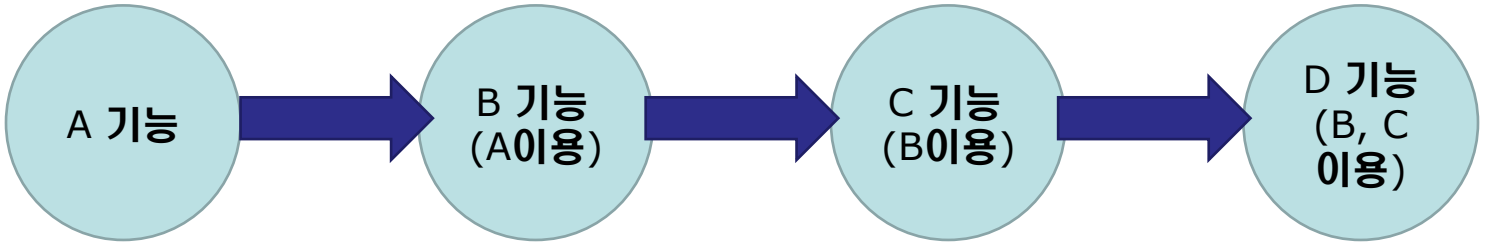


About..

컴퓨터소프트웨어공학과
김 원 일

선형 개발

- **선형 개발**
  - 대부분의 프로그램의 동작은 선형으로 이루어짐
  - 기존의 소프트웨어 개발 또한 선형, 즉 순서대로 개발이 이루어짐
  - 새로운 또는 특정 기능 개발을 위해 다른 기능들이 개발된 상태여야 함
  - 순서에 따라 개발되어야 하는 전체적인 개발 구조를 선형 개발이라 할 수 있음
  - 소프트웨어 대부분이 의존성을 가진 상태로 개발되므로 필수적임
  - 콘솔 프로그램 개발에서 화면 출력을 위한 printf 함수가 대표적인 예
    - printf 함수가 개발되어 있지 않다면 화면에 출력할 수 있는 방법이 없는 상태
    - 또는 printf 함수를 직접 구현하여 화면에 출력할 수 있도록 개발을 진행해야 함
    - printf 함수가 존재하지 않는 상태에서는 화면 출력을 테스트할 수 없음
    - 따라서 printf 함수를 먼저 개발하고, 관련된 기능을 개발하는 형태가 선행 개발

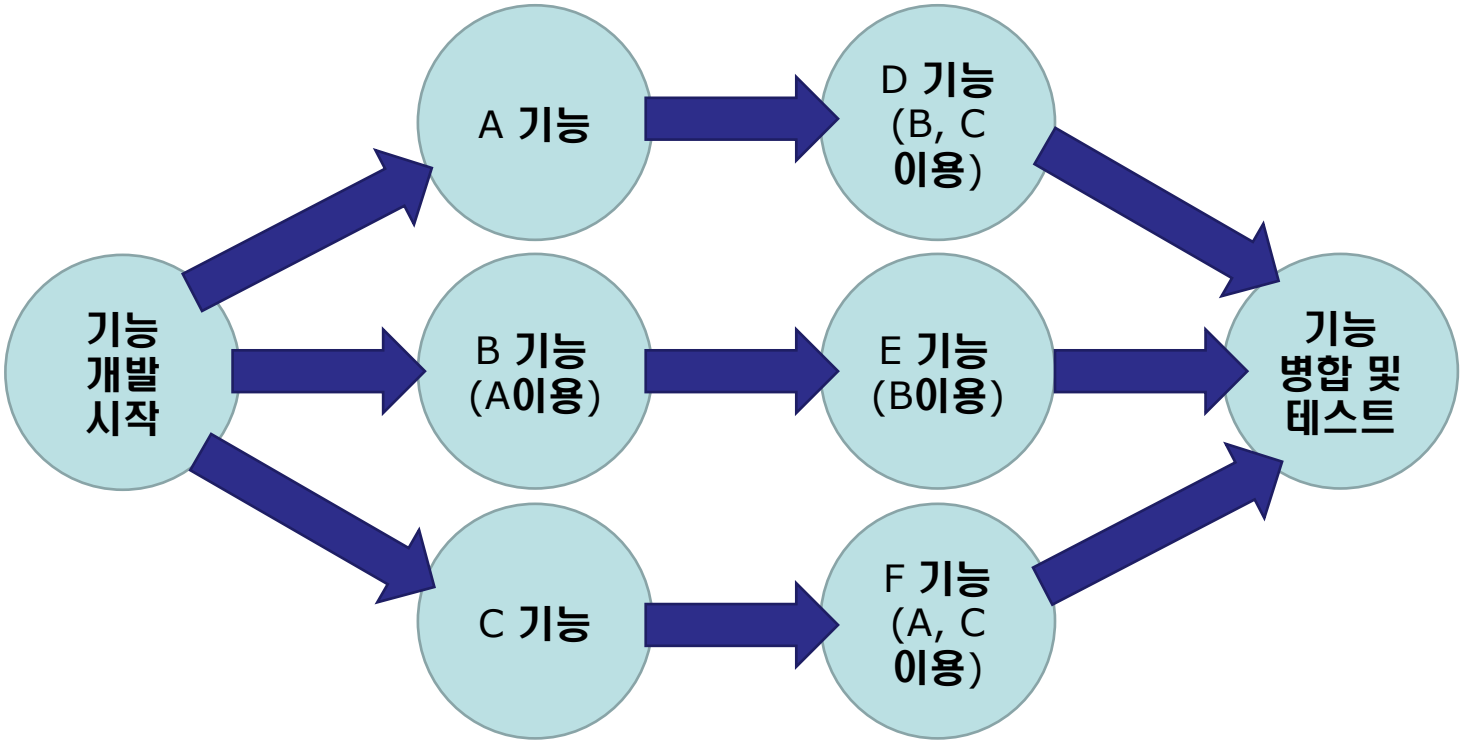


의존성

- **의존성**
  - **소프트웨어 개발 도중 기능 동작을 위해 선행/특정 조건/기능이 만족되어야 함**
    - **실행 시간을 측정하기 위한 소프트웨어는 현재 시간을 알 수 있는 기능이 필요**
    - **시간을 측정하는 조건/기능이 만들어지지 않는다면 해당 기능의 개발 진행이 어려움**
    - **웹에서 보안 기능 소프트웨어 없이 은행과 같은 보안을 요구하는 사이트 기능 사용**
  - **기능 또는 장치 등이 없는 없는 상태에서의 소프트웨어는 정상 동작 불가**
    - **무선 마우스 드라이버가 존재해도 무선 마우스가 없으면 소프트웨어 실행이 의미 없음**
    - **GUI가 없는 환경에서 GUI 프로그램을 실행시키는 경우, 동작하지 않음**
  - **JVM 또한 일종의 의존성이라고 볼 수 있음**
    - **JVM이 없는 상태에서 Java 코드를 실행할 수 있는 방법이 존재하지 않음**
    - **컴파일러와는 다르게 실행 환경인 JVM을 설치하지 않으면 동작하지 않음**
  - **OS는 모든 소프트웨어에 대한 환경적인 의존성을 가짐**
    - **프로그램이 실행될 수 있는 환경을 OS가 제공하기 때문**
    - **동작할 수 있는 환경이 존재하지 않거나 다르면 소프트웨어가 정상적으로 동작하지 않음**
    - **의존성은 소프트웨어뿐만 아니라 하드웨어도 동일하게 적용 받음**
    - **NFC 없이 각종 Pay 기능을 사용할 수 없는 것과 동일**

비선형 개발

- **비선형**
  - 순차적인 개발이 아닌 동시 다발적인 개발 방법
  - 한번에 하나의 기능이 아닌 다수 기능을 여러 개발자가 동시에 개발하는 방법
  - 비선형 개발을 위해서는 소스 코드에 대한 특별한 관리가 필요
  - 비선형 개발을 지원하는 대표적인 도구가 git이라 할 수 있음



비선형 개발 위한 지원

- **개발 시스템 및 소스 파일 설계 우선**
  - 시스템에 대한 분석과 기능 구현을 먼저 **설계**하고 개발을 시작
  - 소스 코드 개발에 대한 **분업**을 먼저 진행하는 것이 좋음
  - 수정할 파일과 수정하지 않을 파일을 구분하여 작업
  - 설계 이후 별도의 파일을 생성하거나 수정할 경우 문제가 발생 함
  - **지정된 파일 이외에는 수정하지 않는 것이 기본**
  - mockup(시제품) 형태의 코드 작성 방법으로 처리를 지원
- **브렌치**(branch) **활용**
  - 브렌치란 **원래 코드**와 관계 없이 독립 개발을 지원하는 논리적인 개념
  - 특정 버전 상태에서 논리적으로 분리된 파일들을 별도로 사용
  - main 브렌치에서 별도 브렌치를 생성하면 파일들을 별도로 사용 가능
  - 병합 후 업로드할 경우, 브렌치는 서버에 업로드하지 않아도 문제 없음
  - 필요한 경우에는 서버에 브렌치를 업로드해야 함

Commit에 대한 이해

- Commit 정보
  - 현재 관리 중인 파일/디렉터리의 변경 상태를 저장소에 반영하는 것
  - 이력을 관리하기 때문에 반드시 이전 상태가 있는 파일의 변경을 저장
  - 신규 파일은 추가된 시점부터 파일 이력을 관리하기 시작
  - 로컬 데이터베이스에 저장할 때 신규, 변경에 대한 정보도 같이 기록

  - Commit 시에 저장되는 정보
    - 파일의 현재 코드 상태와 코드의 스냅샷 관련 정보
    - 이전 코드와의 차이점에 대한 정보
    - 스냅샷의 무결성 확인을 위한 해시 값
    - Commit한 사용자의 이름(또는 별명)과 메일 주소

  - 같은 시점에 동시에 수정된 파일도 Commit하지 않으면 이전 상태가 유지
  - 항상 새로운 작업은 Commit한 이후에 수행하는 것을 권장
  - 중복 작업 관련 소스 코드를 동시에 Commit하면 확인해야 하는 정보가 많음
  - 하나의 기능 개발 도중에도 필요한 경우 여러 번 Commit해도 문제 없음
  - 비선형 개발이나 논리적 파일 분리를 위해서는 먼저 Commit 해야 함

브렌치 기초 – 1

- **브렌치는 일종의 게임 세이브와 동일**
  - 게임 세이브로 **현재 상태를 저장**할 수 있음
  - 저장 상태 후 플레이 도중 이전 상태로 되돌리고 싶을 경우 세이브를 로드
  - 이때, 현재 상태 저장은 사용자의 선택 등을 저장하지 않음
  - 원하는 시점을 로드하여 확인이나 수정 등이 가능

브렌치 기초 – 2

- **실습 환경**
  - **이전에 실습의** github **저장소를 복제한 디렉터리로 이동**
    - **이전 디렉터리 정보가 없는 경우,** github**에서** clone**으로 다시 가져옴**

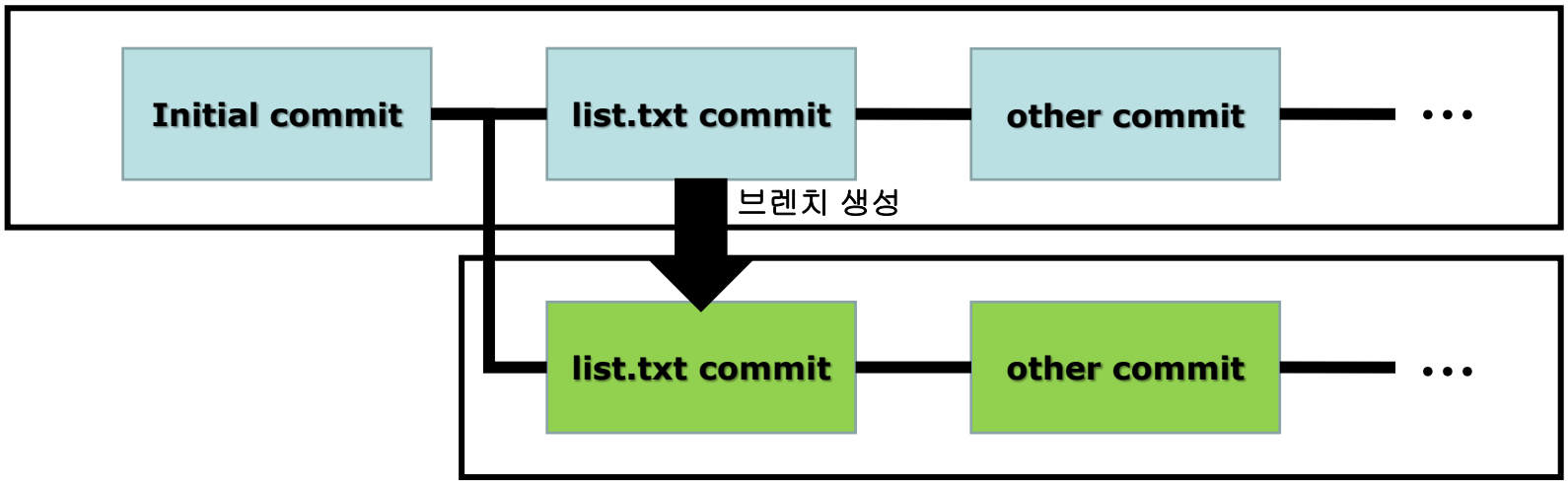
```
MINGW64:/d/202507001/testnew
YUHAN@PC7202 MINGW64 /d/202507001/testnew (main)
$ ls
LICENSE  README.md  list.txt

YUHAN@PC7202 MINGW64 /d/202507001/testnew (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

YUHAN@PC7202 MINGW64 /d/202507001/testnew (main)
$ git log
commit 25a8119964df6be2154665c328b09fbec5f4907a (HEAD -> main, origin/main, origin/H
EAD)
Author: wikim <unangel@yuhan.ac.kr>
Date:   Thu Apr 17 11:11:23 2025 +0900

    1. list.txt
     [ + ] test for github server

commit 10d8d52c2f49a7d3a1169a4ead18623a79c5d1b5
Author: ycs-wikim <80020439+ycs-wikim@users.noreply.github.com>
Date:   Thu Apr 17 10:43:13 2025 +0900

    Initial commit

YUHAN@PC7202 MINGW64 /d/202507001/testnew (main)
$
```

브렌치 기초 – 3

- **브렌치 확인과 생성 명령어**
  - "git branch"로 현재 브렌치 정보 확인
  - 현재 작업중인 브렌치는 앞에 "*"이 출력되고, 색상이 다르게 표시됨
  - 새로운 논리 파일 관리를 위해 브렌치를 생성
  - "git branch work"로 브렌치를 생성
  - 브렌치만 생성한 경우는 여전히 현재 브렌치가 작업 브렌치로 설정

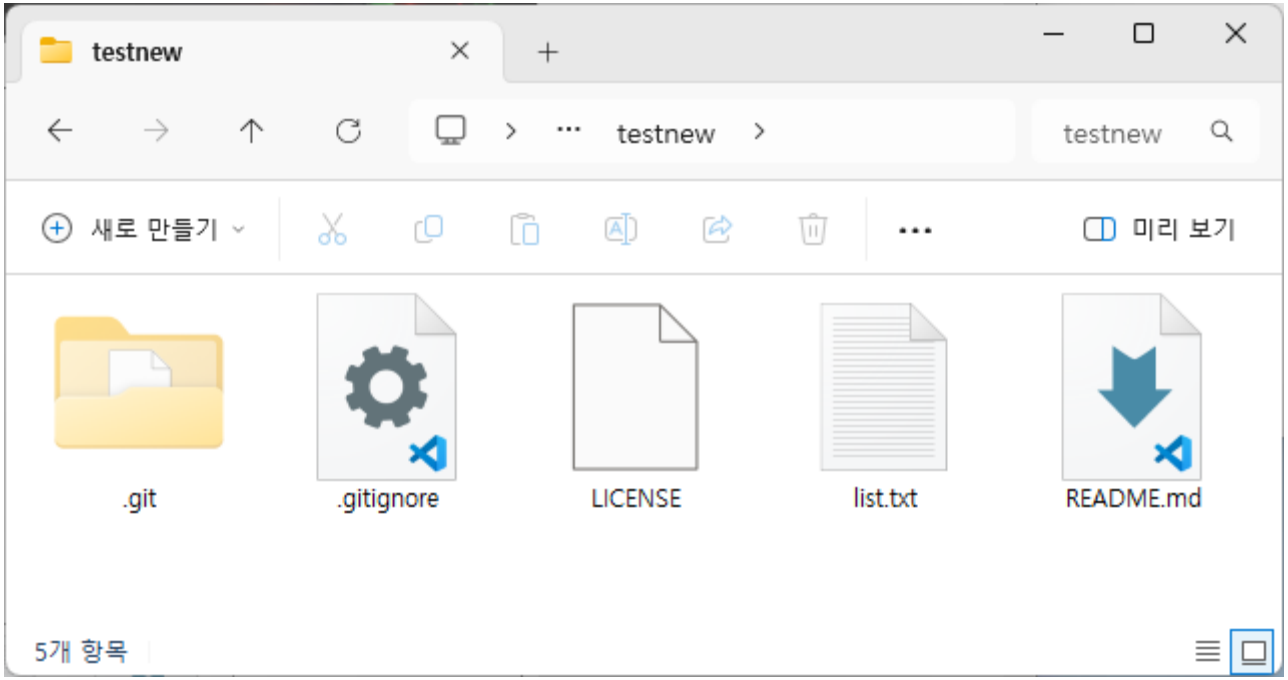
브렌치 기초 – 4

- **브렌치 생성 상태 정보**
  - **브렌치를 생성한 시점은 항상 현재 브렌치의** commit **상태로부터 시작**
  - work **브렌치는** list.txt **파일 추가 후의** Commit **상태를 논리적으로 분할**
  - **분할된 상태에서는 각 브렌치 별로 파일들이 별도로 관리하는 상태가 됨**
  - **각 브렌치에서 별도 파일 추가와** Commit **등은 각자 수행할 수 있는 상태**
  - **별도 파일로 인식되어 동일한 파일이라도 브렌치 별로 내용이 다를 수 있음**
  - **논리적으로 분리된 파일들은 서로 영향을 주지 않음**



