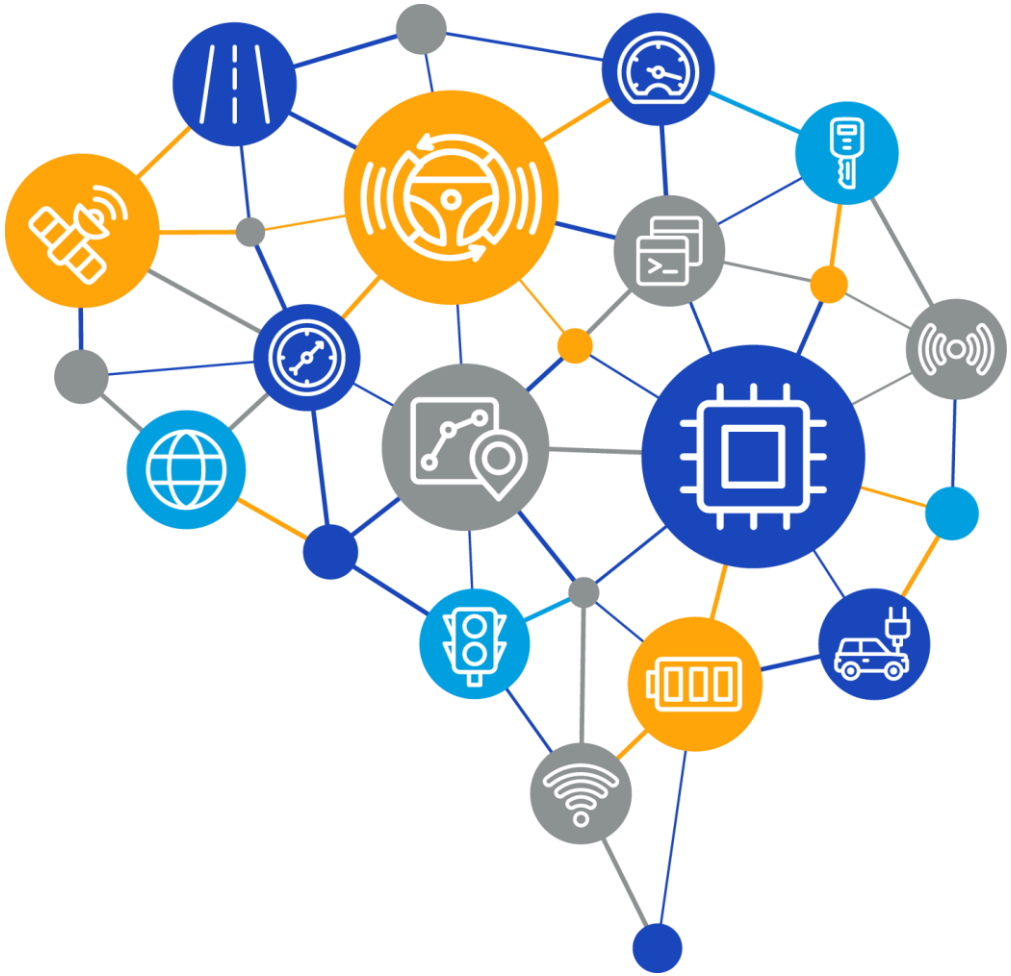
브랜치 개요와 기초 명령어



Git & Github
for pull request

강환수 교수

DMUAi 동양미래대학교
인공지능소프트웨어학과

AI Experts Who Lead
The Future

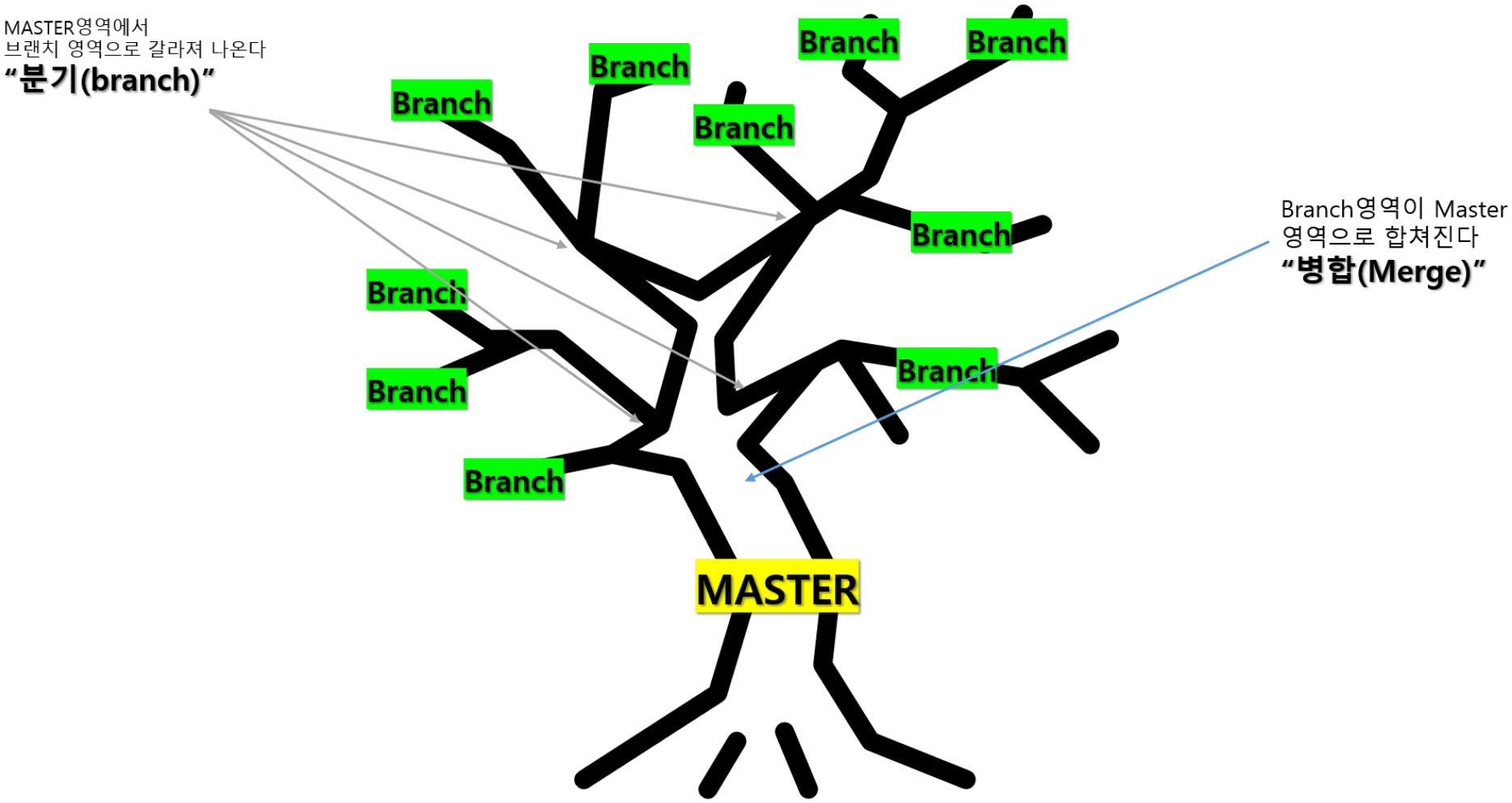
AI Experts
Who Lead
The Future

01

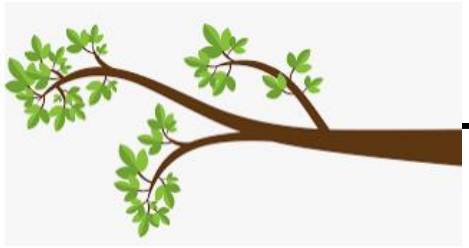
브랜치 이해

브랜치

- 실 행활에서 용어
  - 가지
- Branch, merge



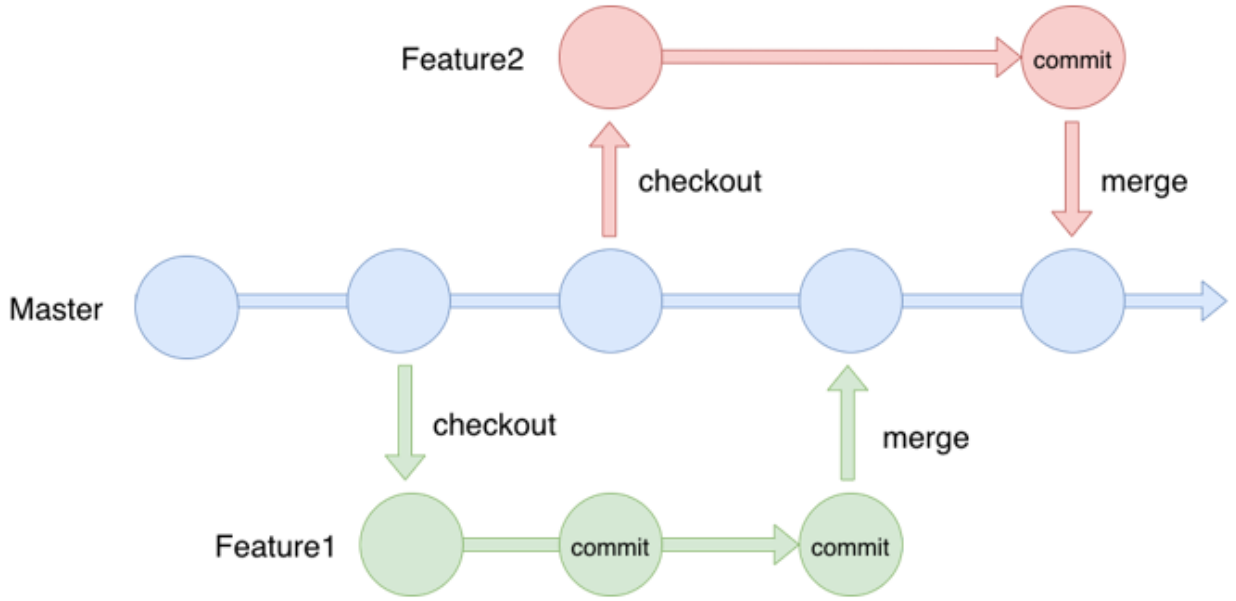
- 버전 관리의 깃허브



GIT Branch and its Operations.
git
Feature - 2
File-2 v1.1
File-4 v1.1
Master
File-1 v1.0
File-2 v1.0
File-3 v1.0
File-4 v1.0
File-1 v1.1
File-2 v1.0
File-3 v1.1
File-4 v1.0
File-1 v1.1
File-2 v1.1
File-3 v1.1
File-4 v1.1
Feature - 1
File-1 v1.0.1
File-3 v1.0.1
File-1 v1.1
File-3 v1.1