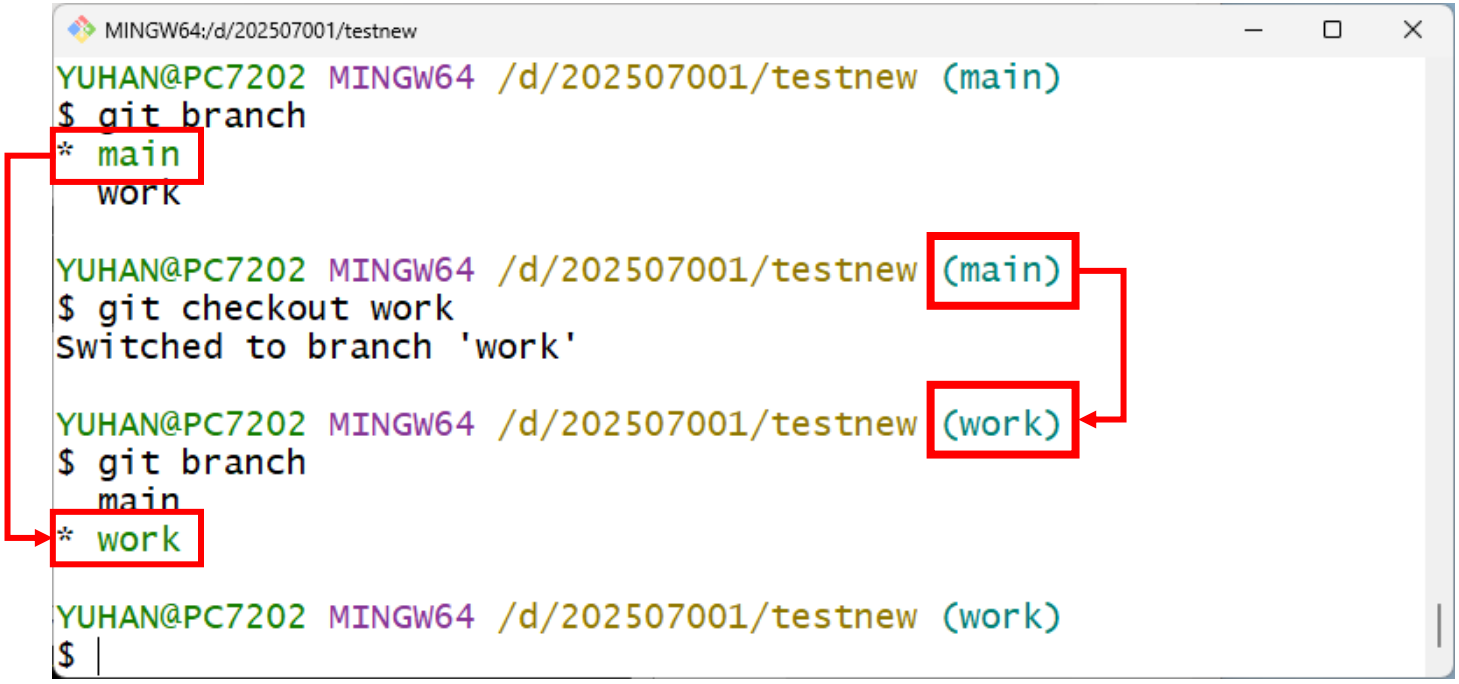
브렌치 기초 – 5

- **현재 디렉터리 상태**
  - **브렌치 별로 차이점을 확인하기 위해 main 브렌치의 현재 파일 상태 확인**
  - **현재 디렉터리는 추가한 파일과 ".git/" 디렉터리를 포함하고 있음**
  - **5개의 파일/디렉터리가 존재하고 있는 상태로 별도의 추가된 파일이 없음**
  - **현재 기준으로 브렌치를 변경하여 작업을 진행할 예정**



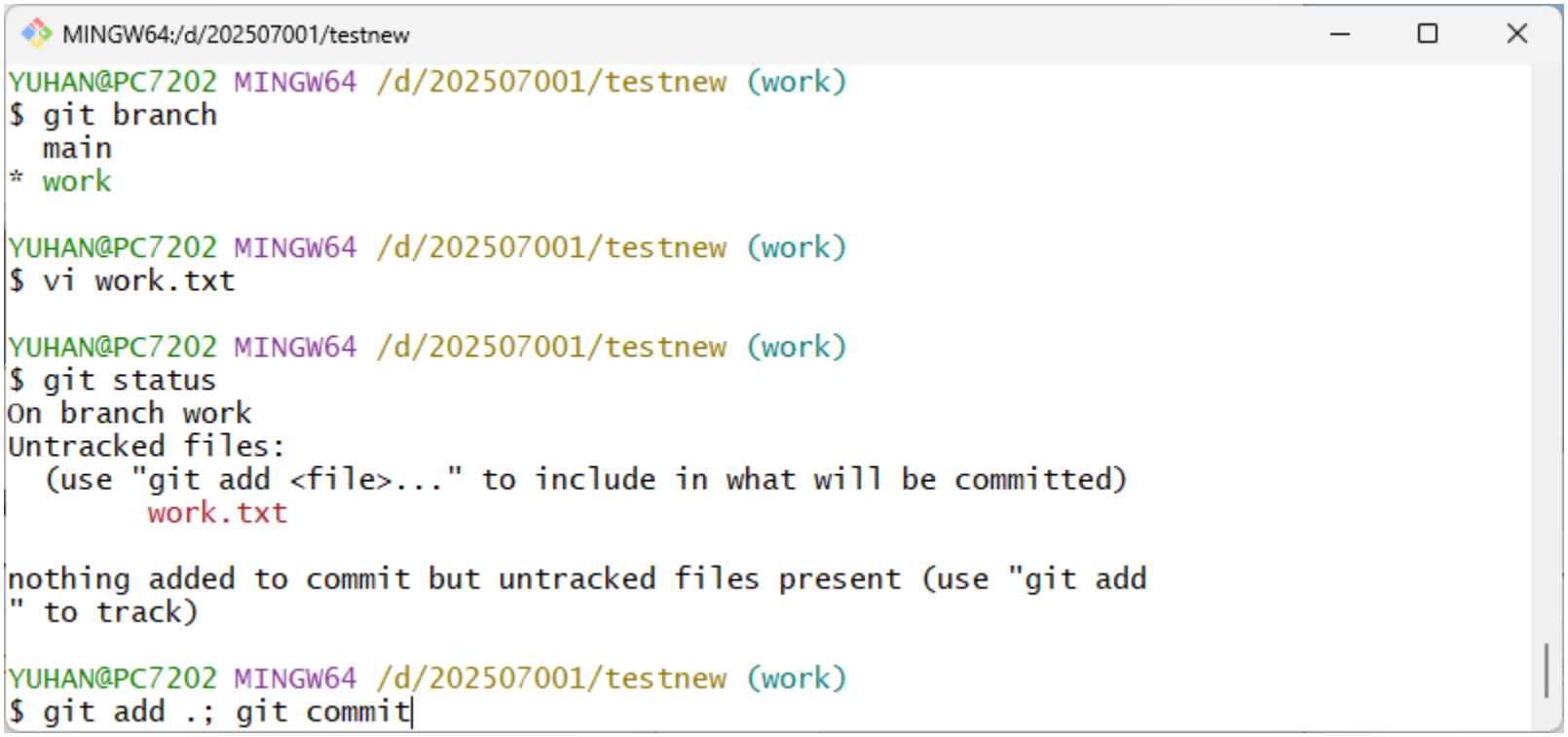
브렌치 기초 – 6

- **브렌치 이동**
  - **브렌치 정보 확인** : git branch
  - **다른 브렌치로 이동 명령어** : git checkout "**브렌치명**"
  - **이동한 후에 경로에 출력되는 브렌치 이름으로 현재 브렌치 확인 가능**
    - **브렌치 변경을 "Switch"라고 부르기도 함**
  - **현재까지** commit**되어 있는 상태에서 브렌치가 생성됨**
  - **현재 상태 이전까지는 동일한 정보를 사용할 수 있는 논리적인 상태**



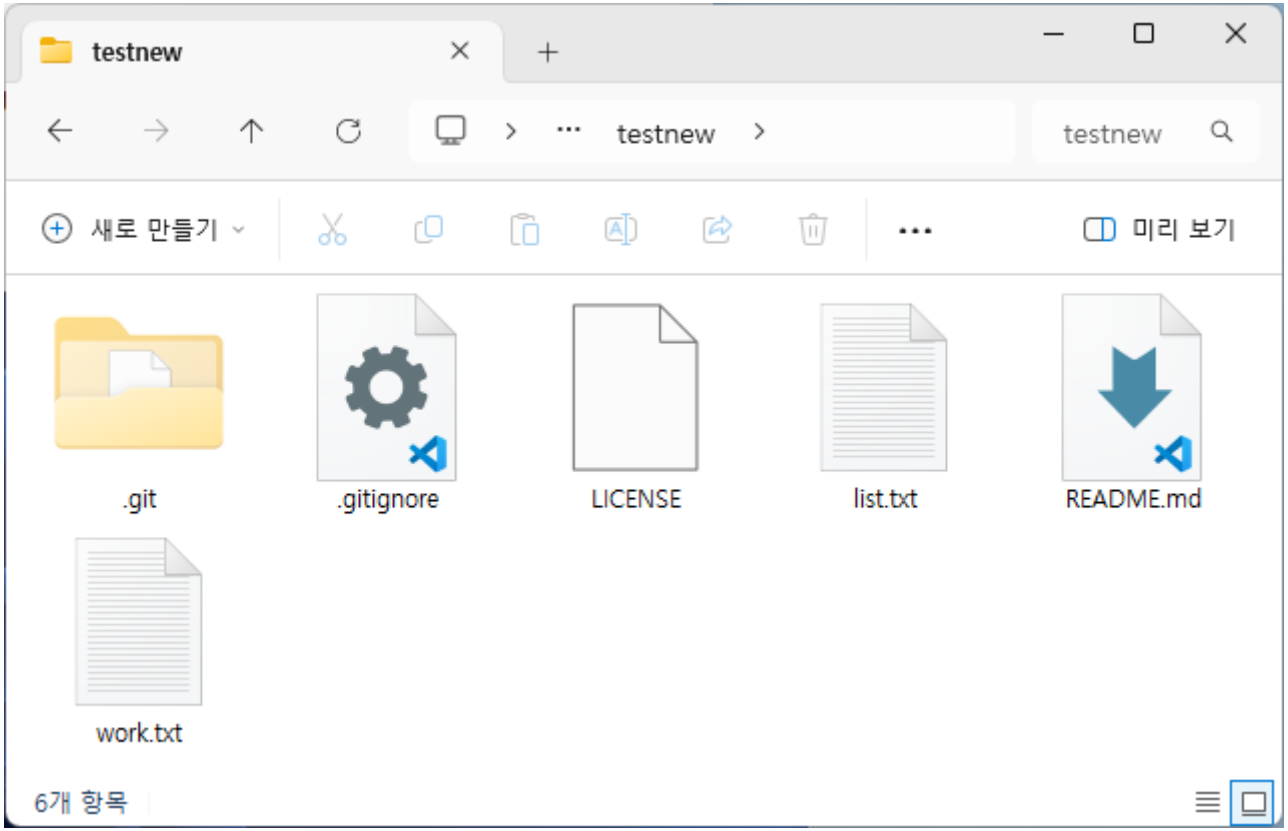
브렌치 기초 – 7

- **파일 추가 및** commit
  - **변경한** “work” **브렌치에서 새로운 파일을 추가** : work.txt
  - **추가된 파일을** staging**하고,** commit**하여 데이터베이스에 기록**
  - commit**까지 진행하면 모든 파일이 관리되고 있는 상태**



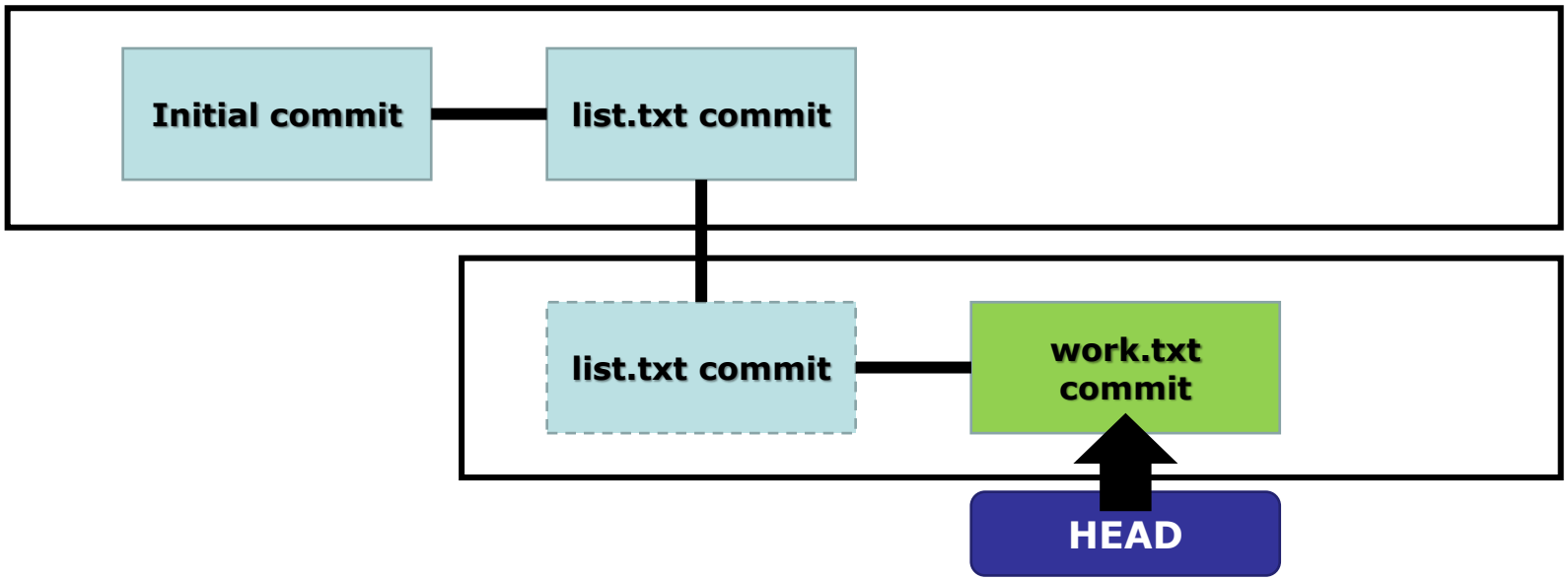
브렌치 기초 – 8

- **현재 디렉터리 상태**
  - **탐색기에서 숨김 파일 표시로 디렉터리 내의 파일과 디렉터리 확인**
  - work.txt **파일을 추가하여 파일/디렉터리의 개수가 6개로 변경된 상태**



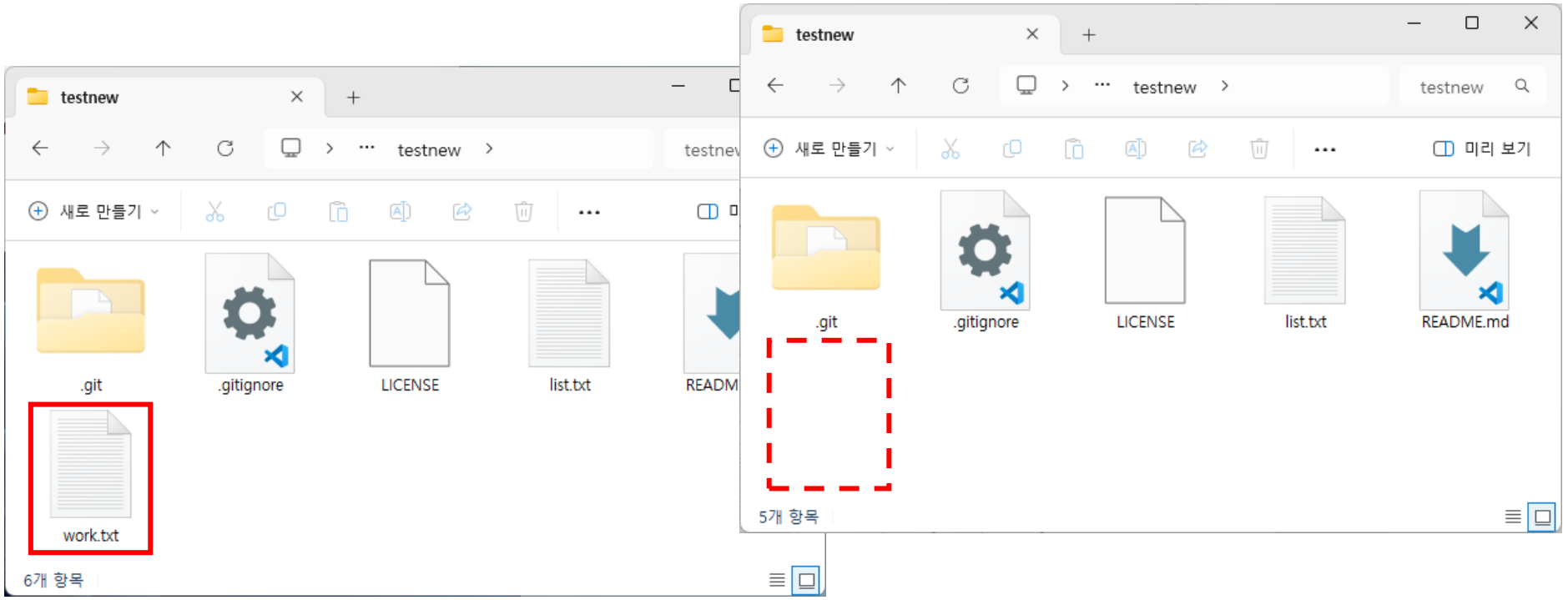
브렌치 기초 – 9

- **현재 작업 디렉터리**
  - **현재 작업 중인 디렉터리 위치는 아래 그림의 녹색 사각형 위치**
  - **현재 작업 중인 위치는 항상** HEAD**라는 포인터가 가리키고 있음**
  - **브렌치 변경은 사실** HEAD**가 다른** Commit**을 가리키도록 변경하는 것**
  - HEAD**는** git **명령어를 사용할 때도 필요에 따라 사용 가능**
  - HEAD**는 저장소에서 항상** 1**개의** Commit**만 가리킬 수 있음**
  - **브렌치는 결국** HEAD**가 가리키는** Commit**에 따라 파일을 표시**



브렌치 기초 – 10

- **브렌치 이동**
  - main**과** work **브렌치를** “git checkout” **명령으로 이동하면서 확인**
  - work **브렌치에서는** work.txt **파일이 보임**
  - main **브렌치에서는** work.txt **파일이 없어짐**
  - **논리적으로 분리되어 있는 상태이기 때문에 브렌치에 따라 파일이 달라짐**



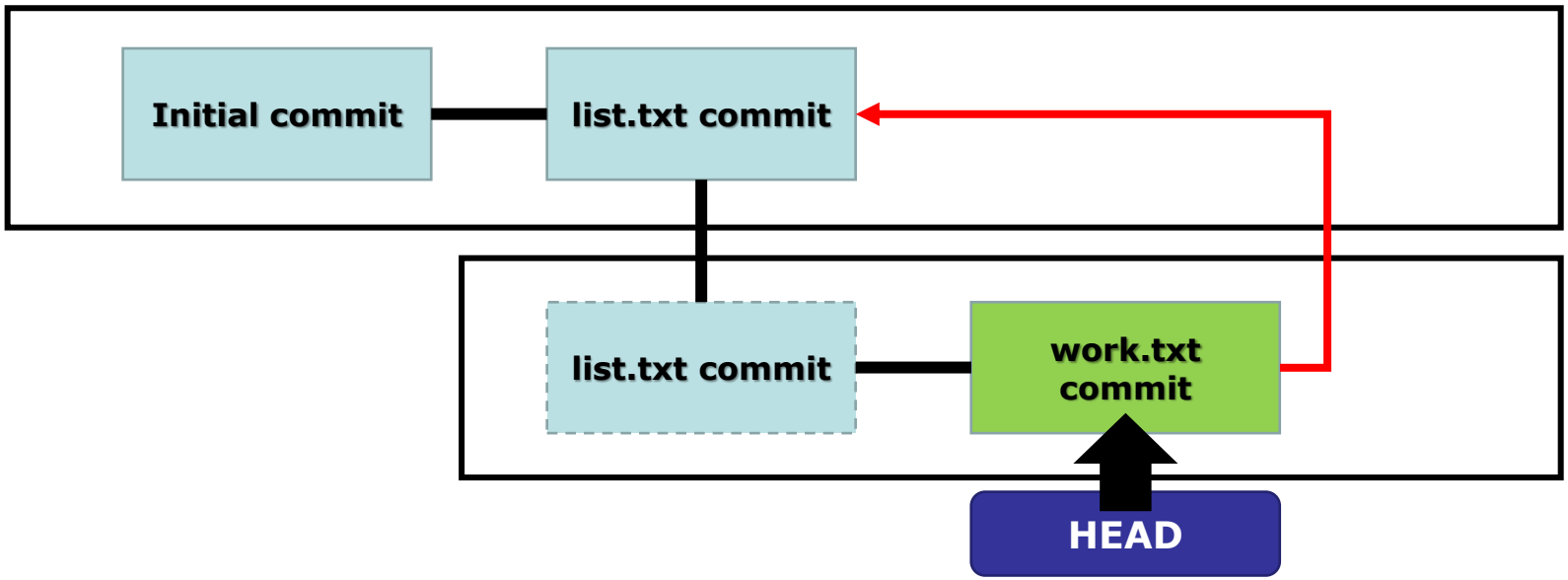
브렌치 기초 – 11

- **추가 작업**
  - **브렌치의** Commit **작업이 미치는 영향을 확인하기 위한 실습**
  - work **브렌치로 변경하고, 추가적으로** 2~3**개 정도의** commit **수행**
  - **브렌치에서 수행한 다수의** Commit**에서 마지막 정보는 확인할 수 있도록 구성**
    - **파일 내용의 마지막에 마지막임을 알 수 있도록 내용을 기록**
    - Commit **메시지에도** work **브렌치에서 마지막으로 수행한** Commit**임을 표시**

```
MINGW64:/f/202507001/testnew
commit ec1e4a5296dbca7f4ec24598afcbc5fc79a974b7 (HEAD -> work)
Author: wikim <unangel@yuhan.ac.kr>
Date:   Wed May 7 20:52:46 2025 +0900

    last file modify

commit aa6ece54dbff3b6ecbc84ff46848b54874eec349
Author: wikim <unangel@yuhan.ac.kr>
Date:   Wed May 7 20:52:03 2025 +0900

    modify file

commit 6be5b910b0adde61184c72e645086caa19250180
Author: wikim <unangel@yuhan.ac.kr>
Date:   Wed May 7 20:51:38 2025 +0900

    modify file

commit 25a8119964df6be2154665c328b09fbec5f4907a (origin/main, origin/HEAD, main)
Author: wikim <unangel@yuhan.ac.kr>
:
```

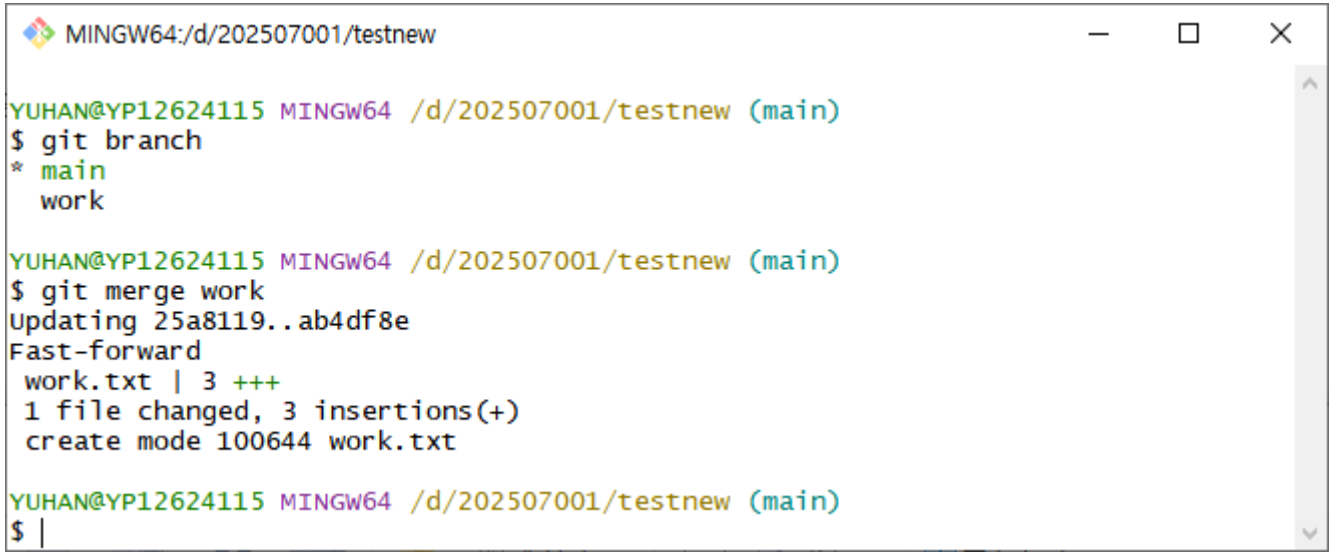
브렌치 기초 – 11

- **브렌치 병합**
  - 저장소에 main 브렌치와 work 브렌치가 같이 존재하는 상태
  - 2개로 분리된 브렌치는 병합을 통해 합쳐서 별도 작업을 합칠 수 있음
  - 병합할 때는 항상 현재 브렌치가 기준이 되므로 주의해야 함
    - 현재 브렌치로 다른 브렌치 정보를 가져와서 파일을 합치므로 주의
  - 메인 브렌치로 work 브렌치에서 작업한 내용을 병합
  - 현재 브렌치가 work 브렌치라면 정상적으로 동작하지 않는 실습



브렌치 기초 – 12

- **브렌치 병합**
  - "git merge" **명령으로 다른 브렌치를 현재 브렌치로 합칠 수 있음**
  - **사용** : git merge **병합하려는 브렌치 이름**
  - main **브렌치에** work **브렌치를 병합** : git merge work
  - **브렌치가 병합되고**, work.txt**가** main **브렌치에도 나타남**
  - work.txt**는 2개의 브렌치에 동시에 존재하면서 서로 다른 파일로 인식**
  - **병합하기 전, 후에도 여전히 다른 파일로 인식**



브렌치 기초 – 13

- **브렌치 병합 후 상태**
  - main과 work 브렌치를 이동하면서 work.txt 파일의 내용을 확인
  - main과 work 브렌치에서 발생한 Commit 로그 내용을 확인

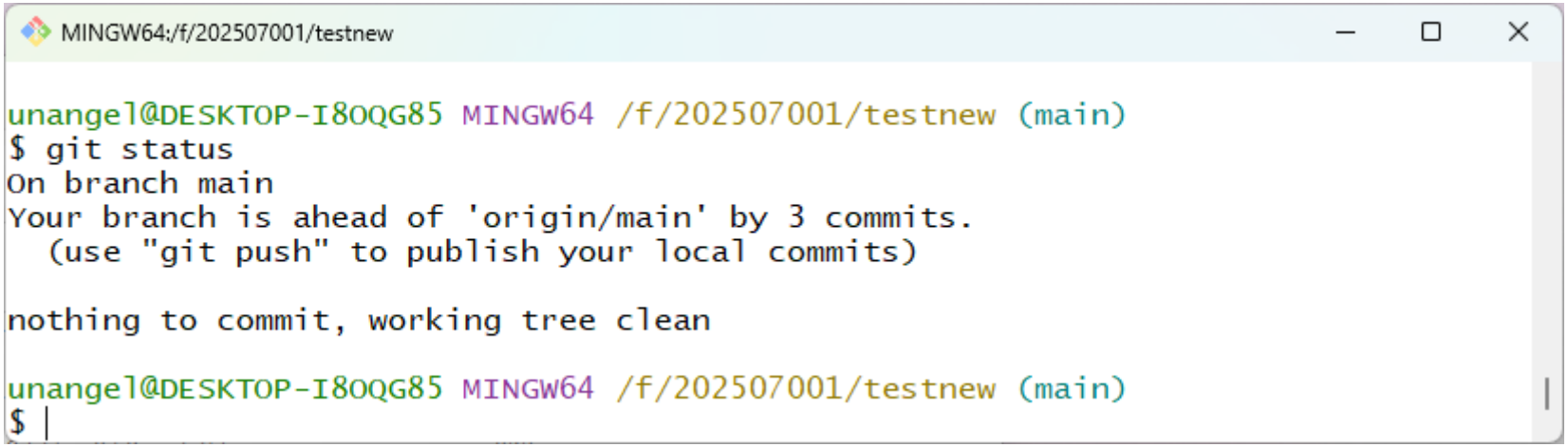
```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git log
commit ec1e4a5296dbca7f4ec24598afcbc5fc79a974b7 (HEAD -> main, work)
Author: wikim <unangel@yuhan.ac.kr>
Date:   Wed May 7 20:52:46 2025 +0900

    last file modify

commit aa6ece54dbff3b6ecbc84ff46848b54874eec349
Author: wikim <unangel@yuhan.ac.kr>
Date:   Wed May 7 20:52:03 2025 +0900

    modify file

commit 6be5b910b0adde61184c72e645086caa19250180
Author: wikim <unangel@yuhan.ac.kr>
Date:   Wed May 7 20:51:38 2025 +0900

    modify file

commit 25a8119964df6be2154665c328b09fbec5f4907a (origin/main, origin/HEAD)
Author: wikim <unangel@yuhan.ac.kr>
Date:   Thu Apr 17 11:11:23 2025 +0900

    1. list.txt
     [ + ] test for github server

commit 10d8d52c2f49a7d3a1169a4ead18623a79c5d1b5
Author: ycs-wikim <80020439+ycs-wikim@users.noreply.github.com>
Date:   Thu Apr 17 10:43:13 2025 +0900

    Initial commit

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$
```

로컬 저장소의 브렌치와 HEAD 정보

원격 저장소의 브렌치와 HEAD 정보

브렌치 기초 – 14

- **저장소 상태 확인**
  - "git status" **명령으로 현재 저장소의 정보를 확인**
  - "origin/**브렌치이름**"**으로 나타난 정보는 원격지 저장소를 의미**
  - **로컬에서 원격 저장소와 달리 추가로** 3**개의** Commit**이 추가됨**
  - **원격 저장소와 로컬 저장소의 보관 상태가 달라진 상태**
    - **원격 저장소는 이전 정보를 유지하고 있는 상태**
    - **로컬 저장소는 추가 및 새로운 정보를 유지하고 있는 상태**
  - **저장소가 동일한 정보를 갖도록 구성하려면** "git push"**로 동기화 수행**
  - **동기화하지 않으면 서로 다른 정보가 유지되므로 판단해야 함**



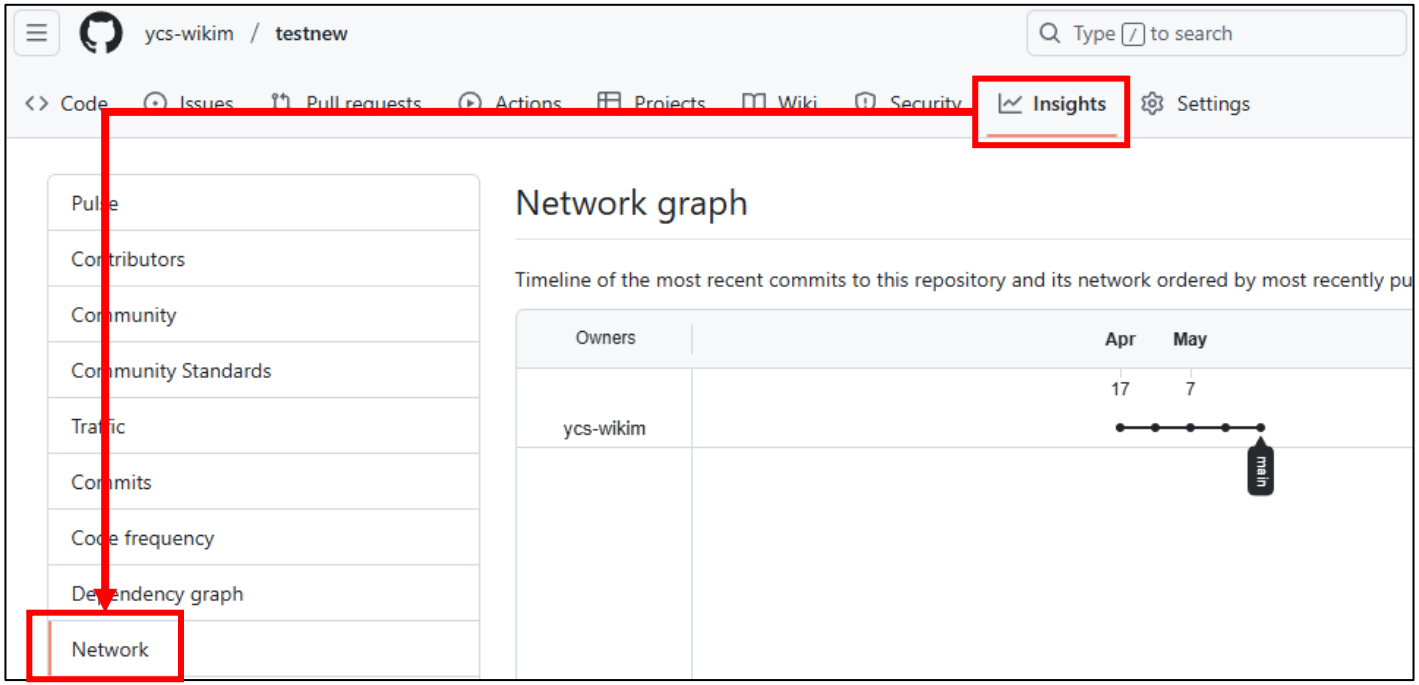
브렌치 기초 – 15

- **원격 저장소와 동기화**
  - "git push"**로 원격과 로컬 저장소 동기화를 수행**
  - **명령어가 정상적으로 수행된다면 버전 관리가 정상적으로 이루어짐을 의미**
  - **만약 오류가 발생하였다면 버전 관리에 문제가 발생하였음을 의미**
  - **버전 관리에 문제가 발생한 상태를 보통 충돌**(Collision)**이라 함**
  - **코드 간의 차이로 인해** git**이 판단할 수 없거나 합칠 수 없는 상태를 의미**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git push
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 16 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (9/9), 777 bytes | 777.00 KiB/s, done.
Total 9 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 1 local object.
To https://github.com/ycs-wikim/testnew
   25a8119..ec1e4a5  main -> main