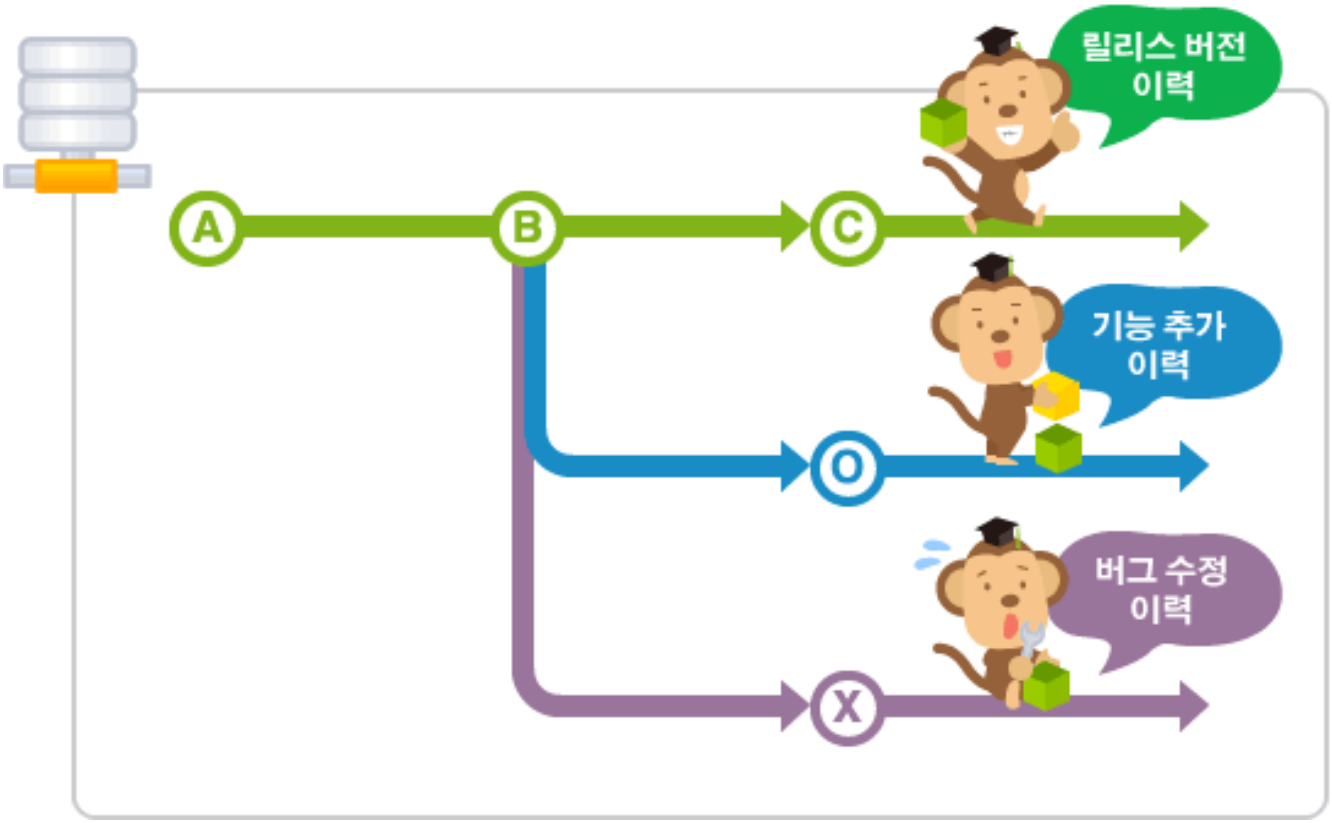
브랜치 개념

- **브랜치 개요**
  - 개발과정에서 각각 서로 다른 버전의 코드가 만들어 질 수 있는 상황
    - 동일한 소스코드 위에서 어떤 개발자는 버그를 수정
    - 또 다른 개발자는 새로운 기능을 만들어 냄
  - '브랜치(Branch)'
    - 여러 개발자들이 동시에 다양한 작업을 할 수 있게 만들어 주는 기능
    - 각자 독립적인 작업 영역(저장소) 안에서 마음대로 소스코드를 변경
      - 분리된 작업 영역에서 변경된 내용은 나중에 원래의 버전과 비교해서 하나의 새로운 버전으로 생성 가능



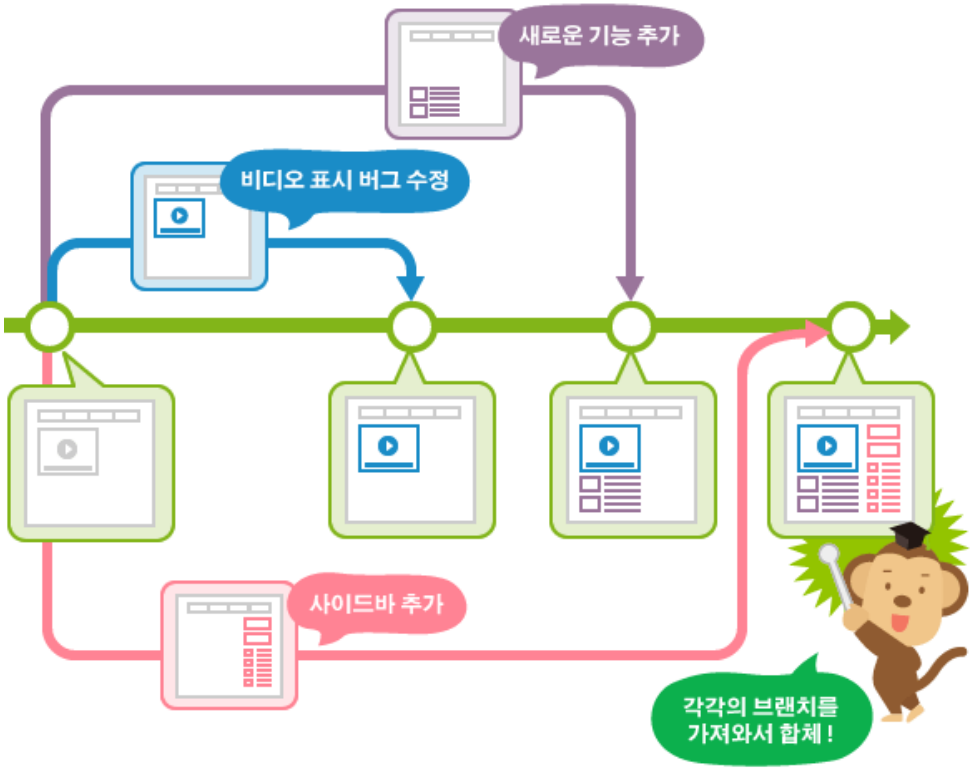
브랜치와 병합 1/2

- **독립적으로 어떤 작업을 진행하기 위한 개념**
  - 필요에 의해 만들어지는 각각의 브랜치는 다른 브랜치의 영향을 받지 않기 때문
    - 여러 작업을 동시에 진행 가능
- **새로운 하나의 브랜치로 모으는 방법**
  - 만들어진 브랜치는 다른 브랜치와 병합(Merge)함으로써, 작업한 내용을 다시 병합



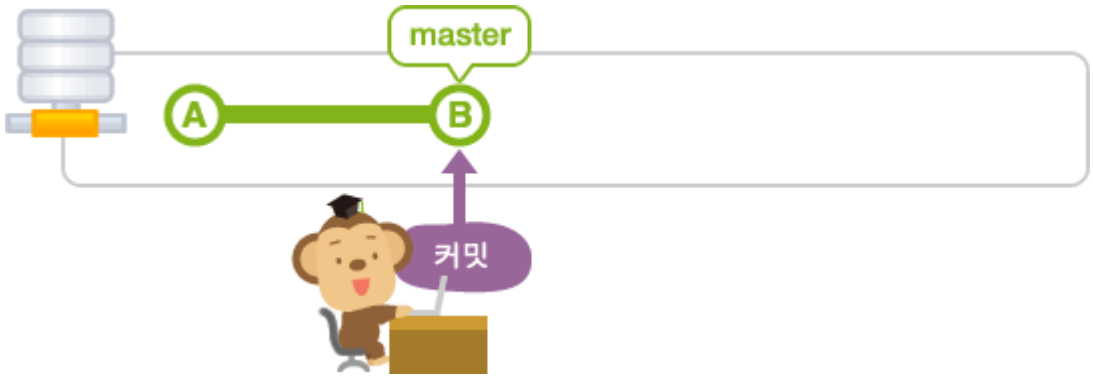
브랜치와 병합 2/2

- 메인 브랜치에서 자신의 작업 전용 브랜치를 생성
  - 여러 명이서 동시에 작업을 할 때에 다른 사람의 작업에 영향을 주거나 받지 않도록
  - 각자 작업을 진행한 후
    - 작업이 끝난 사람은 메인 브랜치에 자신의 브랜치의 변경 사항을 적용
    - 다른 사람의 작업에 영향을 받지 않고 독립적으로 특정 작업을 수행
- 병합: 그 결과를 하나로 모아 나가게 됨



main(master) 브랜치

- 저장소를 처음 만들면, Git은 바로 'main' 이름의 브랜치를 생성
  - 모두 'main' 이름의 브랜치를 통해 처리할 수 있는 일
    - 이 새로운 저장소에 새로운 파일을 추가 하거나
    - 추가한 파일의 내용을 변경하여 그 내용을 저장(커밋, Commit)하는 것