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$
```

브렌치 기초 – 16

- github의 정보 확인
  - 브렌치 동기화 이후의 원격 저장소의 상태 확인
  - github 상단 "Insights" 메뉴에서 "Network" 메뉴를 선택하여 확인
  - main 브렌치와 그 동안의 Commit에 대한 정보를 확인할 수 있음
  - 로컬 저장소에 존재하는 "work" 브렌치에 대한 정보가 표시되지 않음
  - 브렌치는 원격 저장소에 업로드하지 않았기 때문으로 보임



브렌치 기초 – 17

- **브렌치 정보 관리**
  - **원격과 로컬 브렌치는 별도로 관리되기 때문에 브렌치도 동기화를 수행해야 함**
  - **로컬 저장소에서 "work" 브렌치로 전환하여 원격지로 업로드**
  - **업로드는 "git push" 하나이므로 브렌치 변경 후에 사용**
  - **로컬에서 작성하는 브렌치는 서버에도 별도로 생성해야 하므로 오류 발생**
    - **"git push"는 현재 브렌치를 원격지로 업로드한다는 의미이므로 브렌치 생성과는 다른 의미**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git checkout work
Switched to branch 'work'

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$ git push
fatal: The current branch work has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin work

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$
```

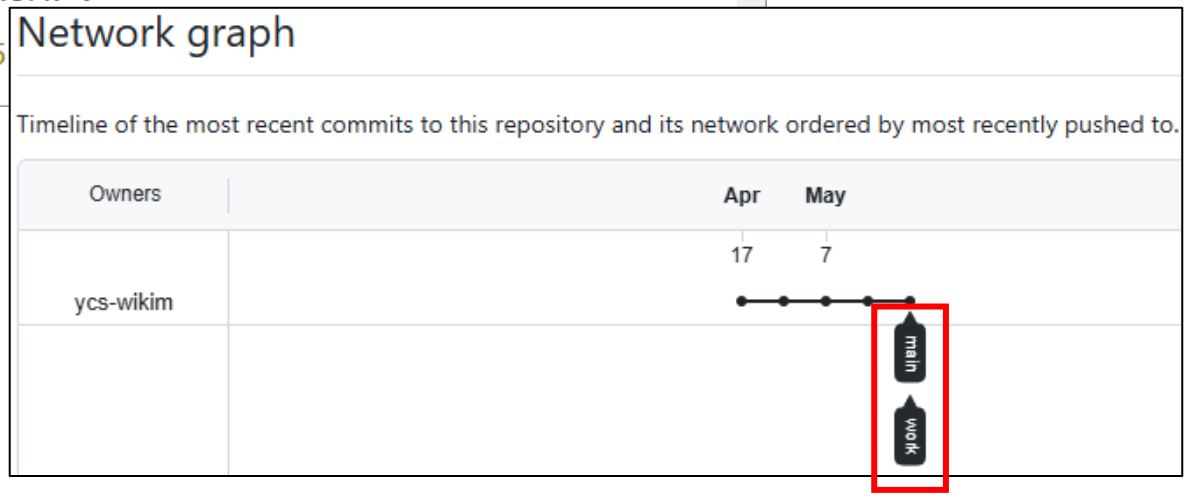
원격 저장소에 work 브렌치 생성과 업로드 명령어

원격 저장소에 새로운 브렌치를 자동으로 업로드하는 설정확인을 위한 웹 페이지

브렌치 기초 – 18

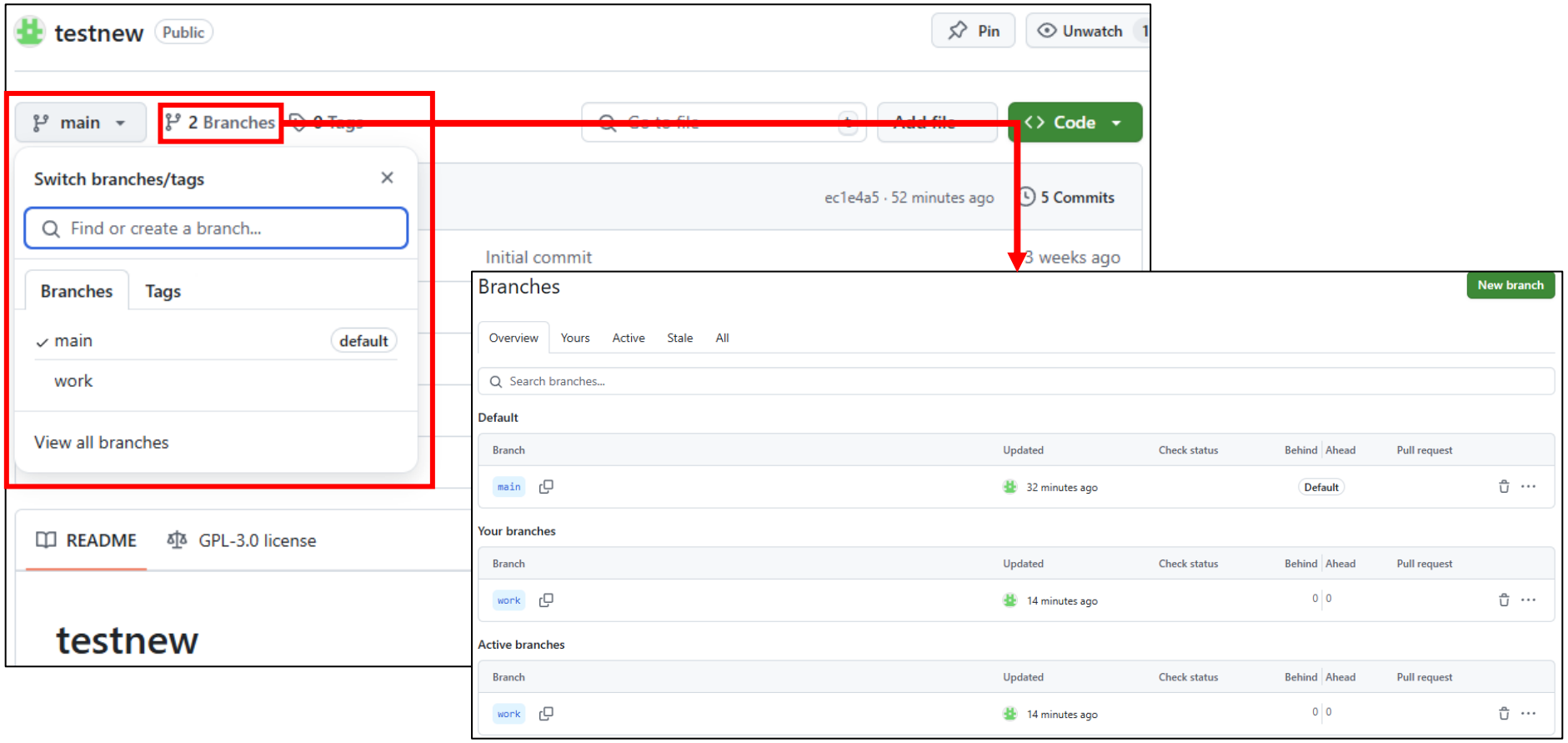
- **브렌치 업로드**
  - "git push --set-upstream origin **브렌치이름**"으로 브렌치 업로드
  - **업로드 후** github**에 브렌치가 나타나는 것을 확인할 수 있음**
  - Quiz : **논리적으로 구분된 브렌치 2개가 동일한 위치로 표시되는 이유는?**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$ git push --set-upstream origin work
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'work' on GitHub by visiting:
remote:      https://github.com/ycs-wikim/testnew/pull/new/work
remote:
To https://github.com/ycs-wikim/testnew
 * [new branch]      work -> work
branch 'work' set up to track 'origin/work'.

unangel@DESKTOP-I8OQG85 MINGW64 /f/2025
$
```



브렌치 기초 – 19

- github의 브렌치 정보
  - 메인 페이지에서 브렌치를 생성, 확인 또는 검색할 수 있음
  - 브렌치에 대한 상세한 정보를 제공하는 페이지도 별도로 존재



브렌치 기초 – 20

- **브렌치 삭제** - 1
  - **로컬 저장소에서 브렌치 삭제 명령** : git branch –d "**삭제할 브렌치 이름**"
  - **먼저** main **브렌치로 이동한 다음** work **브렌치를 삭제**
  - **브렌치 삭제가 정상적으로 이루어지고, 브렌치 목록에서도 삭제**
  - work **브렌치는 로컬에서만 삭제되었고, 원격에는 존재하는 상태**
  - **원격과 로컬 저장소는 별도로 관리되기 때문에 원격과 로컬에 여전히 존재**

브렌치 기초 – 21

- **삭제된 브렌치로 이동**
  - **브렌치 삭제 이후 브렌치 목록에 work 브렌치가 출력되지 않음**
    - **출력되지 않더라도 삭제된 브렌치로 이동이 가능**
    - **이동될 때 origin 즉, 원격지 정보를 이용하여 생성되는 것을 확인할 수 있음**
  - **브렌치가 이미 삭제된 상태이므로 원격지 정보로 새로운 브렌치를 생성**
    - **생성한 work 브렌치는 원격지 정보로 복원되어 이동하는 것을 확인할 수 있음**
  - **내부적으로 원격 저장소와 로컬 저장소는 완전히 분리되어 관리되고 있음**
  - **즉, 삭제는 로컬에서만 발생했고, 원격에는 발생하지 않은 상태**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git branch
* main

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git checkout work
branch 'work' set up to track 'origin/work'.
Switched to a new branch 'work'

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$ git branch
  main
* work

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$
```

원격 저장소에 업로드 된 브렌치는 복구 가능

브렌치 기초 – 22

- **브렌치 삭제** – 2
  - **서버로는 업로드하지 않고, 병합하지 않은 브렌치 생성과 삭제**
  - **로컬에서만 작업하고 삭제할 새로운 브렌치 생성**
  - "git checkout –b del" **로 새로운 브렌치를 생성하고 이동**
  - "git checkout –b": **브렌치 생성과 브렌치 이동을 한번에 수행하는 명령 옵션**
  - **새로운 파일을 추가하고**, commit **수행**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$ git checkout -b del
Switched to a new branch 'del'

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (del)
$ vi del.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (del)
$ git add del.txt ; git commit -m "add del.txt"
warning: in the working copy of 'del.txt', LF will be replaced by CRLF the next
time Git touches it
[del da73fba] add del.txt
 1 file changed, 3 insertions(+)
 create mode 100644 del.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (del)
$
```

브렌치 기초 – 23

- **브렌치 삭제** – 3
  - work **브렌치로 이동하여** del **브렌치를 삭제**
  - **병합되지 않은 로컬 브렌치 삭제에 대한 오류 메시지 출력**
  - **로컬에만 존재하는 브렌치는 완전히 삭제되기 때문에 오류로 처리**
  - **완전 삭제로 데이터 복구가 불가능하므로 삭제 전에 주의 필요**
  - **완전히 제거하려면** "git branch –D **삭제브렌치명**"**으로 삭제해야 함**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (del)
$ git checkout work
Switched to branch 'work'
Your branch is up to date with 'origin/work'.

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$ git branch -d del
error: the branch 'del' is not fully merged
hint: If you are sure you want to delete it, run 'git branch -D del'
hint: Disable this message with "git config set advice.forceDeleteBranch false"

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$ git branch -D del
Deleted branch del (was da73fba).

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$
```

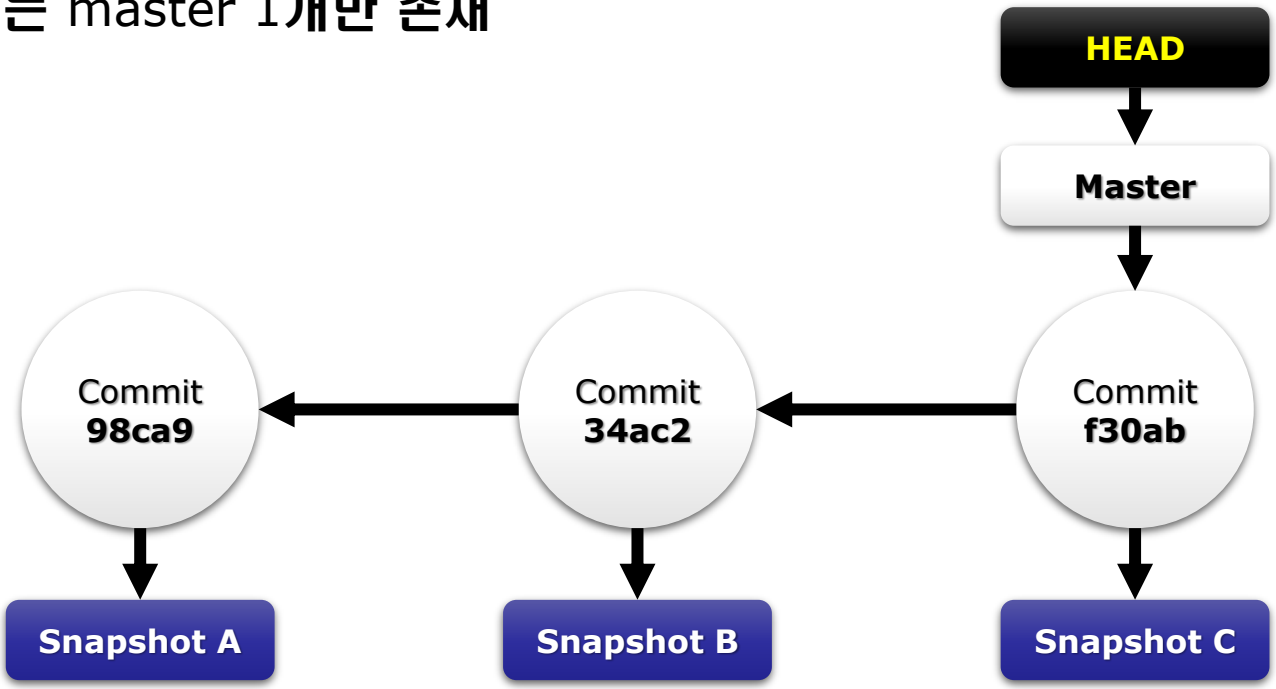
브렌치 기초 – 24

- **브렌치 삭제** – 4
  - **로컬에만 존재하는 브렌치의 삭제는 복구가 불가능**
  - **서버에 존재하는 것과 달리** "git checkout del"**로 이동할 수 없음**
  - **기존 작업 내용은 모두 삭제되므로 테스트 용도로 사용할 경우에 편리**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$ git branch -d del
error: the branch 'del' is not fully merged
hint: If you are sure you want to delete it, run 'git branch -D del'
hint: Disable this message with "git config set advice.forceDeleteBranch false"

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$ git branch -D del
Deleted branch del (was da73fba).

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$ git branch
  main
* work

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$ git checkout del
error: pathspec 'del' did not match any file(s) known to git