  - 이 때의 모든 작업은 'main' 브랜치에서 이루어 짐
  - 'main'가 아닌 또 다른 새로운 브랜치를 만들어서
    - '이제부터 이 브랜치를 사용할거야!'라고 생성(branch, checkout -b)하면 가능

- 새로운 브랜치 생성과 이동 명령
  - 생성
    - branch, switch -c, checkout -b
  - 이동
    - switch, checkout

브랜치(Branch) 장점과 활용

- **브랜치 장점**
  - 하나의 큰 작업 흐름의 단위인 브랜치로 작업
    - 작업의 기록을 중간 중간에 남기게 되므로
    - 문제가 발생했을 경우 원인이 되는 작업을 찾아내거나
    - 그에 따른 대책을 세우기 쉬워짐

- **브랜치 생성 및 활용 규칙 제정**
  - 새로운 브랜치의 이름은 어떤 규칙으로 지을 것인지
    - 어떤 상황에서 브랜치를 만들지, 어느 시점에 통합할 것인지 등등 규칙은 정해 활용
  - 일반적 규칙: 총 5가지의 브랜치
    - 항상 유지되는 메인 브랜치들
      - main, develop
    - 일정 기간 동안만 유지되는 보조 브랜치들
      - feature, release, hotfix

- **통합 브랜치(Integration Branch): main**
  - 언제든지 배포할 수 있는 버전을 만들 수 있어야 하는 브랜치
  - 늘 안정적인 상태를 유지하는 것이 중요
    - '그 어플리케이션의 모든 기능이 정상적으로 동작하는 상태'를 의미

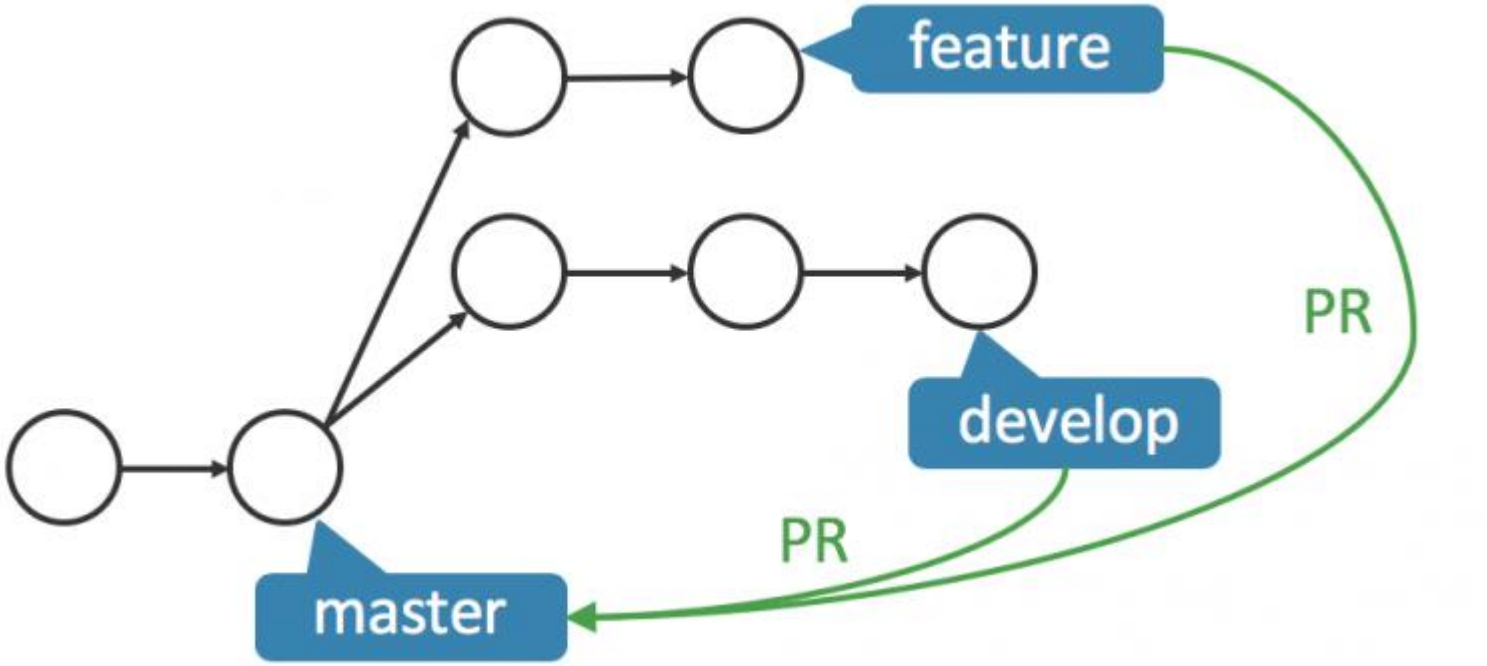
Branch 정리

- 분기사용
  - 저장소에서 다른 브랜치에는 영향 없이
    - 새로운 기능을 개발하거나
    - 버그를 수정하거나
    - 새로운 아이디어를 안전하게 실험 가능

- 기본 브랜치(default branch)
  - 이름 기본이 main
    - 저장소가 생성될 때 자동으로 생성되는 base 브랜치
      - 예전에는 master
  - 기본 설정으로 수정
    - $ git config --global init.defaultBranch main
  - 전역 확인
    - $ git config --global init.defaultBranch
    - $ git config --global --get init.defaultBranch
  - 지역 확인
    - $ git config init.defaultBranch
    - $ git config --get init.defaultBranch

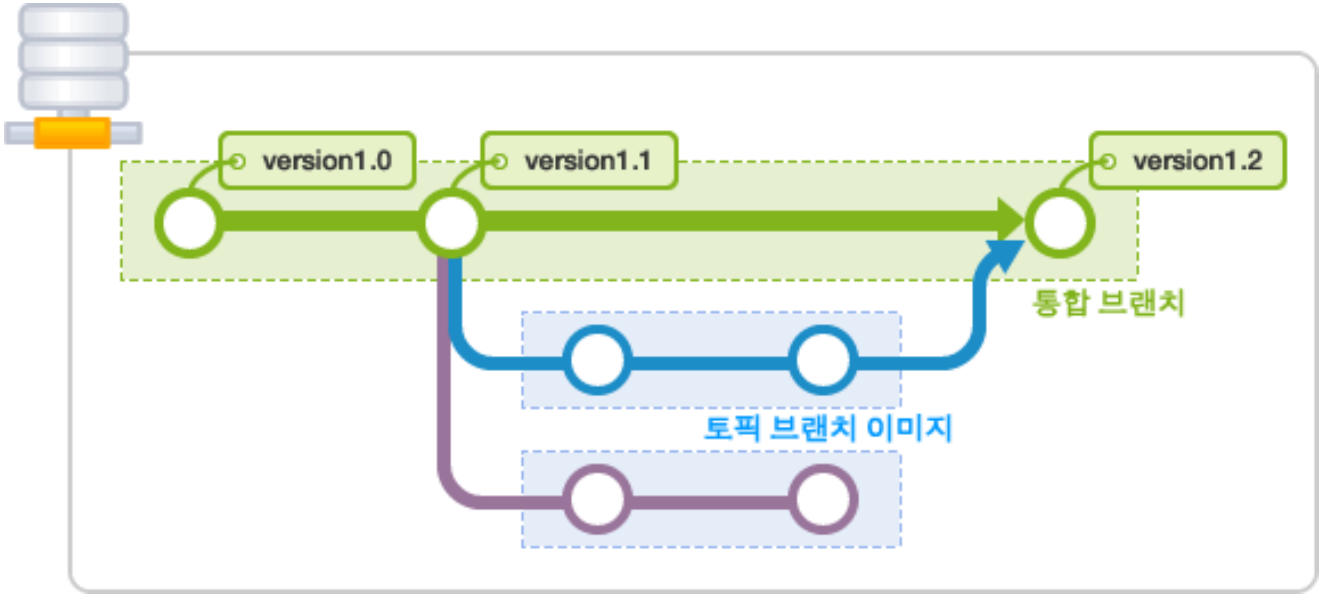
Pull Request, PR, 풀 요청, 끌기 요청

- **Pull Request로 기존 브랜치와 병합**
  - 작업에 만족하면 현재 분기(헤드 분기)의 변경 사항을 다른 분기에 병합하기 위해 풀 요청 수행



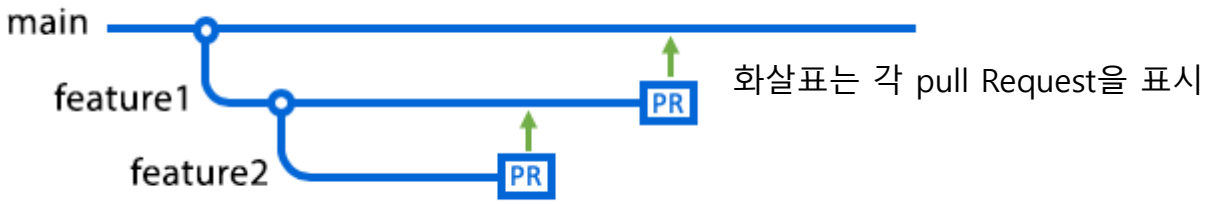
토픽 브랜치(Topic branch)

- 토픽 브랜치는 '피처 브랜치(Feature branch)' 라고 부름
  - 통합 브랜치에서 토픽 브랜치를 생성
  - 기능 추가나 버그 수정과 같은 단위 작업을 위한 브랜치
    - 어떤 문제가 발견되어 그 문제(버그)를 수정하거나
    - 새로운 기능을 추가
  - 토픽 브랜치에서 특정 작업이 완료되면 다시 통합 브랜치에 병합



브랜치 합병 개념

- **main 브랜치에서 브랜치 feature1을 생성**
  - feature1에서 다시 feature2를 생성
    - feature2에서 feature2으로 pull Request
    - Feature1에서 main으로 Pull Request

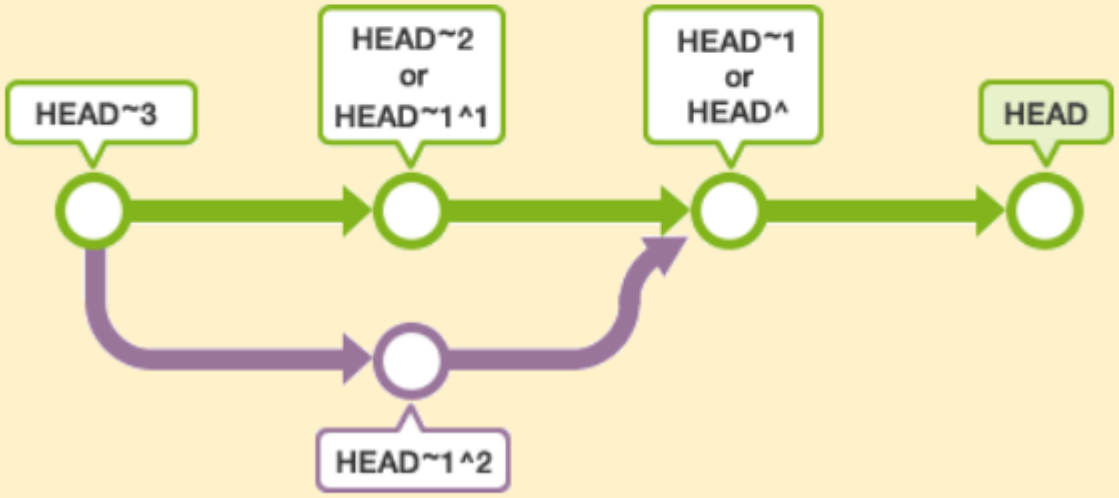


- **main 브랜치에서 브랜치 feature1을 생성**
  - feature1에서 다시 feature2를 생성
    - feature2에서 main으로 pull Request
    - feature1은 그대로 남아 있음

head에서의 ~(틸드)'와 '^(캐럿)' 사용

Note

커밋을 지정할 때, '~(틸드, 물결기호)'와 '^(캐럿, 삽입기호)'을 사용하여 현재 커밋으로부터 특정 커밋의 위치를 가리킬 수 있습니다. 이 때 자주 사용하는 것이 'HEAD' 로서, '~(틸드)'와 숫자를 'HEAD' 뒤에 붙여 몇 세대 앞의 커밋을 가리킬 수 있습니다. '^(캐럿)'은, 브랜치 병합에서 원본이 여럿 있는 경우 몇 번째 원본인지를 지정할 수 있습니다.



AI Experts
Who Lead
The Future

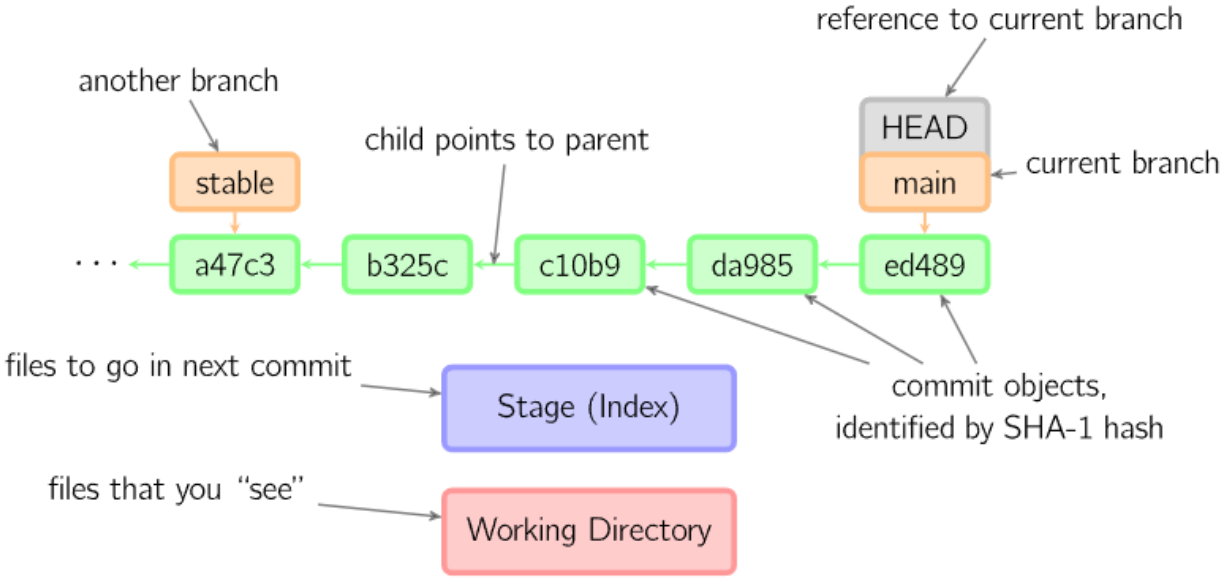
02

깃 브랜치 명령어

https://meetup.toast.com/posts/116

그래프를 사용하여 Git 사용법을 설명

- 커밋은 5글자의 ID로 표현: 녹색
  - 부모 커밋을 화살표로 가리김
- 브랜치
  - 오렌지색이며 어떤 특정 커밋을 가리킴
  - HEAD
    - 현재 브랜치 가리킬 수 있음
- 그림 예
  - 5개의 커밋이 있으며 ed489 커밋이 가장 최근의 커밋
  - main 브랜치(현재 선택한 브랜치)는 가장 최근의 커밋을 가리킴
  - stable 브랜치는 main 브랜치의 조상(Ancestor)에서 가지 치기



브랜치 관련 명령어 1/2

- $ git branch
  - 저장소 목록 보기
- $ git branch -a
  - 저장소 목록 보기, 원격 포함 전체 목록

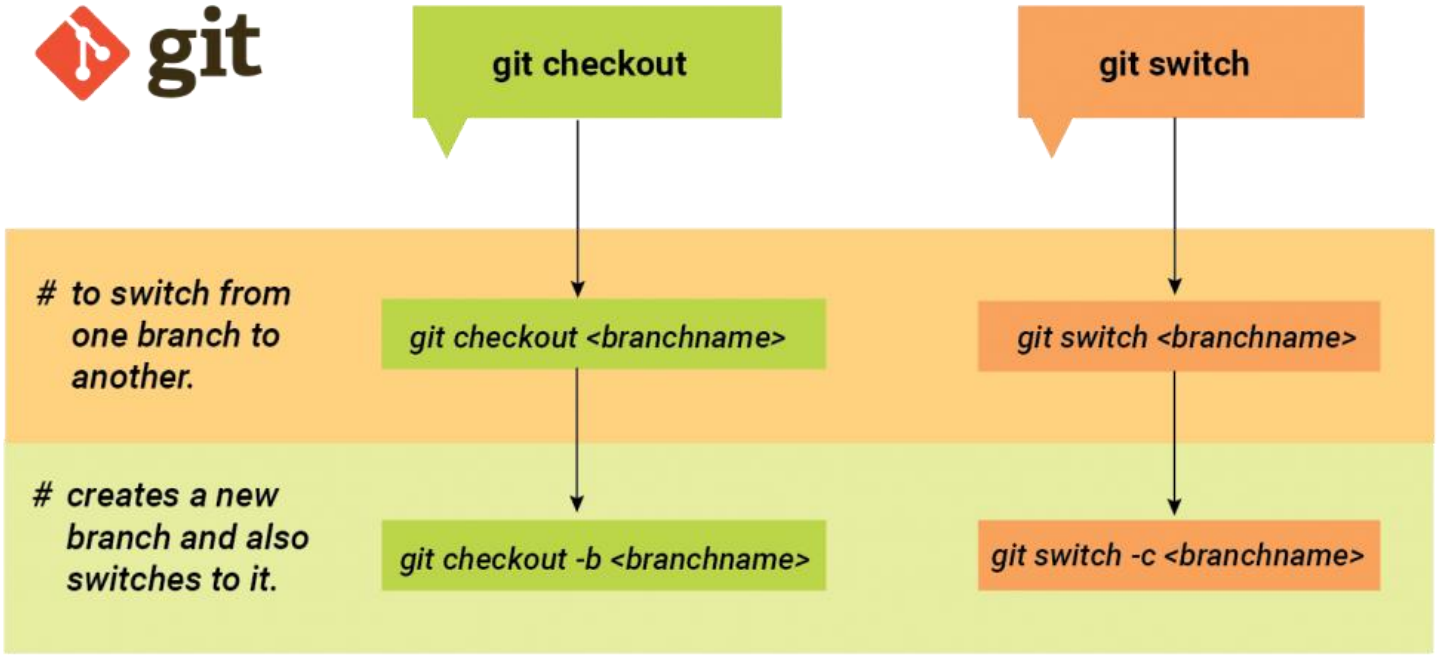
- $ git branch <new-branch>
  - 저장소 생성만

- $ git checkout -b <new-branch>
  - 저장소 생성하고 이동
- $ git switch –c <new-branch>
  - 저장소 생성하고 이동
  - $ git branch <new-branch>
    - 저장소 생성만
  - $ git switch <new-branch>
    - 저장소 이동