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (work)
$
```

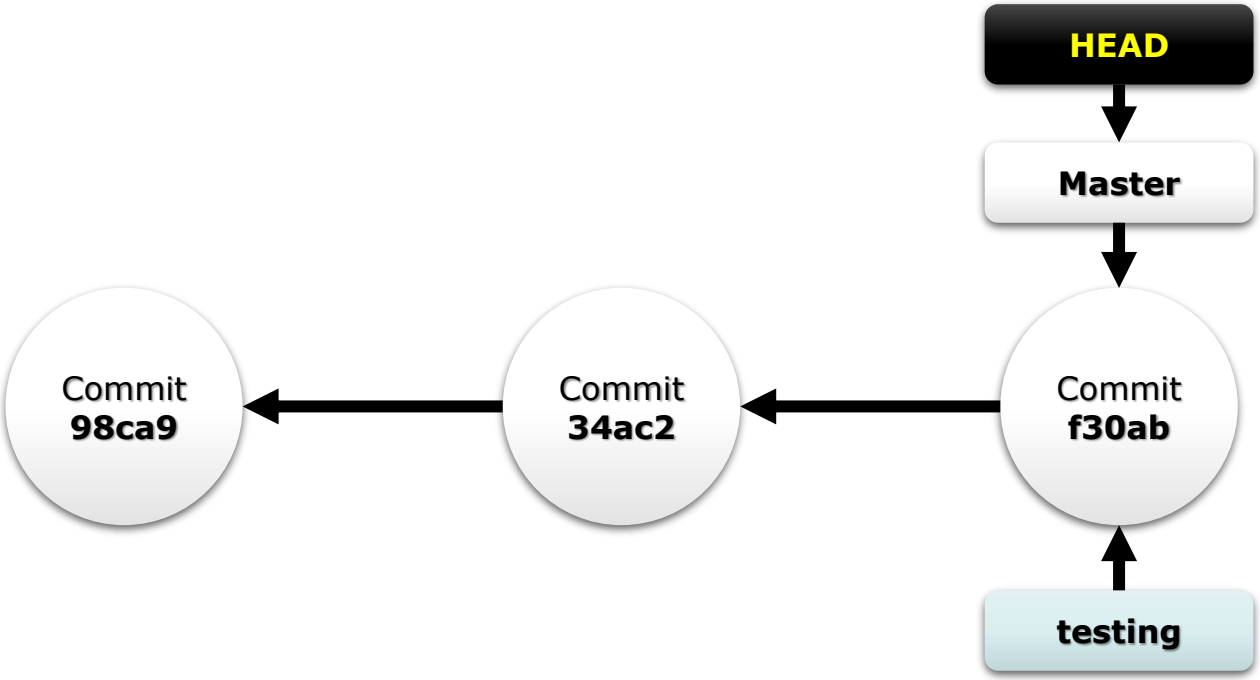
브렌치 기초 정리 - 1

- **브렌치**, Commit**과** HEAD
  - **저장소의 기본 브렌치 : master**(git) **또는 main**(github)
  - HEAD**는 항상 현재 작업 중인 브렌치의** Commit**을 가리키고 있음**
  - **현재 저장소의 상태**
  - 3**개의** Commit**이 존재**
  - **현재** 3**번째** Commit**까지 작업이 진행된 상태**
  - **브렌치는** master 1**개만 존재**



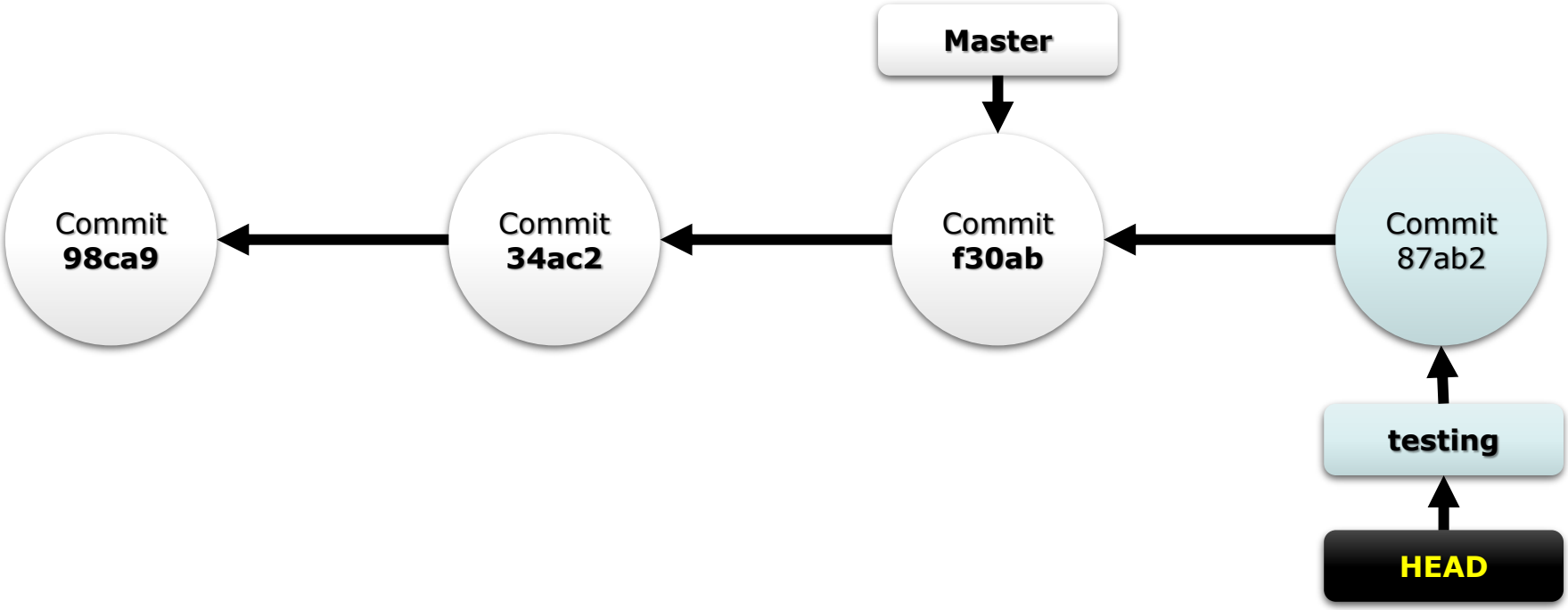
브렌치 기초 정리 - 2

- **브렌치의 생성과 이동**
  - **저장소의 브렌치 생성: git branch "브렌치이름"**
  - **현재 상태에서 논리적으로 파일들의 개별 상태를 생성**
  - "git branch testing" **명령을 수행하면** "testing" **브렌치가 생성됨**
  - "**git checkout** testing" **명령으로 해당 브렌치로 이동**
  - "git branch –b testing" **명령으로 한번에 수행 가능**



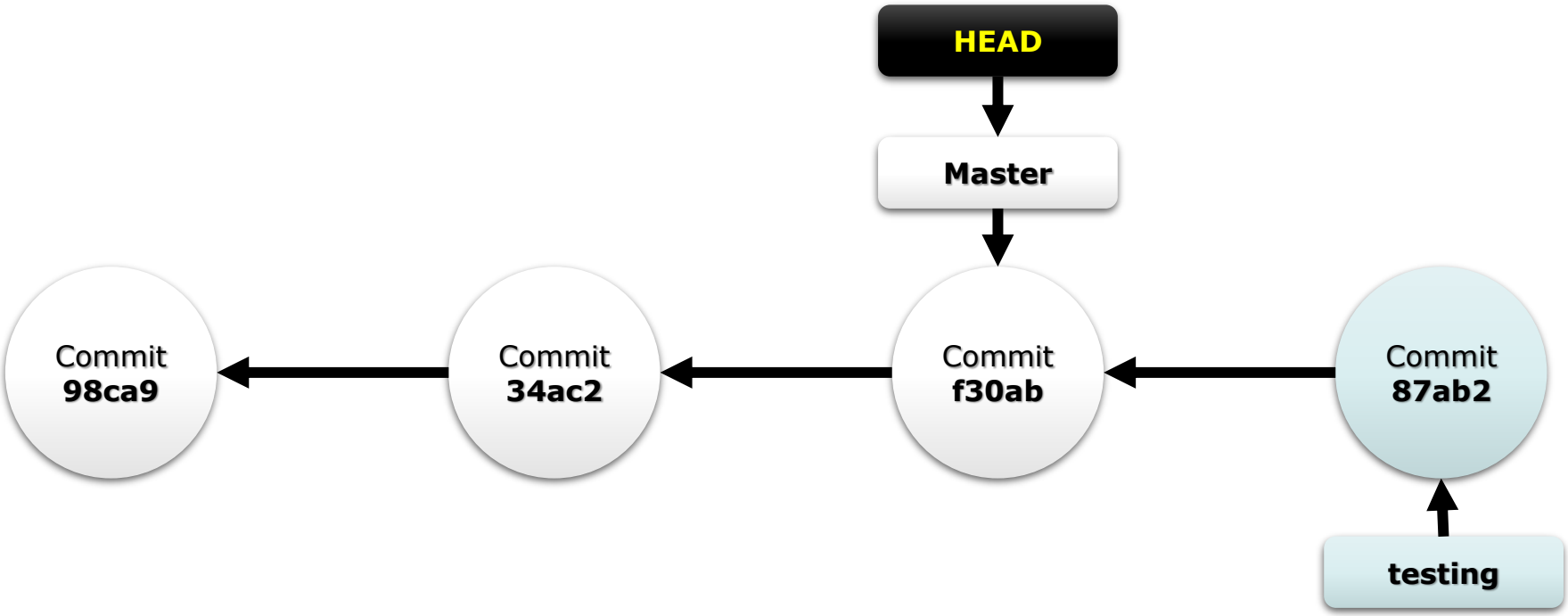
브렌치 기초 정리 - 3

- **생성한 브렌치에서** Commit
  - **생성된 브렌치로 이동하여 파일 수정이나 추가 등의 작업 후** Commit
  - **해당 브렌치에** Commit**된 내용이 추가로 저장**
  - master**는 별도** Commit**이 추가된 상태가 아니므로 이전 위치에 존재**
  - **새로운** Commit**은** master **브렌치에 적용되지 않은 상태**



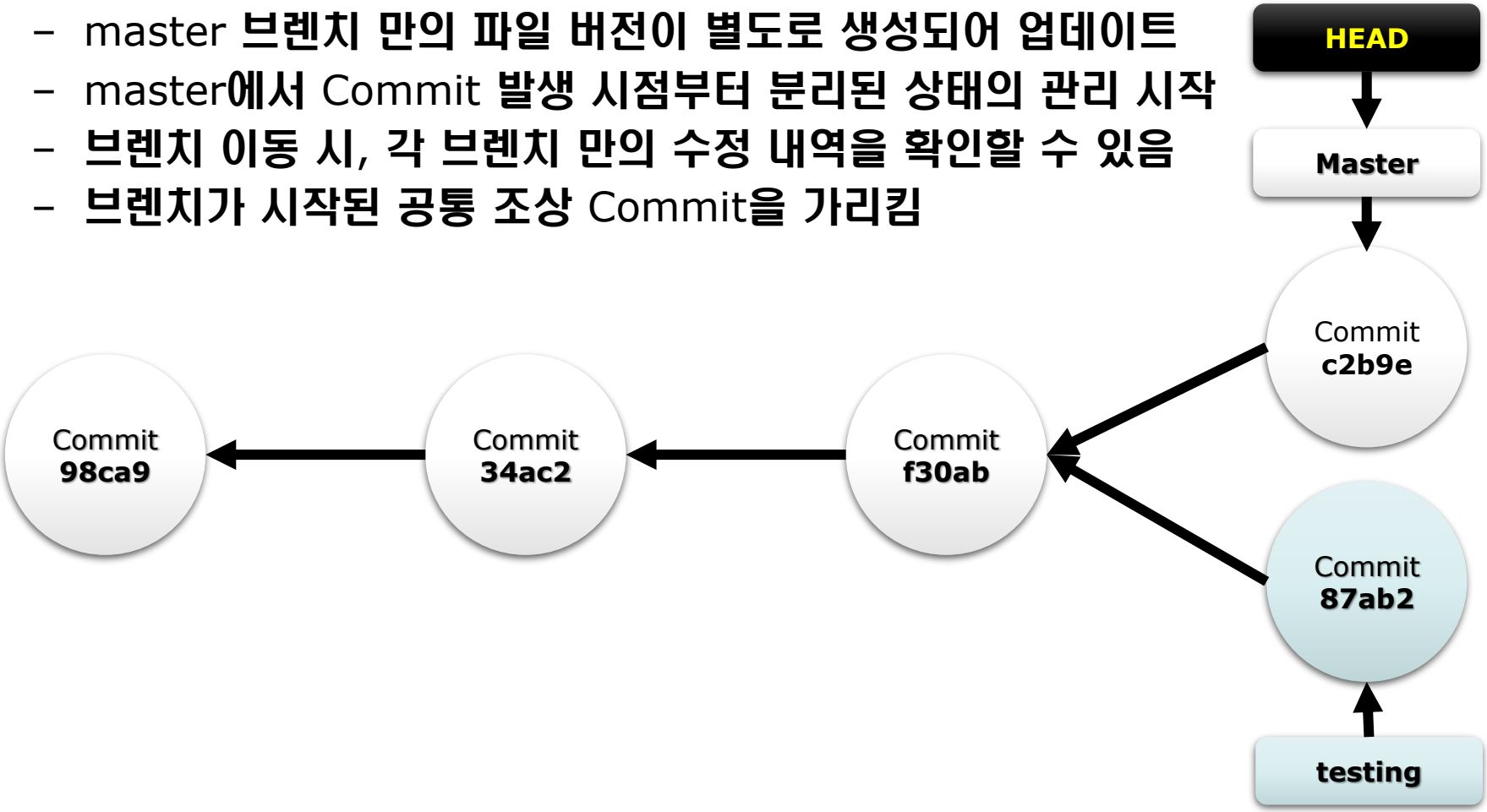
브렌치 기초 정리 - 4

- master **브렌치로 이동**
  - "git checkout master"**로** master **브렌치로 이동**
  - "**git branch**" **명령은 현재 로컬의 브렌치 목록을 출력**
  - **브렌치 이동은 단순히** HEAD**의 이동이므로 큰 변화는 없음**



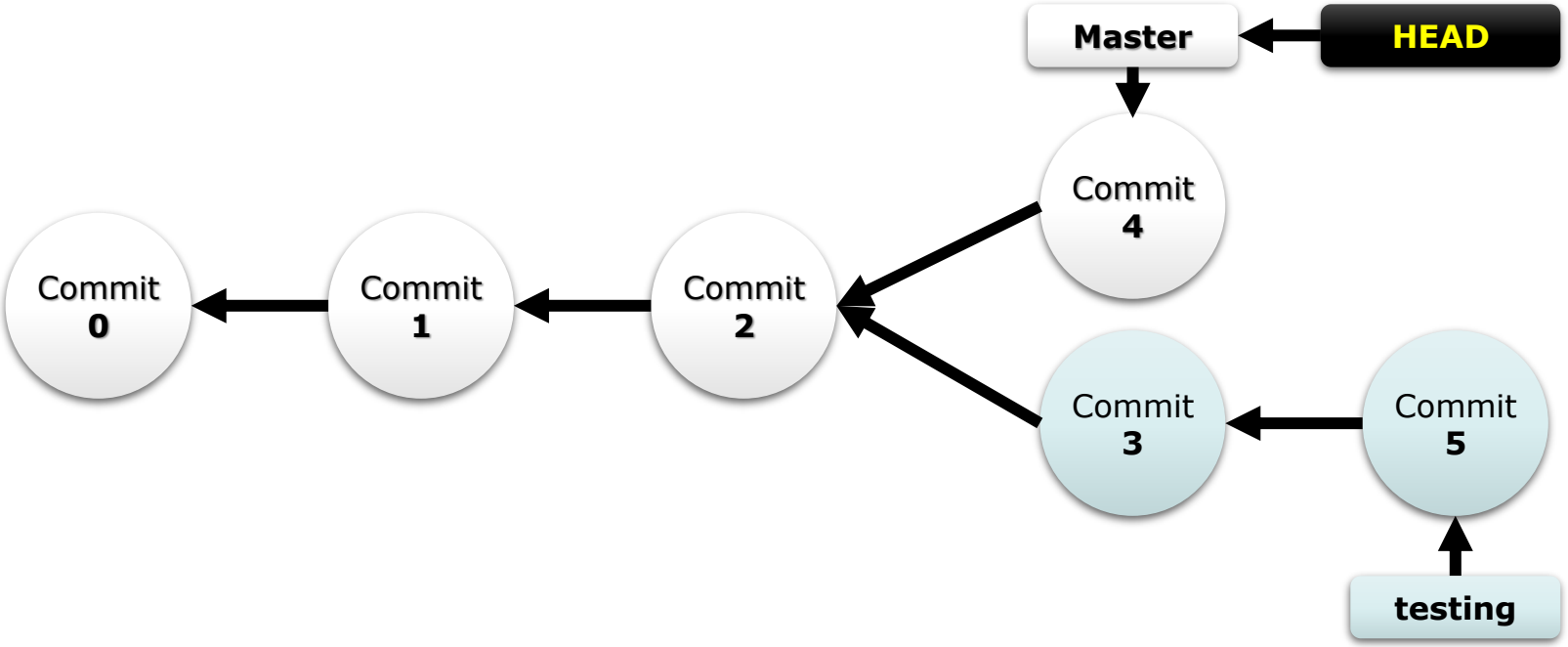
브렌치 기초 정리 - 5

- master **브렌치의** Commit
  - **이동한** master **브렌치에서 파일을 수정, 추가 후** Commit
  - master **브렌치 만의 파일 버전이 별도로 생성되어 업데이트**
  - master**에서** Commit **발생 시점부터 분리된 상태의 관리 시작**
  - **브렌치 이동 시, 각 브렌치 만의 수정 내역을 확인할 수 있음**
  - **브렌치가 시작된 공통 조상** Commit**을 가리킴**



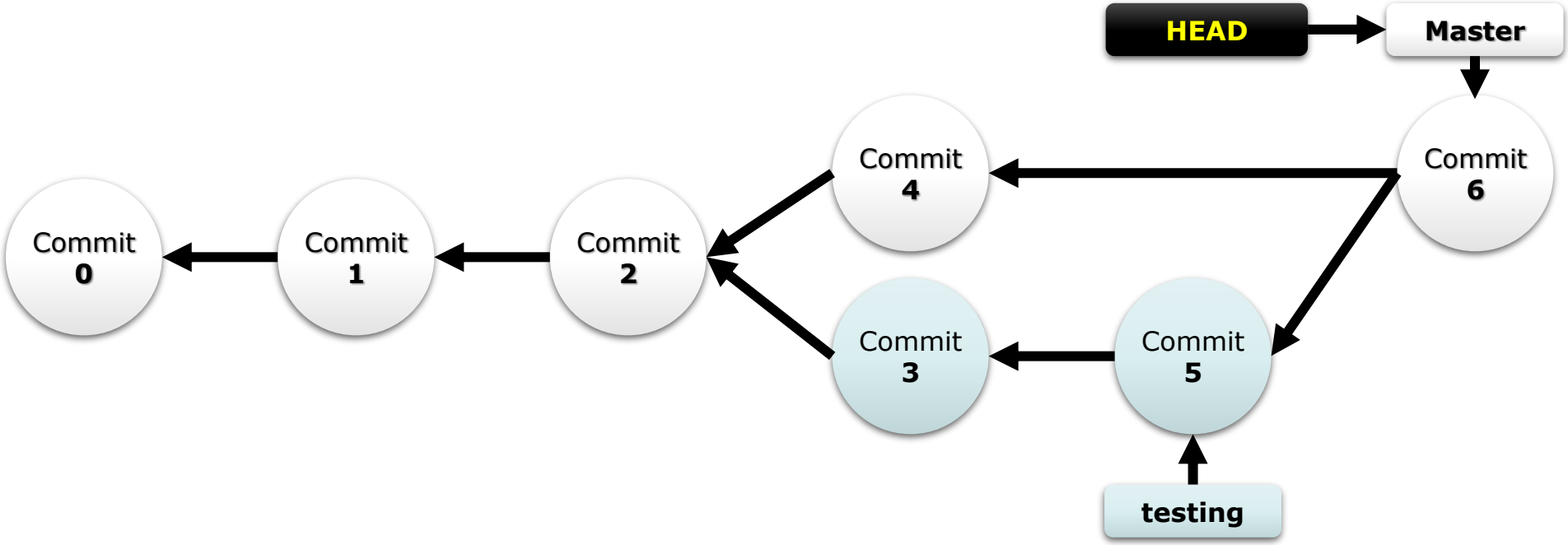
브렌치 기초 정리 - 6

- **브렌치 병합** – 1
  - 브렌치를 생성한 목표를 달성한 경우 병합을 수행
  - 병합은 2개의 다른 파일 흐름을 합치는 과정
  - "**git merge** 병합할브렌치명" 명령으로 병합
  - 현재 브렌치에 다른 브렌치를 합치는 형태로 동작
  - master에 testing **브렌치를 병합하려면** master **브렌치에서 수행**해야 함



브렌치 기초 정리 - 7

- **브렌치 병합** – 2
  - "git checkout master"**로** master **브렌치로 이동**
  - "git merge testing"**으로 브렌치를 병합**
  - **병합되면** testing **브렌치 삭제가 가능해짐**
  - **Commit 6 스냅샷에는** testing**에서 수행된 모든 작업이 포함**



branch 병합

About..

컴퓨터소프트웨어공학과
김 원 일

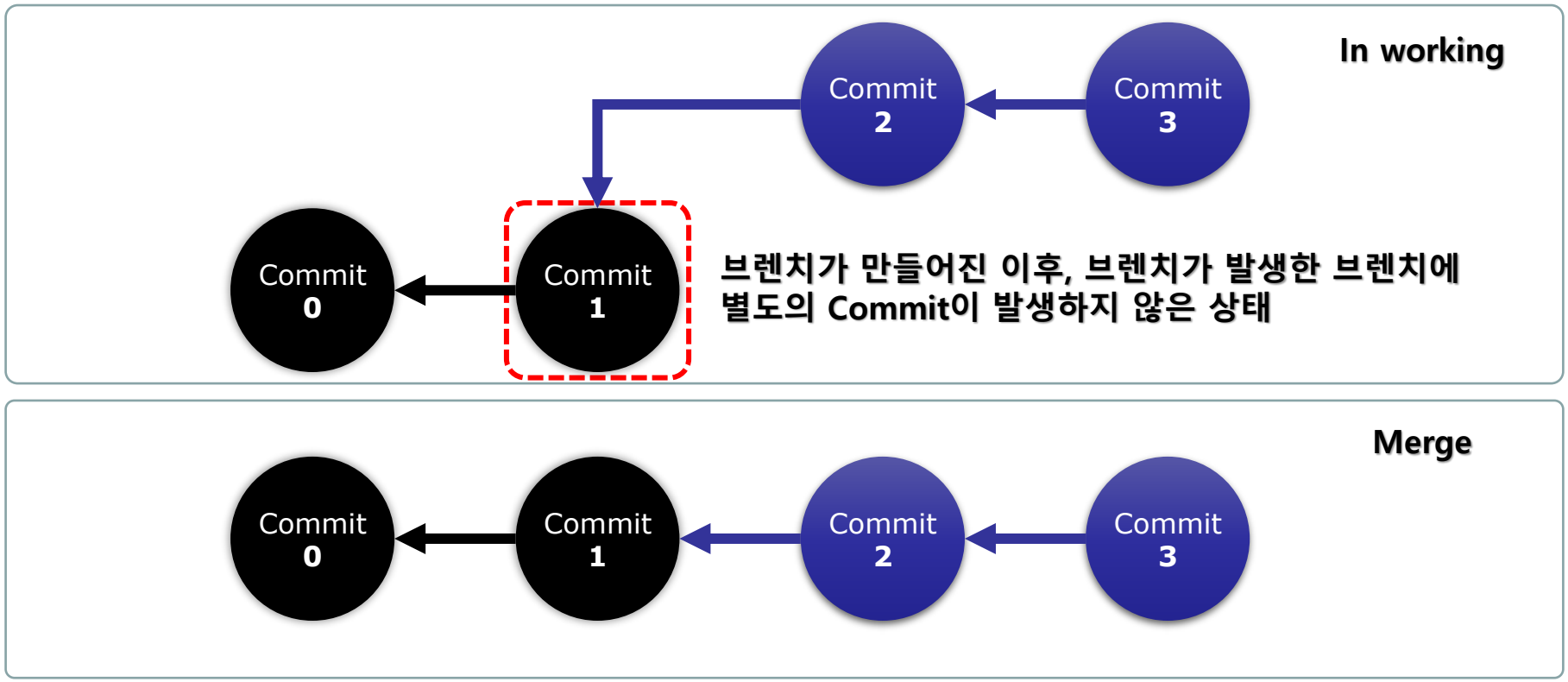
브렌치 병합 방법

- **병합 방법**
  - git에서 제공되는 기본적인 병합 방식은 여러 가지가 존재함
  - 프로젝트 진행이나 설계 또는 팀장의 의도에 따라 선택하여 결정
  - 팀장과 회사에 따라 다른 형식의 병합 방식을 이용할 수 있음
  - 개인적으로 다양한 형식의 병합 방식을 연습하는 것이 중요

  - fast-forward merge
    - 일반적으로 사용할 수 있는 방식이나 의도 없이 이루어질 수 있음
    - 다른 병합 방법에서도 자동으로 적용되는 경우가 있음
  - 3-way merge
    - 가장 일반적인 병합 방식으로 브렌치를 쉽게 병합할 수 있는 방법
  - squash merge
    - 병합할 브렌치를 압축하여 하나의 커밋으로 만들어 병합하는 방식
    - 파일 이력을 압축하지만 코드 관리 면에서는 깔끔하게 코드를 볼 수 있음
  - rebase
    - 충돌 등으로 문제 발생을 대비하여 병합을 수행할 수 있는 방식
    - base 정보를 변경하여 문제가 발생할 수 있는 부분을 정리할 수 있음

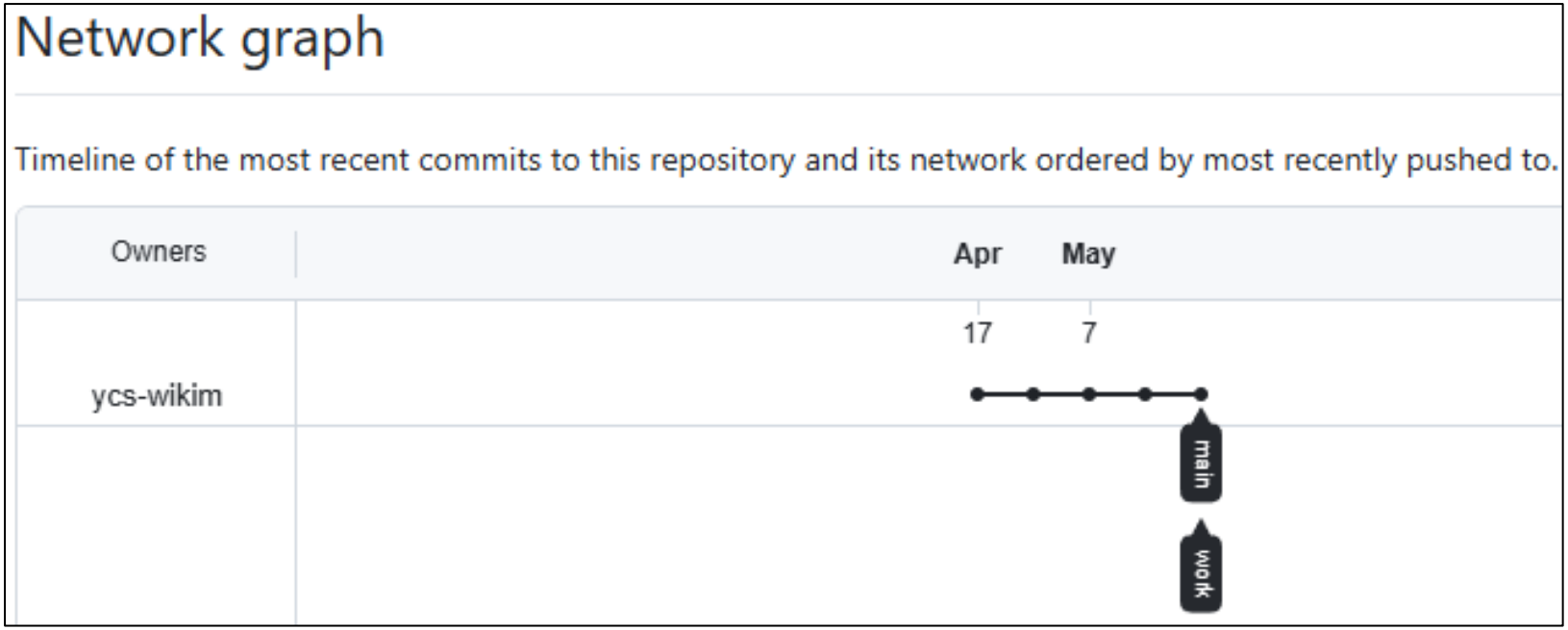
fast-forward merge - 1

- **병합 방식**
  - Commit 1**에서 브렌치 생성 후 원래 브렌치에서 별도 작업이 없는 상태**
  - **병합 시, 원래 브렌치에서 순서대로 작업한 것과 동일한 경우의 병합**
  - **교재 첫 번째 병합에서** main, work **브렌치가 이 방식을 이용하여 병합된 것**



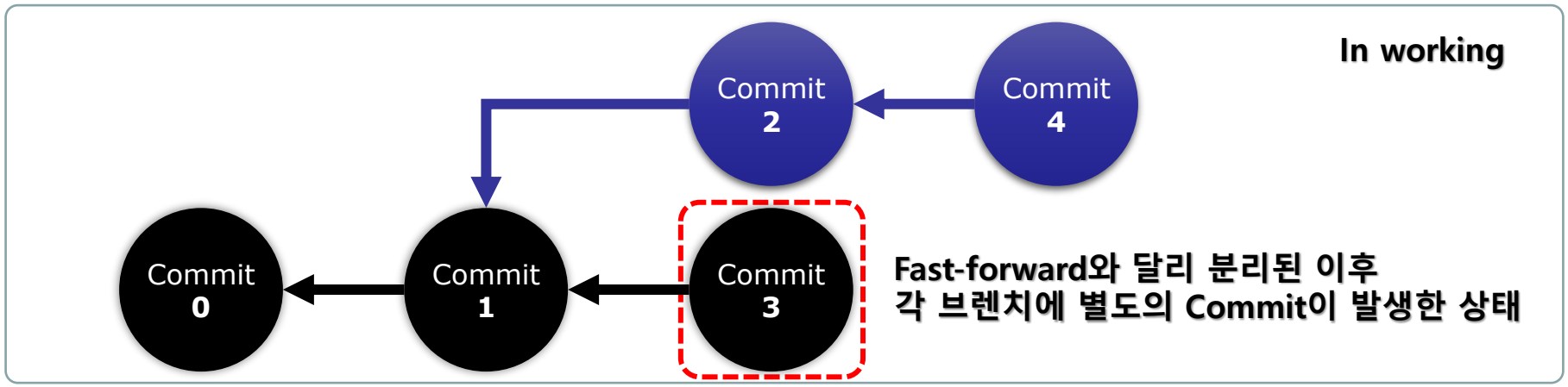
fast-forward merge - 2

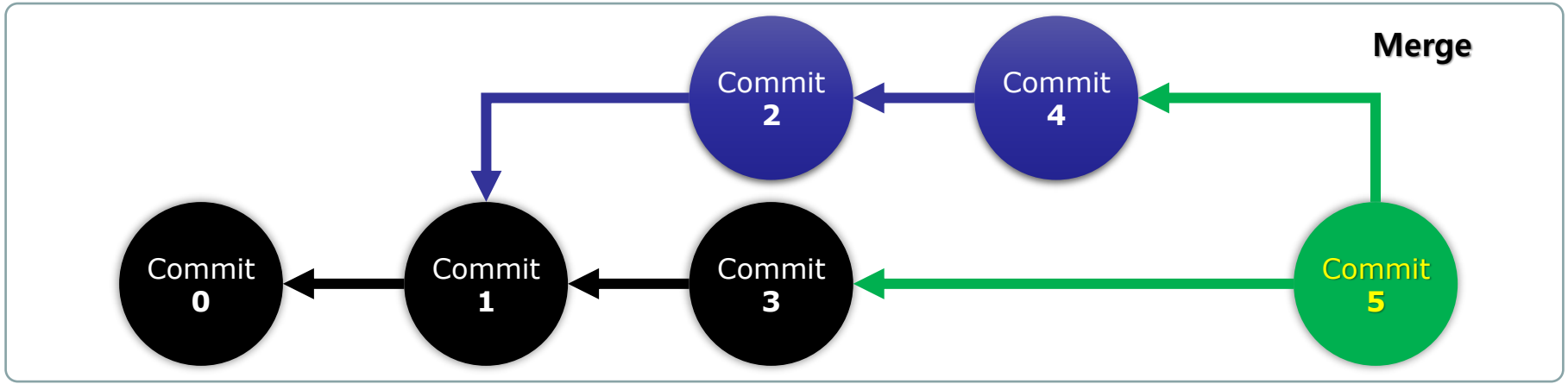
- **병합 예제**
  - **브렌치 기초** - 18 **예제에서 확인한 것과 같이 브렌치가 모두 같은 위치**
  - main **브렌치에 별도의** Commit**이 발생하지 않아 하나로 병합**
  - work **브렌치에서** Commit **이 발생하면** main **브렌치 상태에 따라 분리 됨**
  - main **브렌치에 변화가 없는 상태가 지속되면 지속적으로 같은 상태가 됨**



3-way merge - 1

- **병합 방식**
  - **브렌치를 병합하는 가장 일반적인 형태로 많이 볼 수 있는 병합 형태**
  - Commit 5는 3의 다음 버전으로 4의 내용이 합쳐진 형태로 만들어짐





3-way merge - 2

- **병합 예제** – 1
  - main**과 새로운 브렌치에서 각각** Commit**을 수행한 후 병합하여 결과 확인**
  - 3way **브렌치를 생성하고**, main **브렌치에 새로운 파일을** Commit
  - 3way **브렌치로 이동하여 새로운 파일을 추가 또는 변경 후** Commit **수행**