- $ git branch –d branch-name
  - 저장소 삭제
- $ git branch –D branch-name
  - 저장소 삭제, 강제 삭제

브랜치 관련 명령어 2/2

- **$ git checkout branch-name**
  - 전환, 이동
- **$ git switch branch-name**
  - 전환, 이동
- **$ git checkout -**
  - 이전 브랜치로 전환, 이동
- **$ git switch -**
  - 이전 브랜치로 전환, 이동
- **$ git branch –h**
  - 도움말 보기



실습 준비

- **저장소 생성**
  - **branch-lab**

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]
$ pwd
/c/[git tutorial]

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]
$ git init branch-lab
Initialized empty Git repository in C:/[git tutorial]/branch-lab/.git/

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]
$ cd branch-lab

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ pwd
/c/[git tutorial]/branch-lab
```

브랜치 main에서 파일 생성 후 2번 커밋

- 파일 basic.py
  - print('branch basic’)
  -
- $ git add basic.py
- $ git commit -m ‘Create basic.py, main’

- 파일 basic.py 수정
  - print(' branch basic’)
  - print(1, 2, 3)

- $ git commit -am ‘Add print literals, main'

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ cat basic.py
print('branch basic')

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git add basic.py

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git commit -m 'Create basic.py, main'
[main (root-commit) fab1006] Create basic.py, main
 1 file changed, 1 insertion(+)
 create mode 100644 basic.py

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git lo
fab1006 (HEAD -> main) Create basic.py, main
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git lo
fab1006 (HEAD -> main) Create basic.py, main

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ D:/Python3102/python.exe "c:/[git tutorial]/branch-lab/basic.py"
branch basic
1 2 3

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git commit -am 'Add print literals, main'
[main 834e72e] Add print literals, main
 1 file changed, 1 insertion(+)

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git lo
834e72e (HEAD -> main) Add print literals, main
fab1006 Create basic.py, main
```

새 브랜치 생성

- **$ git branch feat/list**
  - 우리 실습

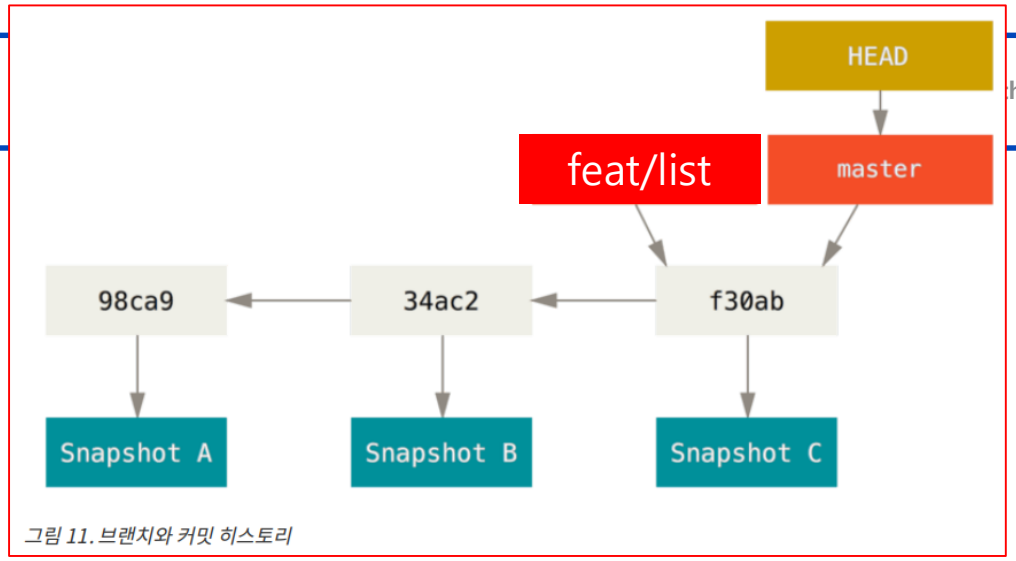


그림 11. 브랜치와 커밋 히스토리

- **'HEAD'라는 특수한 포인터**
  - 이 포인터는 지금 작업하는 로컬 브랜치를 가리킴

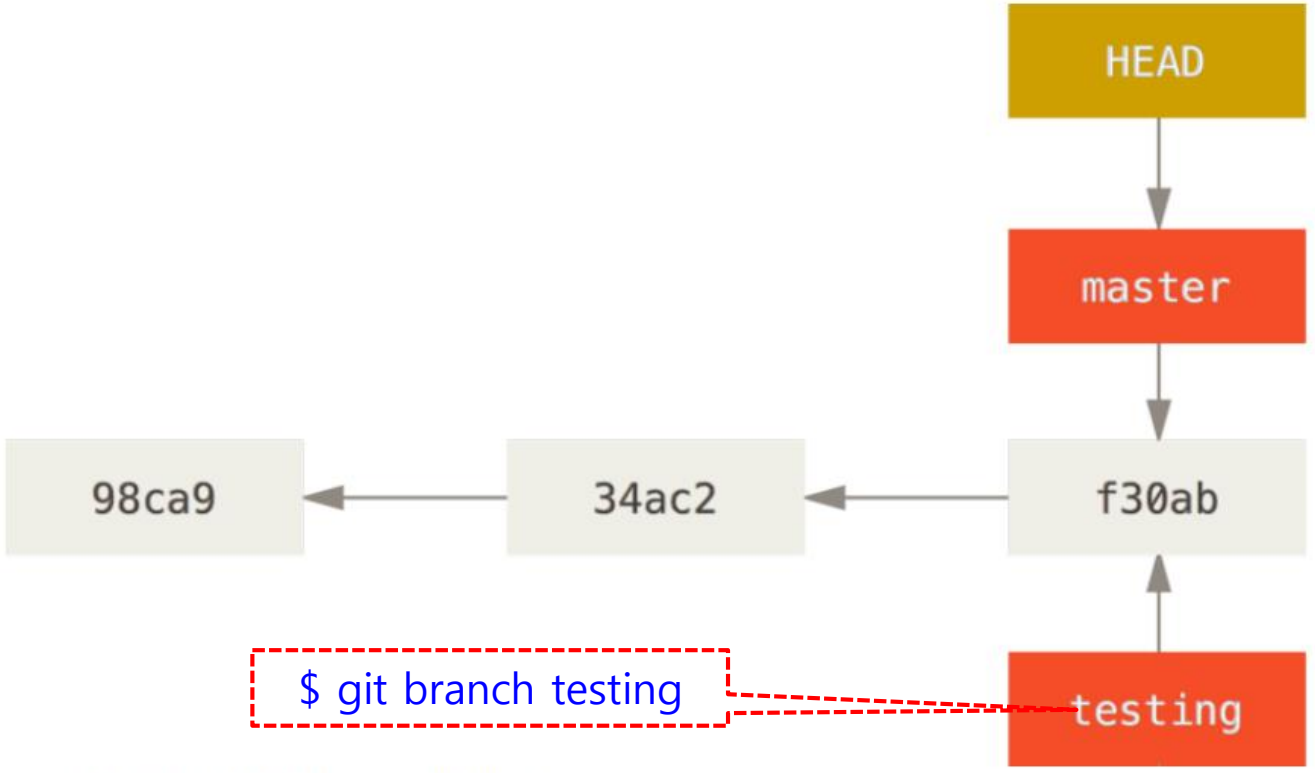


그림 13. 현재 작업 중인 브랜치를 가리키는 HEAD

브랜치 생성 실습

- **$ git branch -a**
  - * main

- **$ git branch feat/list**
  - 브랜치 이름 feat/list 생성

- **$ git branch -a**
  - feat/list
  - * main

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git branch -a
* main

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git branch feat/list

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git branch -a
  feat/list
* main
```

단축 명령 설정

- $ git config --global alias.logg 'log --graph'
- $ git config --global alias.logo 'log --oneline --graph'

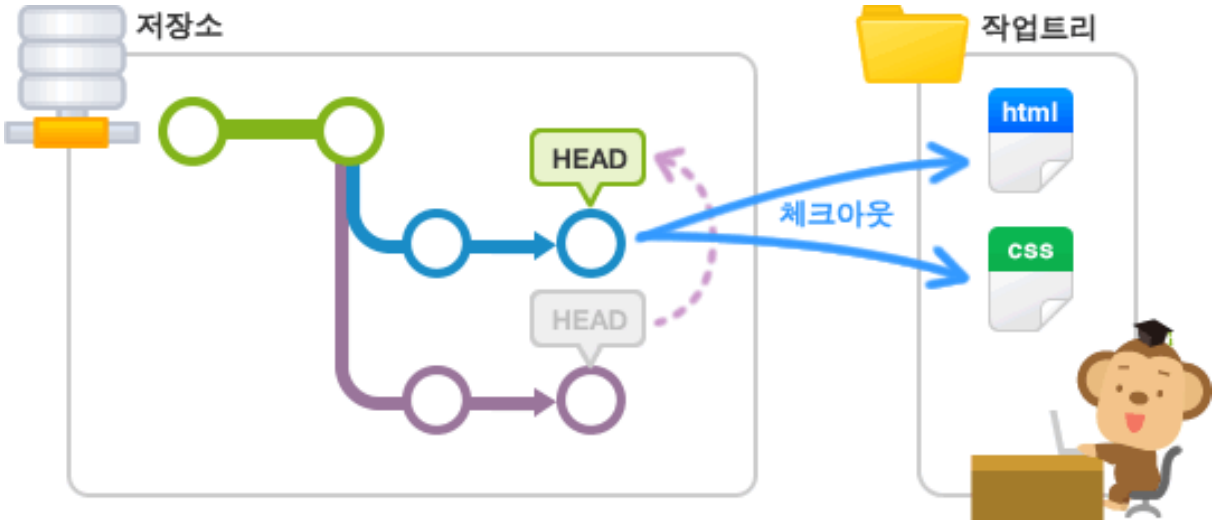
```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git config --global alias.logg 'log --graph'