```
MINGW64:/d/202507001/testnew
YUHAN@YP12624115 MINGW64 /d/202507001/testnew (main)
$ git branch
* main
  work

YUHAN@YP12624115 MINGW64 /d/202507001/testnew (main)
$ git branch 3way

YUHAN@YP12624115 MINGW64 /d/202507001/testnew (main)
$ vi main.txt

YUHAN@YP12624115 MINGW64 /d/202507001/testnew (main)
$ git add main.txt ; git commit -m "add main.txt"
warning: in the working copy of 'main.txt', LF will be replaced by CRLF the next ti
me Git touches it
[main 6effe3c] add main.txt
 1 file changed, 1 insertion(+)
 create mode 100644 main.txt

YUHAN@YP12624115 MINGW64 /d/202507001/testnew (main)
$ git checkout 3way
Switched to branch '3way'

YUHAN@YP12624115 MINGW64 /d/202507001/testnew (3way)
$
```

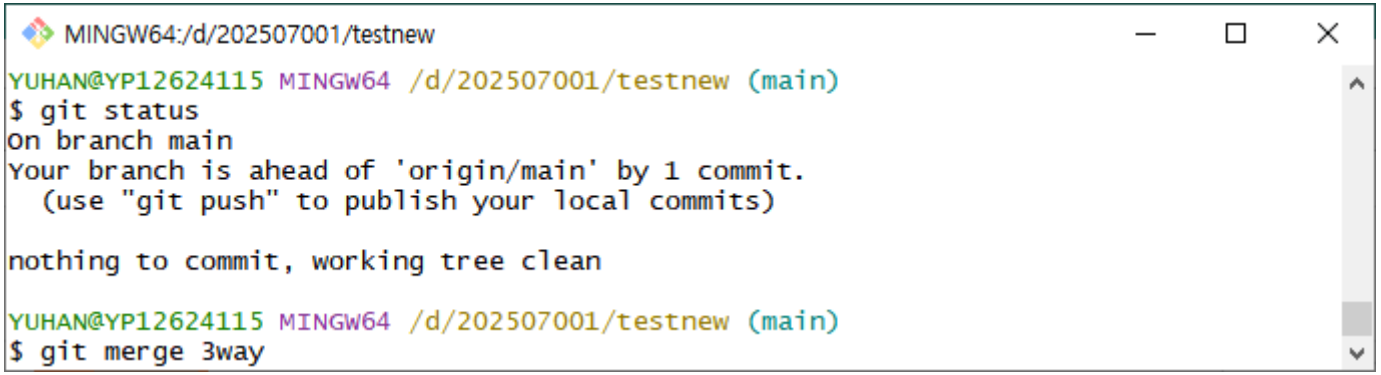
3-way merge - 3

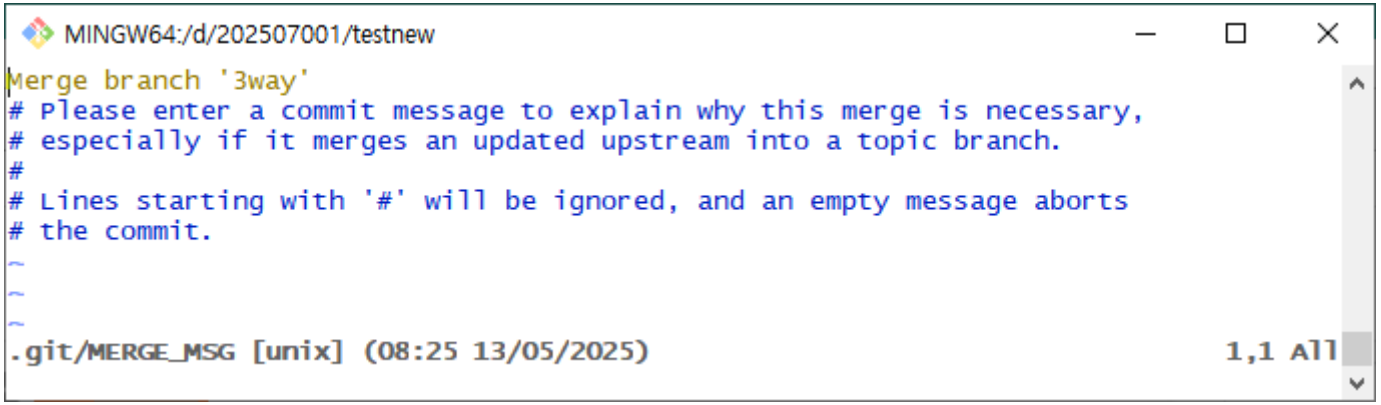
- **병합 예제** – 2
  - 3way **브렌치에서 파일을 추가하고**, Commit**을 수행**
  - main**과** 3way **브렌치에서 각각** Commit**을 수행한 상태**
  - main **브렌치로 돌아와서 병합 준비**

```
MINGW64:/d/202507001/testnew
YUHAN@YP12624115 MINGW64 /d/202507001/testnew (3way)
$ git status
On branch 3way
nothing to commit, working tree clean

YUHAN@YP12624115 MINGW64 /d/202507001/testnew (3way)
$ vi 3way.txt

YUHAN@YP12624115 MINGW64 /d/202507001/testnew (3way)
$ git add 3way.txt ; git commit -m "add 3way.txt"
warning: in the working copy of '3way.txt', LF will be replaced by CRLF the next ti
me Git touches it
[3way 95006e0] add 3way.txt
 1 file changed, 2 insertions(+)
 create mode 100644 3way.txt

YUHAN@YP12624115 MINGW64 /d/202507001/testnew (3way)
$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

YUHAN@YP12624115 MINGW64 /d/202507001/testnew (main)
$
```

3-way merge - 4

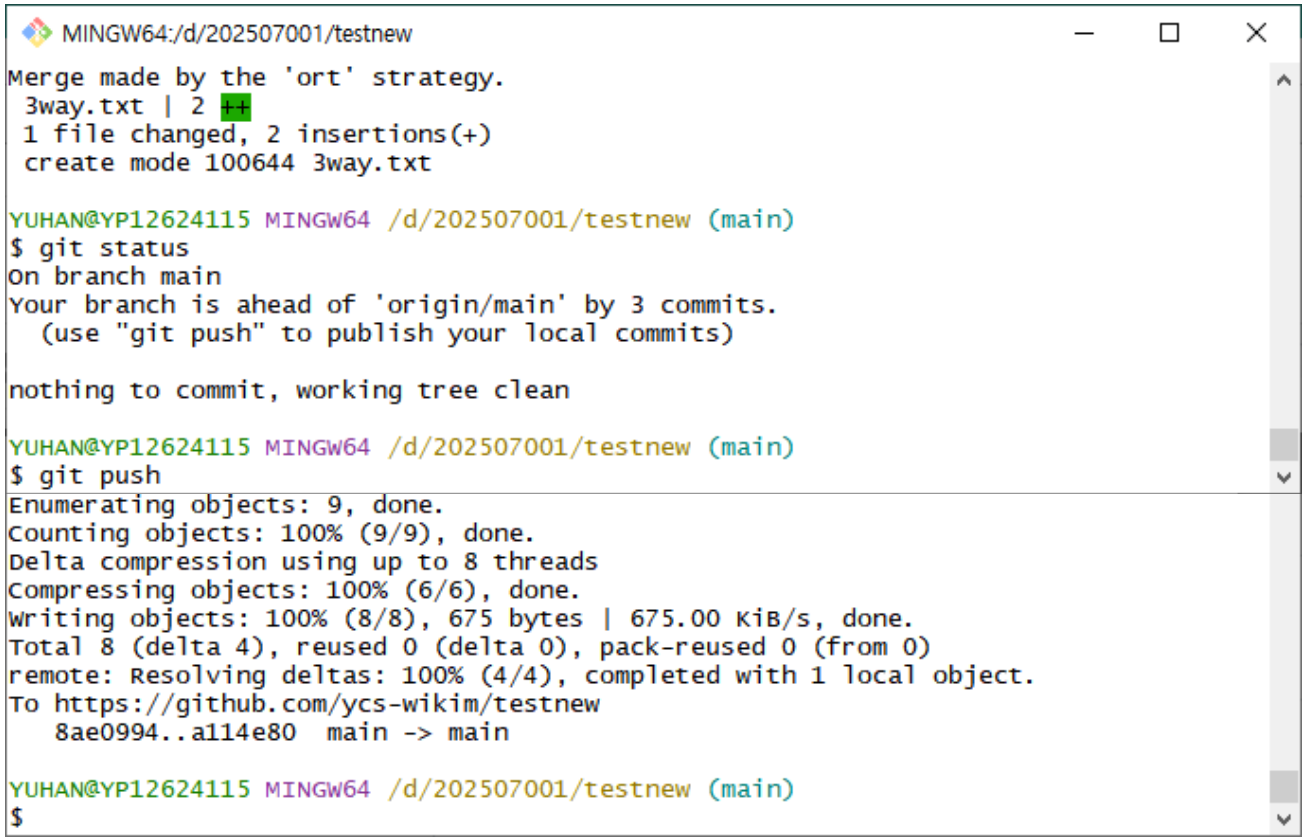
- **병합 예제** – 3
  - 3way **브렌치를** main **브렌치에 병합하는 명령어를 수행**
  - **브렌치 기초에서의 병합과는 다르게** Commit**이 자동으로 수행**
  - Commit **메시지에** 3way **브렌치를 병합함을 출력하고 있음**
    - Commit **메시지가 자동 입력되어 있으므로 저장하고 종료하면 자동으로** Commit **수행**





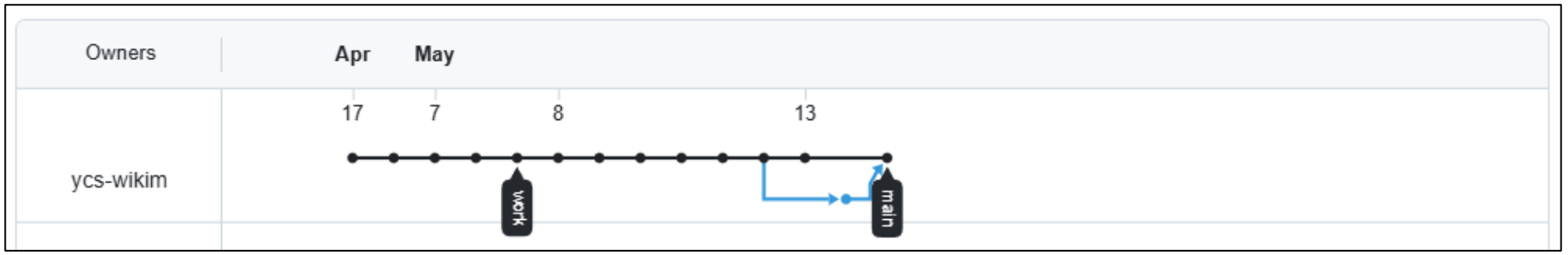
3-way merge - 5

- **병합 예제** – 4
  - ‘ort’ **병합 전략** : Ostensibly Recursive Three-way merge
    - **표면적으로 재귀적이라는 의미로 기존의** 3-way **병합이 향상된 형태의 병합 방법으로 이해**
  - **결과 확인을 위해** 3**개의** Commit**을 원격 저장소로 전달**
    - main 1**개**, 3way 1**개**, **병합에서 발생한** 1**개로 총** 3**개의** Commit**이 발생**



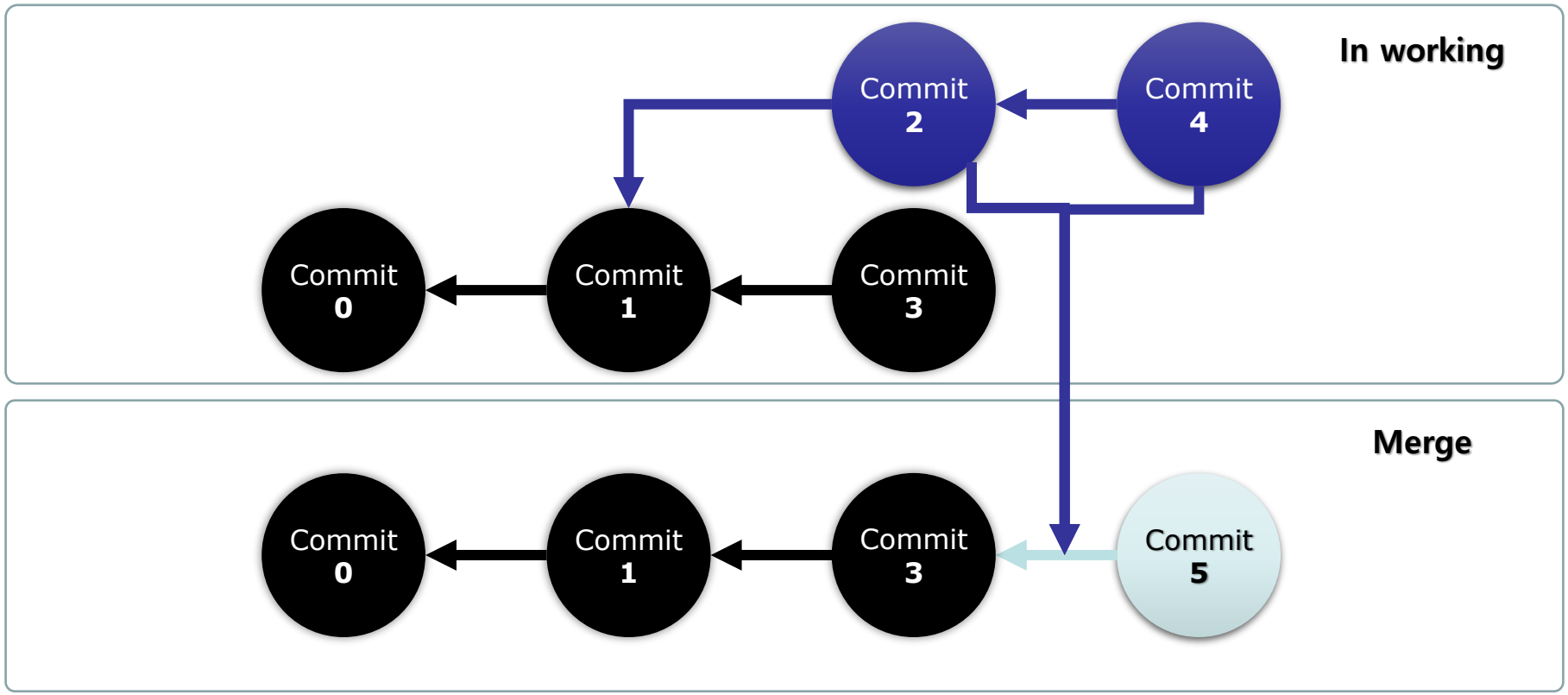
3-way merge - 6

- **병합 예제** – 5
  - **원격 저장소에 정상적으로 업로드되면** github**로 접속**
  - Insights → Network **메뉴를 통해 브렌치의** Commit **상태 확인**
  - **기존의** fast-forward **방식과 다르게 색상이 있는 브렌치 정보가 출력**
  - 3way **브렌치는 원격 저장소에 업로드하지 않아 출력되지는 않음**
  - **어느 시점에 어떤 작업이 진행되었는지를 확인할 수 있음**



squash merge - 1

- **병합 방식**
  - **병합하려는 브렌치의 모든 Commit의 수정 내용을 합쳐서 병합하는 방식**
  - **브렌치 Commit 2, 4의 수정 내용을 main 브렌치를 직접 수정한 상태로 병합**
  - **병합한 다음 개발자가 별도로 Commit을 수행하여 병합 완료를 수행해야 함**
  - **변경된 코드만 브렌치에 적용되며 강제로 Fast-Forward 병합으로 처리**



squash merge - 2

- **테스트 환경**
  - **브렌치를 생성하고**, Commit **메시지를 인식할 수 있도록** 2**개의** Commit **수행**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git checkout -b squash
Switched to a new branch 'squash'

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (squash)
$ vi squash.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (squash)
$ git add squash.txt ; git commit -m "commit-1 for test merge"
warning: in the working copy of 'squash.txt', LF will be replaced by CRLF the ne
xt time Git touches it
[squash 07df77e] commit-1 for test merge
 1 file changed, 2 insertions(+)
 create mode 100644 squash.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (squash)
$ vi squash.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (squash)
$ git add squash.txt ; git commit -m "commit-2 for test merge"
warning: in the working copy of 'squash.txt', LF will be replaced by CRLF the ne
xt time Git touches it
[squash 6114dc7] commit-2 for test merge
 1 file changed, 3 insertions(+)

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (squash)
$
```

squash merge - 3

- squash **병합**
  - main **브렌치에서** squash **옵션으로 병합을 수행하고, 저장소 정보 확인**
    - **병합한 브렌치에서 작업한 내용이** main **브렌치에** Staged **상태로 병합된 것을 확인할 수 있음**

```
MINGW64:/f/202507001/testnew

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (squash)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git merge --squash squash
Updating a114e80..6114dc7
Fast-forward
Squash commit -- not updating HEAD
 squash.txt | 5 +++++
 1 file changed, 5 insertions(+)
 create mode 100644 squash.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   squash.txt


unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$
```

squash merge - 4

- **병합 후** Commit
  - **다른 병합처럼** Commit**까지 진행되는 무조건 병합이 아님**
  - **병합되는 브렌치 정보를 병합하려는 브렌치에 준비 상태로 병합**
  - Commit **전에 수정할 내용이 있다면 미리 처리할 수 있는 상태**
  - **브렌치의 여러** Commit**이 한번에 적용되므로 추가 확인을 위한 조치**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   squash.txt


unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git commit -m "squash merge commit"
[main aafc176] squash merge commit
 1 file changed, 5 insertions(+)
 create mode 100644 squash.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$
```

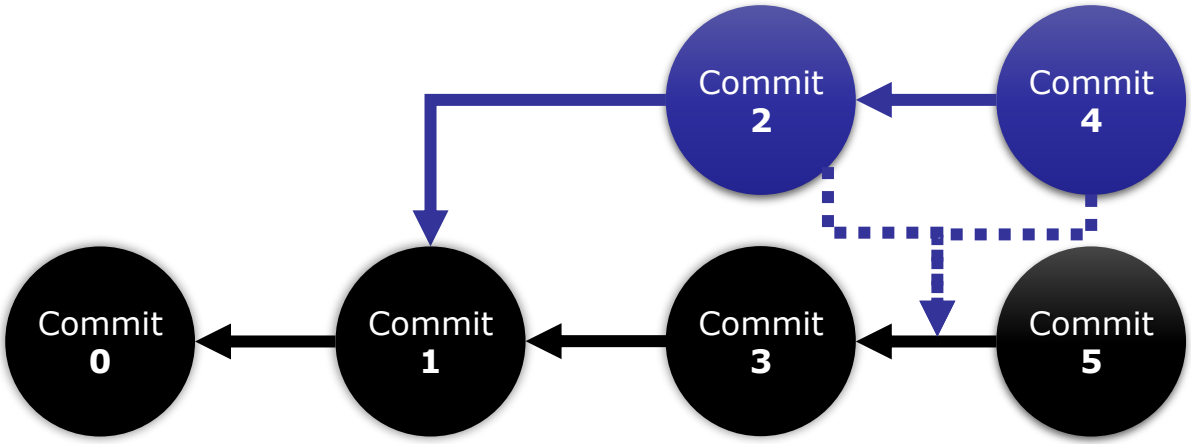
squash merge - 5

- **병합 상태**
  - Staged **상태로 병합되었기 때문에 취소도 당연히 가능한 상태**
  - **병합과 동시에** Commit**이 아니기 때문에 가능**
  - **즉, 병합 후 코드를 처리할 수 있으며 병합 자체를 취소할 수도 있음**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   squash.txt


unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git restore --staged squash.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

nothing added to commit but untracked files present (use "git add" to track)

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$
```

squash merge - 6

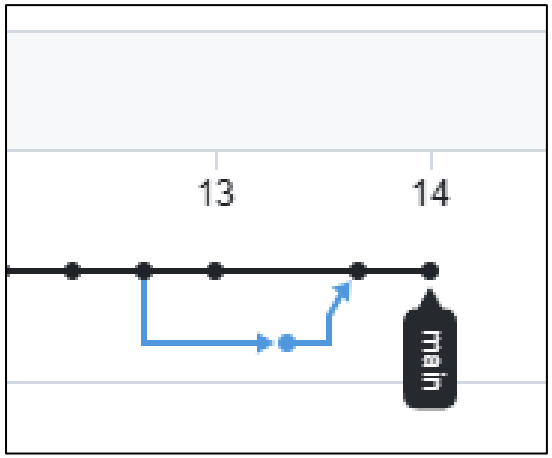
- **활용 방법**
  - **병합할 브렌치에서 작성한 내용 전체가 필요 없는 경우에 사용**
  - **작업한 브렌치의 중간 단계 Commit이 필요/의미 없는 경우**
  - **작업을 수행한 브렌치 작업을 깔끔하게 정리해서 한번에 병합할 수 있음**
  - **병합이 Fast-Forward 방식으로 수행되므로 연결 관계가 표시되지 않음**
    - **연결 관계가 표시되지 않기 때문에 브렌치가 작업 완료되지 않은 것으로 인식될 수 있음**
  - **병합 후 브렌치 유지가 필요 없으면 브렌치를 삭제해도 문제 없음**



squash merge - 7

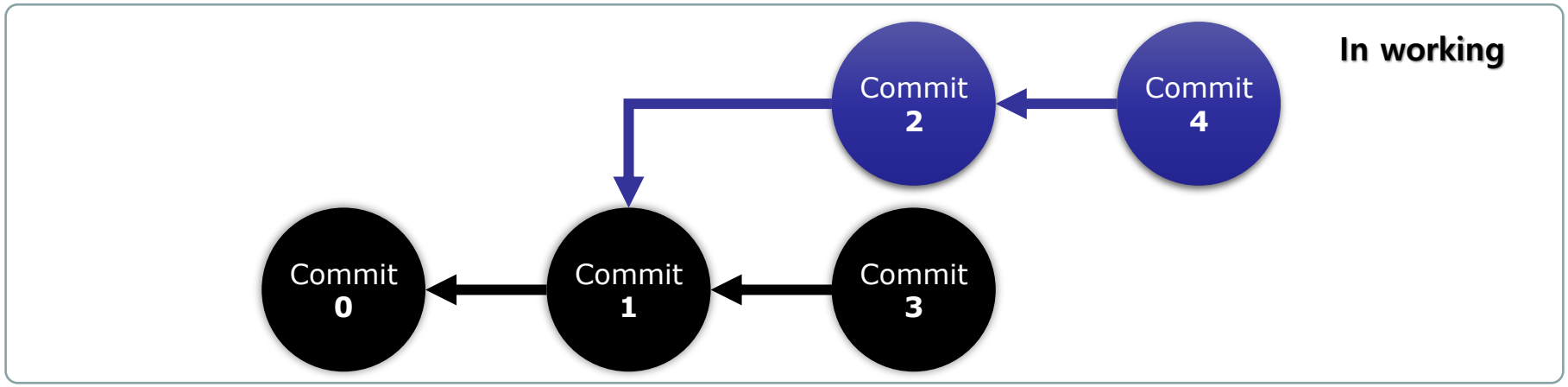
- **병합 결과**
  - squash **병합 후** Commit**하여 원격 저장소로 업로드**
  - Fast-Forward **방식으로 병합되어 바로 다음** Commit**으로만 인식**
  - **무제한으로 생성되는 브렌치를 관리할 수 있는 방식 중 하나로 사용 가능**
  - **소스 코드 이력과 관리를 중간 단계를 최소화하며 스마트하게 수행**
  - **브렌치에서 수행한 작업에 대한 정보를 병합** Commit**에 기록하는 것이 좋음**
  - **병합되는** Commit**에서 소스 코드만을 적용하므로 메시지 작성에 주의**
    - **버전 관리에 숙련된 상태에서는 활용할 수 있는 방법이 많이 존재**