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git logg
* commit 834e72e6ba2c21f921f345e4b2a6e15c93792171 (HEAD -> main, feat
/list)
| Author: edu4py <edu4py@outlook.kr>
| Date:   Sun May 29 09:53:34 2022 +0900
|
|     Add print literals, main
|
* commit fab10065687594c7e835b44c38d1a40694778545
  Author: edu4py <edu4py@outlook.kr>
  Date:   Sun May 29 09:50:24 2022 +0900

      Create basic.py, main
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git config --global alias.logo 'log --oneline --graph'

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git logo
*   c1d4ca6 (HEAD -> main) Resolve conflict, main
|\
| * 3c4835f (feat/list) add print of tuple, feat/list
* | 2ecae57 add divmod, main
|/
* 56361e0 Add print list of range, feat/list
* 5be70c1 Add print list, feat/list
* 834e72e Add print literals, main
* fab1006 Create basic.py, main
```

브랜치 전환(이동) 1/2

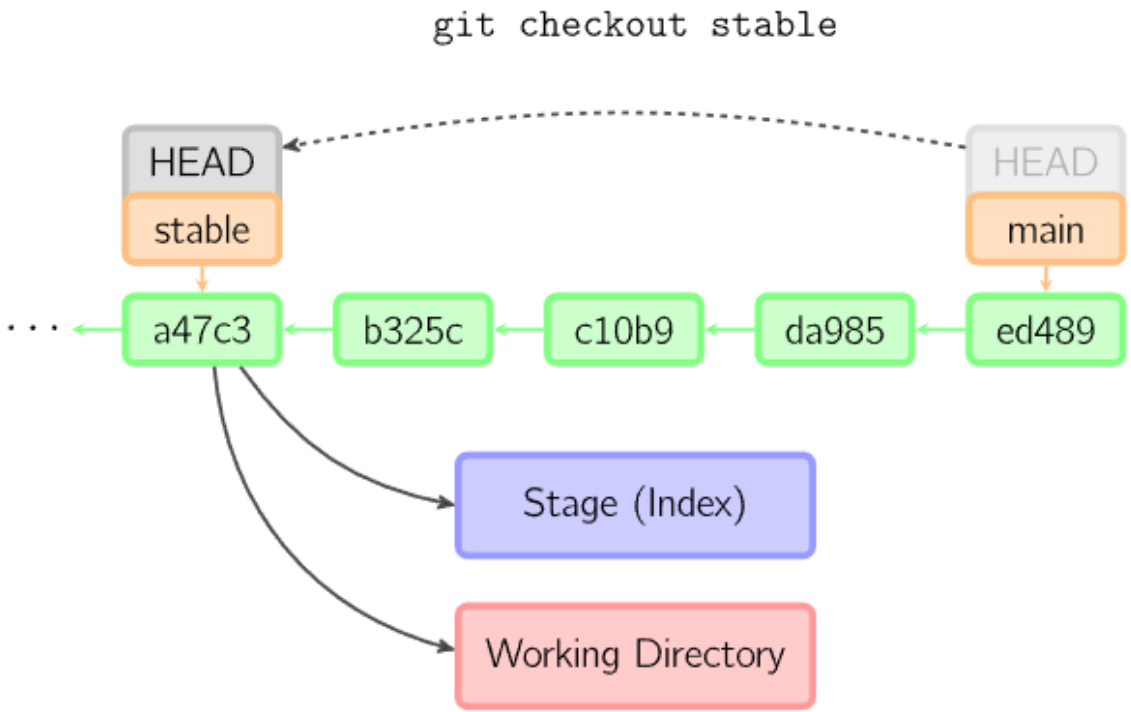
- $ git checkout branch-name
- $ git switch branch-name
- $ git checkout - # 바로 이전 브랜치로 이동

- 항상 작업할 브랜치를 미리 선택
  - Git을 설치하게 되면 'main' 브랜치가 선택
- HEAD
  - 현재 사용 중인 브랜치의 선두 부분을 나타내는 이름
    - 기본적으로는 'main'의 선두 부분을 나타냄
  - 'HEAD' 를 이동하면, 사용하는 브랜치가 변경



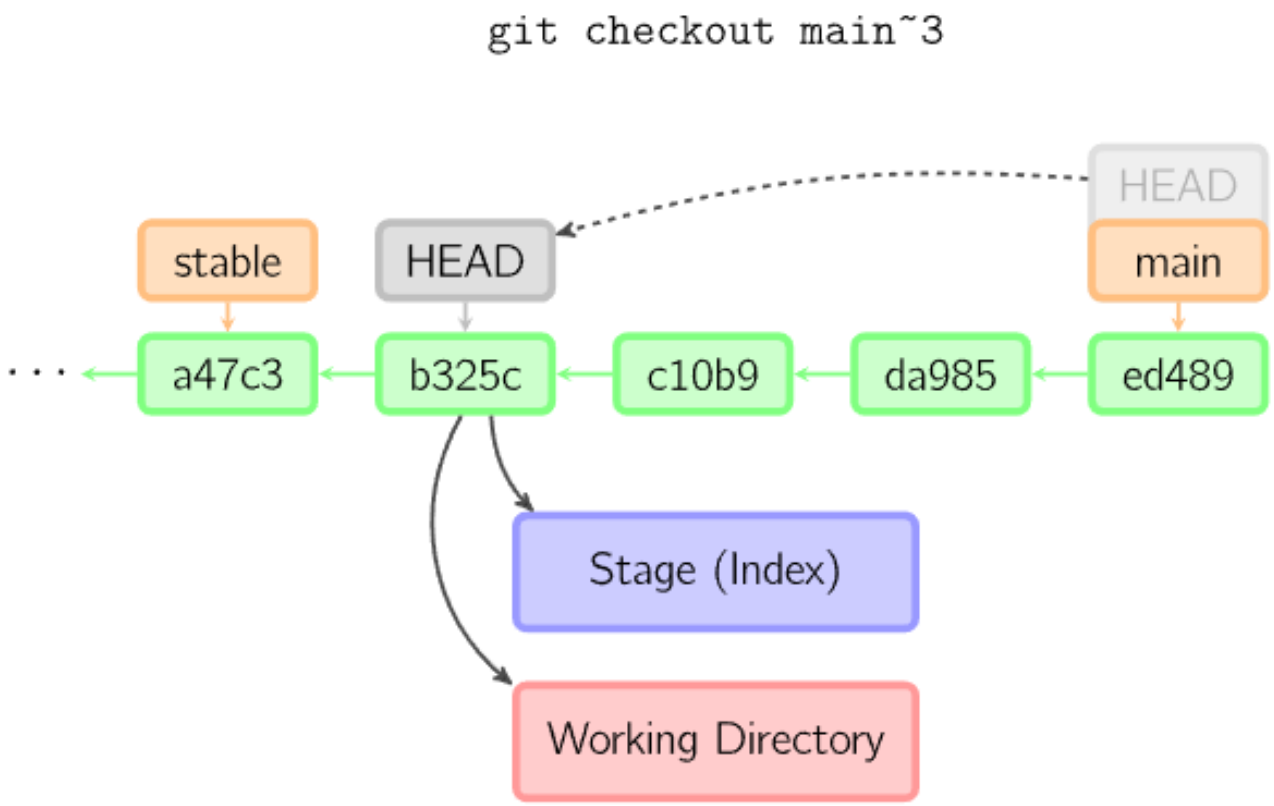
브랜치 전환(이동) 2/2

- **branch_name으로 브랜치 변경**
  - HEAD를 지정한 브랜치를 가리키도록 변경
  - 자동으로 Stage 영역과 작업 디렉토리의 내용은 해당 브랜치의 내용으로 변경
- **$ git checkout stable**
  - a47c3 커밋에 포함된 파일들을 현재 디렉토리로 복사
  - 이전 ed489 커밋에는 포함되었지만 a47c3 커밋에 포함되어 있지 않은 파일들은 삭제



이전 커밋[브랜치] 전환(이동)

- **$ git checkout main~3**



배포버전[브랜치] 전환(이동)

- 프로젝트의 히스토리를 옮겨다닐 때 detached HEAD 상태
  - 체크아웃으로 과거의 이력에 해당하는 커밋으로 이동한 경우
  - 이를 '분리된 헤드(detached HEAD)' 상태에 있다고 함

- $ git checkout v1.6.6.1
  - 'Git 프로젝트'의 1.6.6.1 버전을 컴파일
    - 소스를 Checkout하여 컴파일하고, 그 결과 바이너리들을 설치
    - 익명 브랜치 v1.6.6.1은 브랜치는 아니고 태그
  - 되돌아오기
    - 다시 git checkout main 명령으로 다른 브랜치로 변경

실습

- 브랜치 feat/list로 이동

- $ git checkout feat/list
- $ git switch feat/list

- $ git checkout –
  - 이전 브랜치로 이동

- $ git branch
  - 현재 확인 브랜치 확인

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git branch
  feat/list
* main

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git checkout feat/list
Switched to branch 'feat/list'

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ git branch
* feat/list
  main

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ git switch main
Switched to branch 'main'

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git branch
  feat/list
* main

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git switch -
Switched to branch 'feat/list'
```