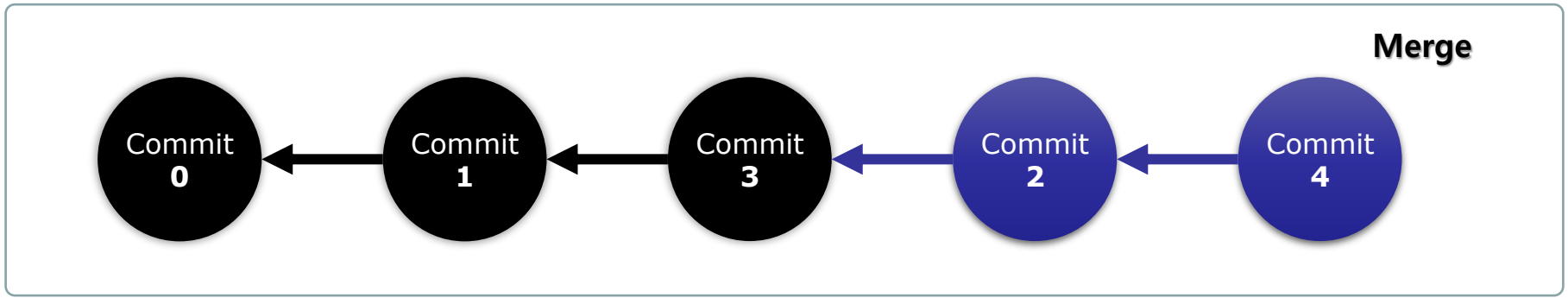
```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git commit -m "squash merge"
[main 7e1e1d8] squash merge
 1 file changed, 5 insertions(+)
 create mode 100644 squash.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 306 bytes | 306.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ycs-wikim/testnew
   a114e80..7e1e1d8  main -> main

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$
```



rebase merge - 1

- rebase **병합**
  - base(**브렌치 생성 부모 브렌치**)**의** Commit**을 재 지정한다는 의미**
  - Commit 2, 4**는** Commit 1**을** base**로 브렌치가 생성되어 작업한 이력**
  - Commit 2**의** base**를 병합하려는 브렌치의 최신** Commit**으로 변경하는 병합**
  - **병합 결과는** Fast-Forward**로 병합한 것과 동일한 형태**





rebase merge - 2

- **병합 예제** - 1
  - rebase **브렌치를 생성하고**, main **브렌치에서 파일 수정 및** Commit
  - **파일은** git**이 관리하고 있는 파일로 새로운 파일이 아님**
  - **파일 내부에** main **브렌치에서 수정하였음을 입력하고** Commit
  - Commit **메시지에** main **브렌치에서** Commit**하였음을 입력**
  - rebase **브렌치로 브렌치를 이동**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git branch rebase

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ vi main.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git add main.txt ; git commit -m "main : modify main.txt"
[main 6b714e7] main : modify main.txt
 1 file changed, 4 insertions(+)

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git switch rebase
Switched to branch 'rebase'

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (rebase)
$
```

rebase merge - 3

- **병합 예제** - 2
  - **신규 파일인** rebase.txt **파일을 생성하고** rebase**에서 작성한 것임을 입력**
  - Commit **시에** rebase **브렌치에서 수행한 것임을 표시하도록 메시지 입력**
  - **브렌치를 변경하지 않은 상태에서** main **브렌치에 대해** rebase**를 수행**
  - **정상 수행 시** rebase**의** base**가** main**의 마지막(최신)** Commit**으로 변경**

```
MINGW64:/f/202507001/testnew
unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (rebase)
$ vi rebase.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (rebase)
$ git add rebase.txt ; git commit -m "rebase : add rebase.txt"
warning: in the working copy of 'rebase.txt', LF will be replaced by CRLF the ne
xt time Git touches it
[rebase 0a221b4] rebase : add rebase.txt
 1 file changed, 2 insertions(+)
 create mode 100644 rebase.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (rebase)
$ git rebase main
Successfully rebased and updated refs/heads/rebase.

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (rebase)
$
```

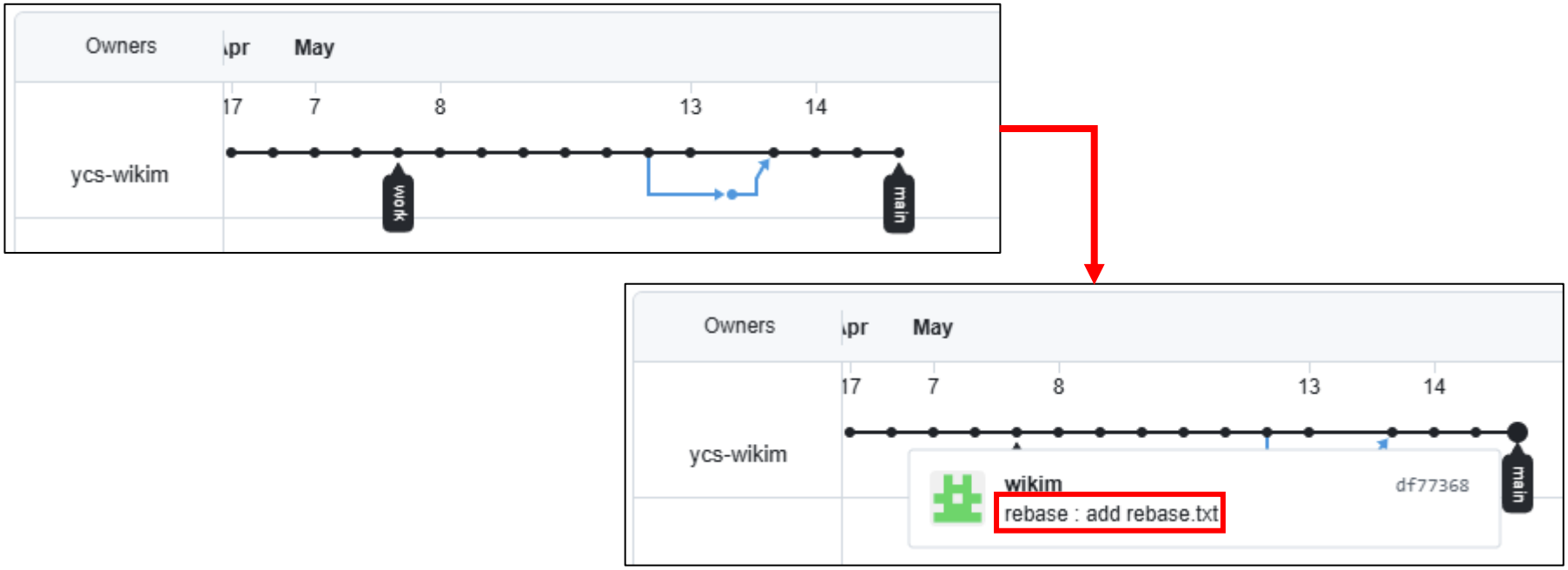
rebase merge - 4

- **병합 결과**
  - rebase **브렌치의 작업 내용이** main **브렌치에 문제 없이 병합됨**
  - **병합될 때** Fast-forward **방식으로 병합되는 것을 확인할 수 있음**
  - main **브렌치에는 파일을 수정한 다음에** rebase.txt **파일이 추가된 형태**

```
MINGW64:/f/202507001/testnew

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (rebase)
$ git switch main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$ git merge rebase
Updating 6b714e7..df77368
Fast-forward
 rebase.txt | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 rebase.txt

unangel@DESKTOP-I8OQG85 MINGW64 /f/202507001/testnew (main)
$
```

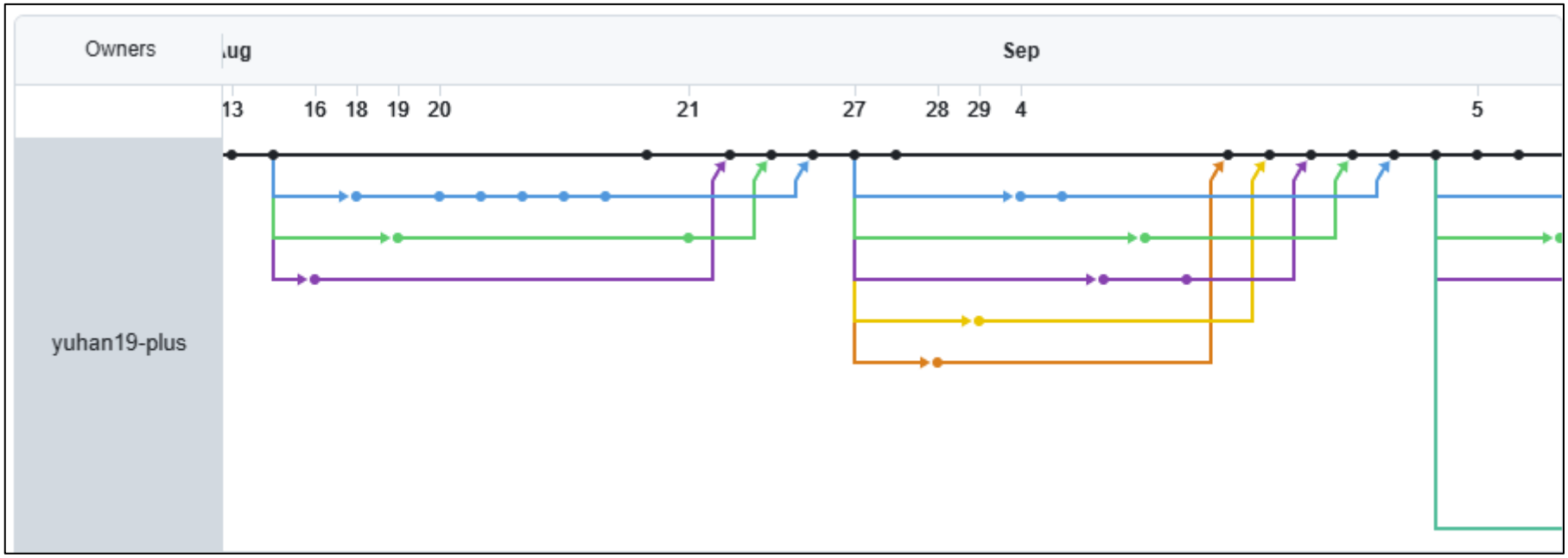
rebase merge - 5

- **병합의 효과**
  - squash **병합과 같이 브렌치에서 작업한 내용을 주요 브렌치에 병합이 목적**
  - squash**와 달리 작업 중간에 발생한** Commit**을 그대로 가져올 수 있음**
  - **중간 이력이 반드시 필요한 경우에 사용하는 병합 방법**
  - Fast-Forward**로 적용되기 때문에 코드 충돌 발생이 약간 감소 가능**
  - github**에서 최종** Commit**이** rebase **브렌치에서 수행한 것임을 확인 가능**



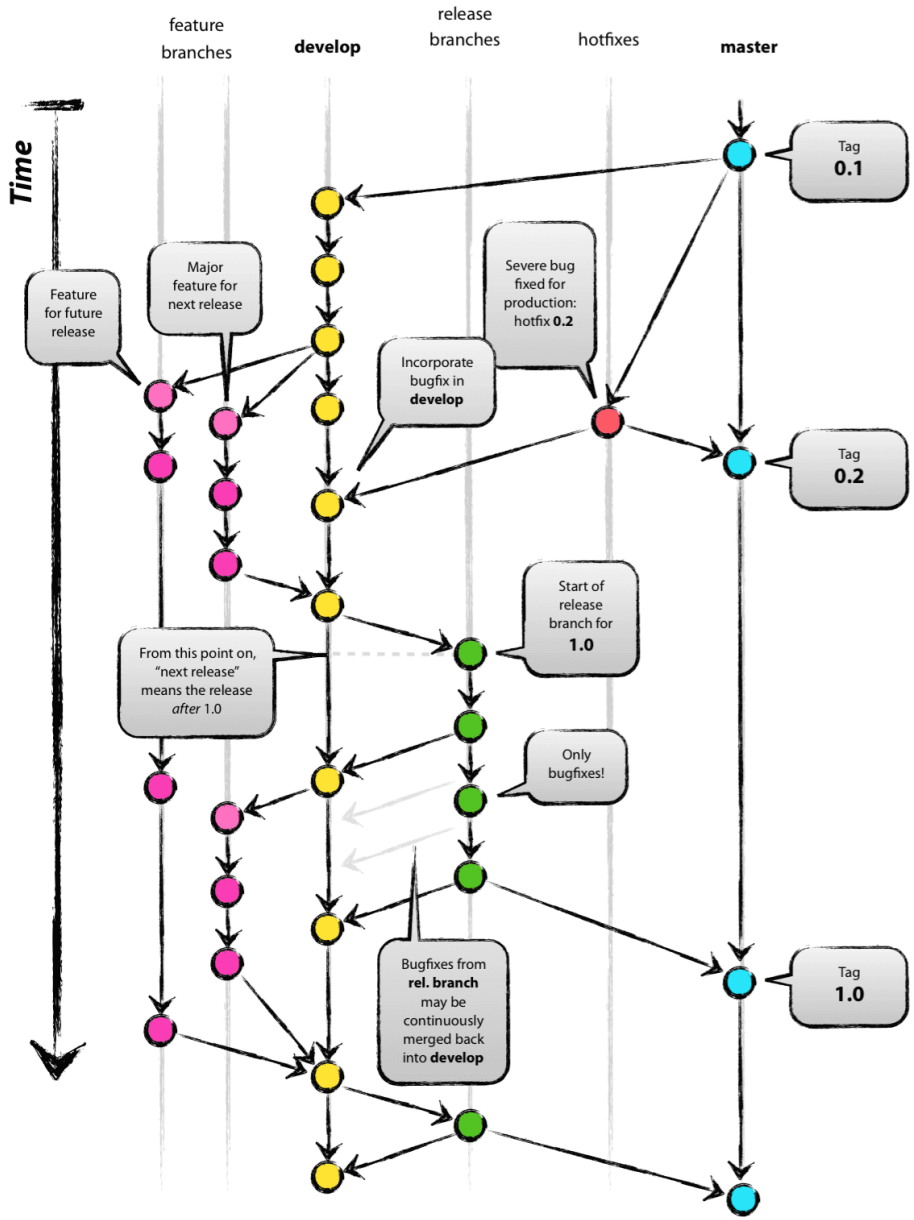
병합 선택

- **특성에 따라 적용**
  - **브렌치 병합 방법은 가지고 있는 특징이 서로 다르기 때문에 추가 학습 필요**
  - **회사 또는 팀마다 병합 방식이 서로 다르게 적용될 수 있음**
  - **모든 방식에 대한 이해와 충분한 실습과 연습을 수행하는 것이 좋음**
  - **개인 프로젝트부터 브렌치와 병합을 활용하는 것을 추천**
  - https://github.com/yuhan19-plus/yuhan-interactive-web/network



브렌치 전략 - 1

- gitflow



브렌치 전략 - 2

- **브렌치를 활용한 개발**
  - git flow **전략을 구성하는 방법에 따라 달라질 수 있음**
  - **버전 관리는 자유롭게 진행해야 하나,** git**은 자유도가 너무 높음**
  - **충돌을 최소한으로 처리하기 위한 협업 준비와 협의가 필요**
  - **프로그램 작성을 위한 설계와 소스 코드 구성 등을 미리 지정하고 수행**

Pull Request 할때, 하나의 클래스를 몇명이서 수정하게되면 겹치게될텐데 그런경우엔 어떻게 하시나요?

좋아요 · 답글 달기 · 3년

저희는 되도록 코드 충돌이 발생하지 않도록 작업을 나누어서 진행을 합니다.
그래서 하나의 클래스를 여러 명이서 건들지 않도록 하고 있습니다.
작업을 나눌 때 같은 코드를 여러 명이 건드려야 한다면 작업자들끼리 이야기를 한 후 한 명이 먼저 작업을 처리 합니다.
그렇게 하더라도 간혹 코드 충돌이 발생하게 되는데요. 대부분 이 사실은 오전 티타임을 할 때나 코드리뷰를 할 때 알게됩니다.
코드 충돌 해결은 전적으로 뒤에 코드 병합을 하는 사람이 책임을 지게됩니다.
코드 충돌이 작으면 스스로 처리하지만, 코드 충돌 범위가 크면 작업 코드가 겹친 개발자와 함께 충돌 해결을 하고 있습니다.

좋아요 · 답글 달기 · 8 · 3년