브랜치 feat/list에서 2번 commit

- $ git branch
  - 브랜치 확인

- 파일 basic.py 수정
  - print('branch basic')
  - print(1, 2, 3)
  - print([1, 2, 3]) # 3번째 커밋

- $ git commit --am ‘Add print list, feat/list'

- 파일 basic.py 수정
  - print('branch basic')
  - print(1, 2, 3)
  - print([1, 2, 3]) # 3번째 커밋
  - print(list(range(1, 11))) # 4번째 커밋

- $ git commit -am ‘Add print list of range, feat/list’
  - 혹시 오류가 난다면

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ cat basic.py
print('branch basic')
print(1, 2, 3)
print([1, 2, 3])

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ git commit -am 'Add print list, feat/list'
[feat/list 5be70c1] Add print list, feat/list
 1 file changed, 1 insertion(+)

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ git lo
5be70c1 (HEAD -> feat/list) Add print list, feat/list
834e72e (main) Add print literals, main
fab1006 Create basic.py, main
```

Add, commit

- $ git commit -am ‘Add print list of range, feat/list’
  - 혹시 오류가 난다면

- $ git add nasic.py
- $ git commit -m ‘Add print list of range, feat/list’
  - 분리해서

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ git st
On branch feat/list
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working director
y)
        modified:   basic.py

no changes added to commit (use "git add" and/or "git commit -a")

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ git add basic.py

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ git commit -m 'Add print list of range, feat/list'
[feat/list 56361e0] Add print list of range, feat/list
 1 file changed, 1 insertion(+)
```

커밋 로그

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ git logg
* commit 56361e06d63dfe1f850aacd913b041cb5e2f5ae4 (HEAD -> feat/list)
| Author: edu4py <edu4py@outlook.kr>
| Date:   Sun May 29 10:26:38 2022 +0900
|
|     Add print list of range, feat/list
|
* commit 5be70c13d396a4ea3ccee0ca516dfff26829096c
| Author: edu4py <edu4py@outlook.kr>
| Date:   Sun May 29 10:15:50 2022 +0900
|
|     Add print list, feat/list
|
* commit 834e72e6ba2c21f921f345e4b2a6e15c93792171 (main)
| Author: edu4py <edu4py@outlook.kr>
| Date:   Sun May 29 09:53:34 2022 +0900
|
|     Add print literals, main
|
* commit fab10065687594c7e835b44c38d1a40694778545
  Author: edu4py <edu4py@outlook.kr>
  Date:   Sun May 29 09:50:24 2022 +0900

      Create basic.py, main
```

AI Experts
Who Lead
The Future

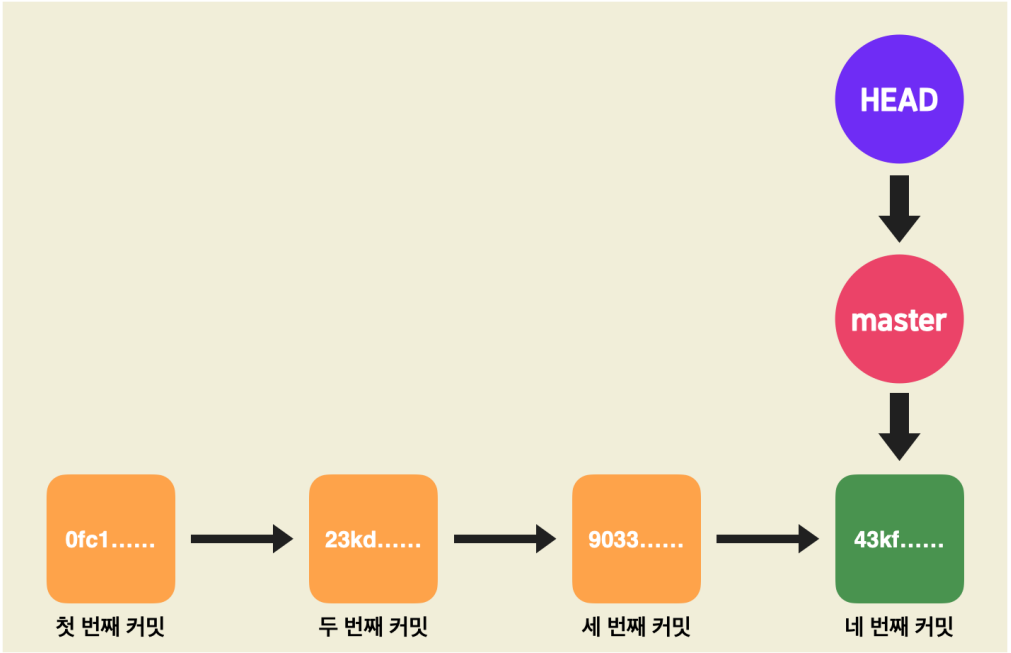
03

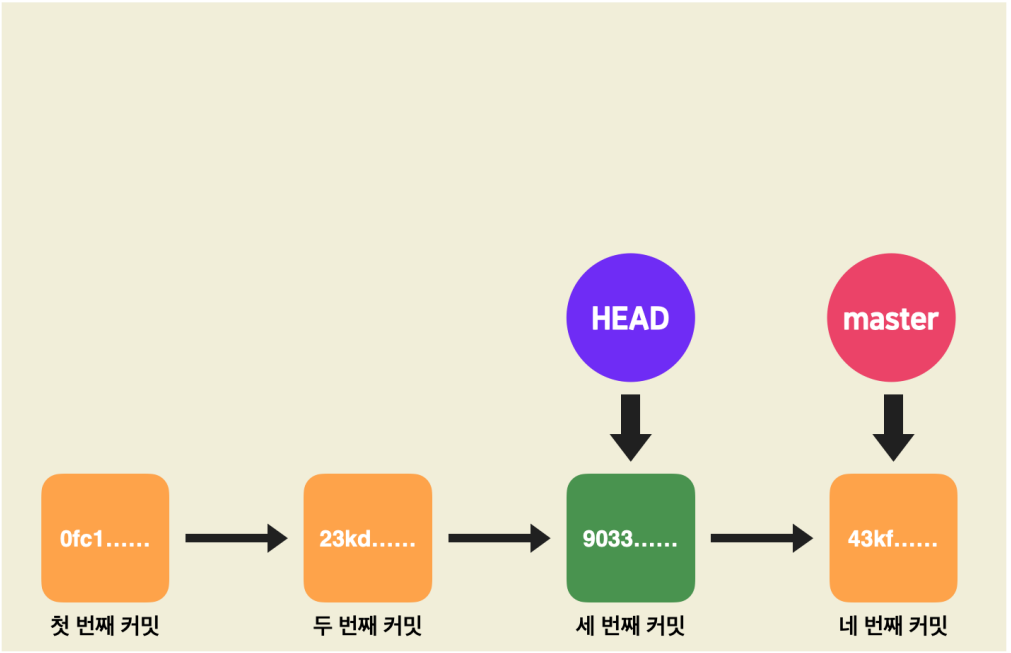
다양 기능의 checkout

Checkout 다양한 이동

- **$ git checkout [커밋]**
  - HEAD 자체가 가리키던 것을 이동

- **$ git checkout 9033**

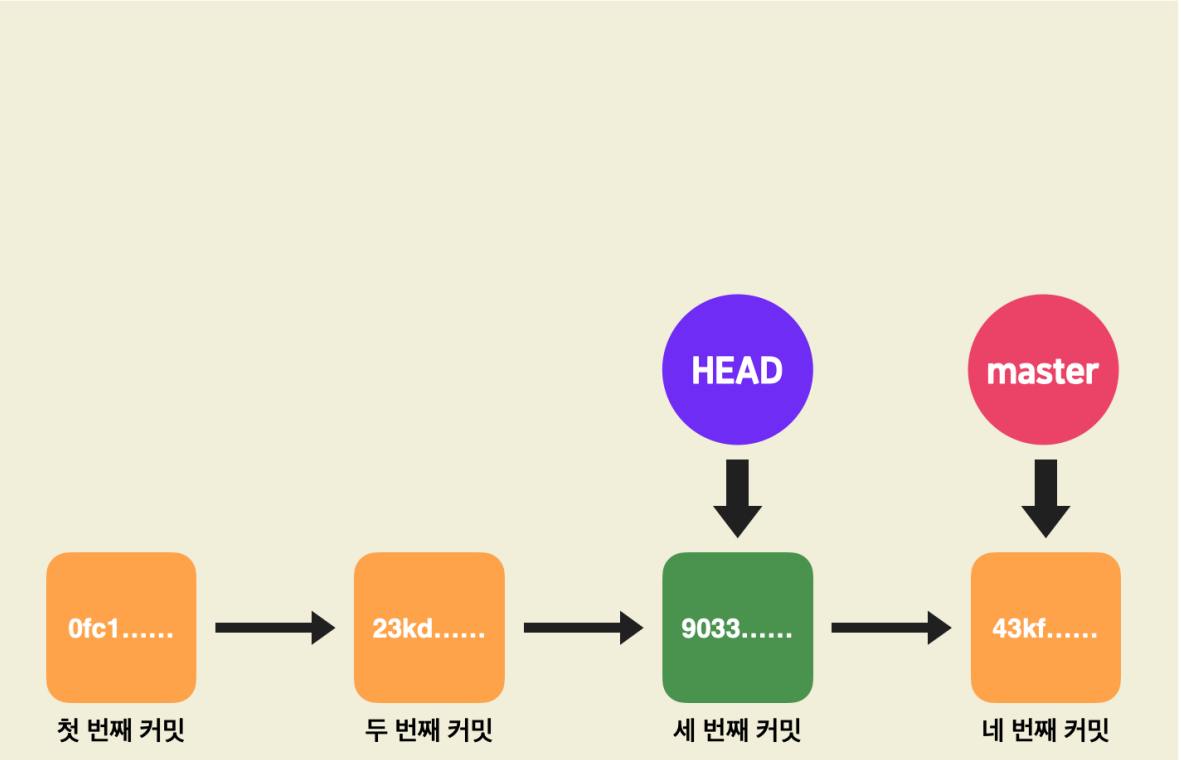




Detached HEAD

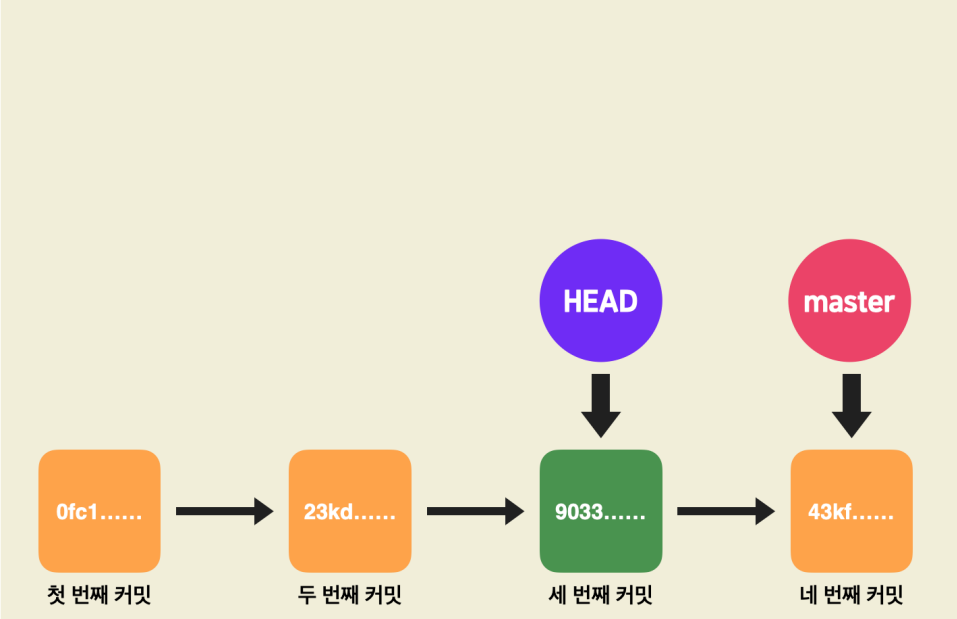
- 브랜치로부터 떨어진 HEAD
  - detached
    - '~로부터 떨어진, 분리된'이라는 뜻

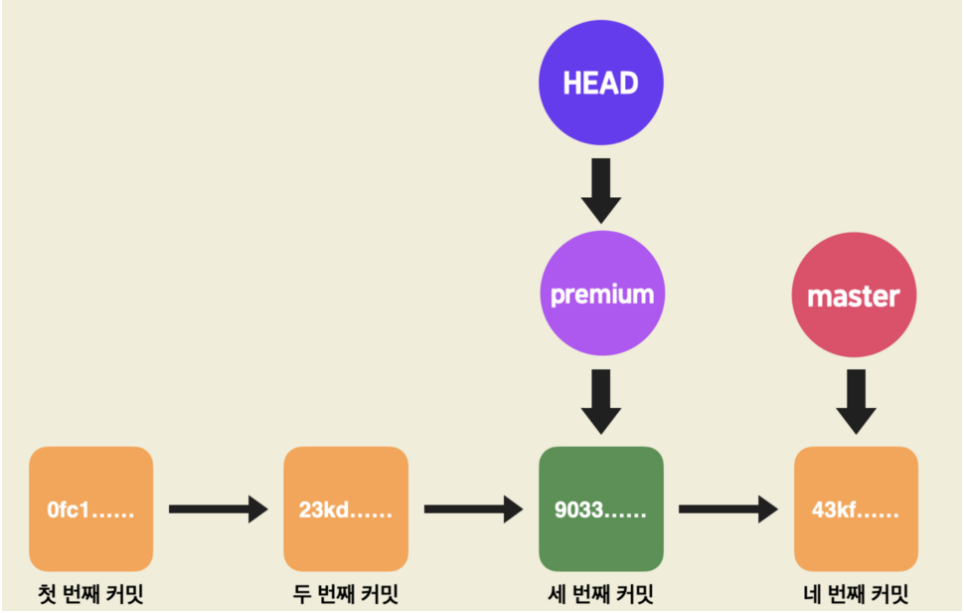
- 이유 중, 한 가지
  - 과거의 특정 커밋에서 새로운 브랜치를 만들고 싶을 때



중간에서 새로운 브랜치 생성

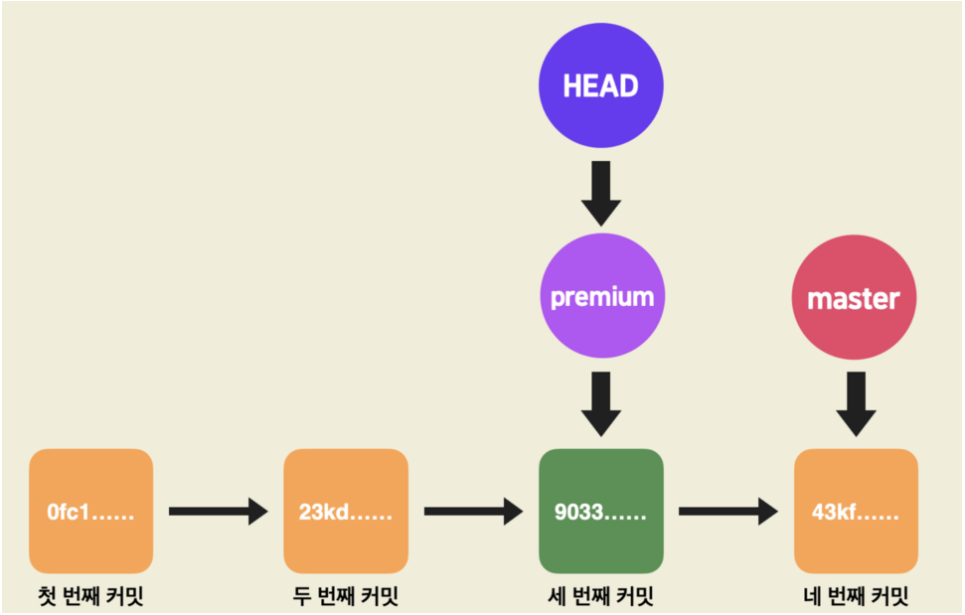
- **$ git branch premium**
  - Detached HEAD인 상태에서 위 실행

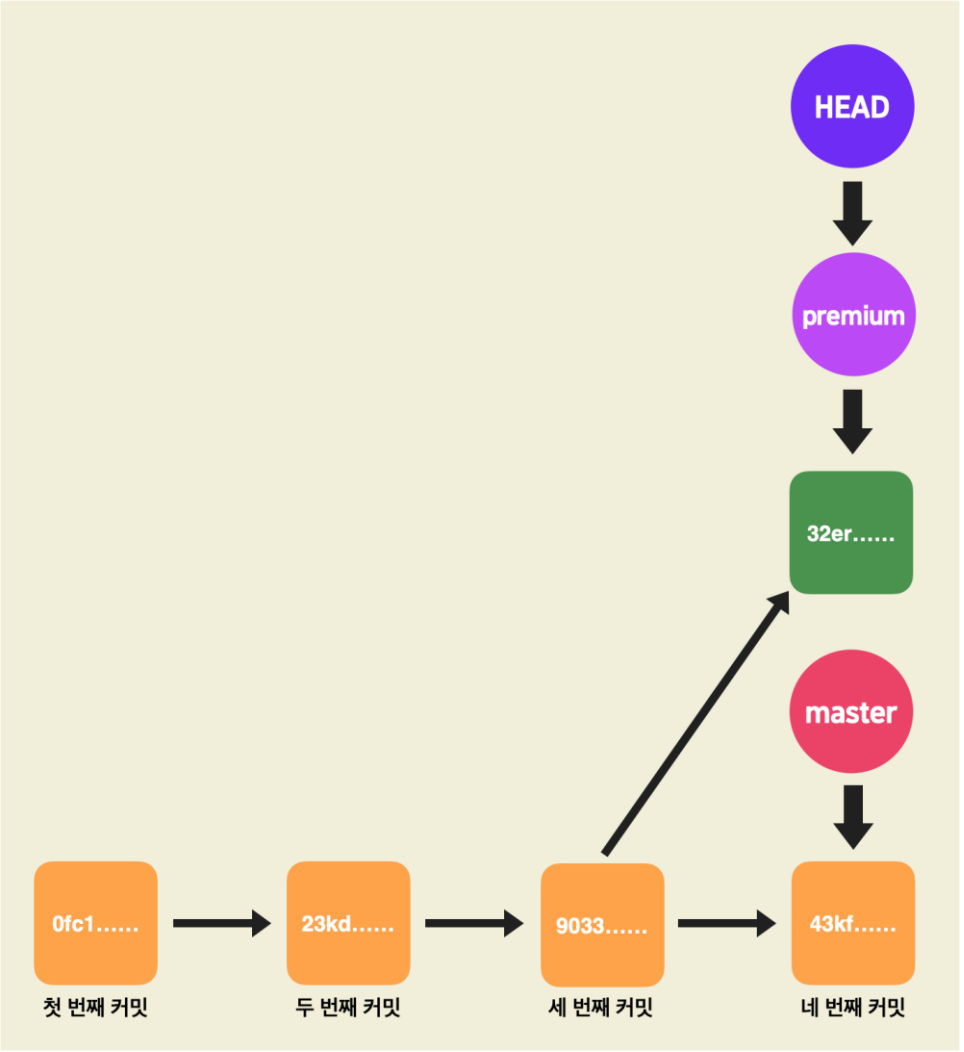




분기에서 커밋

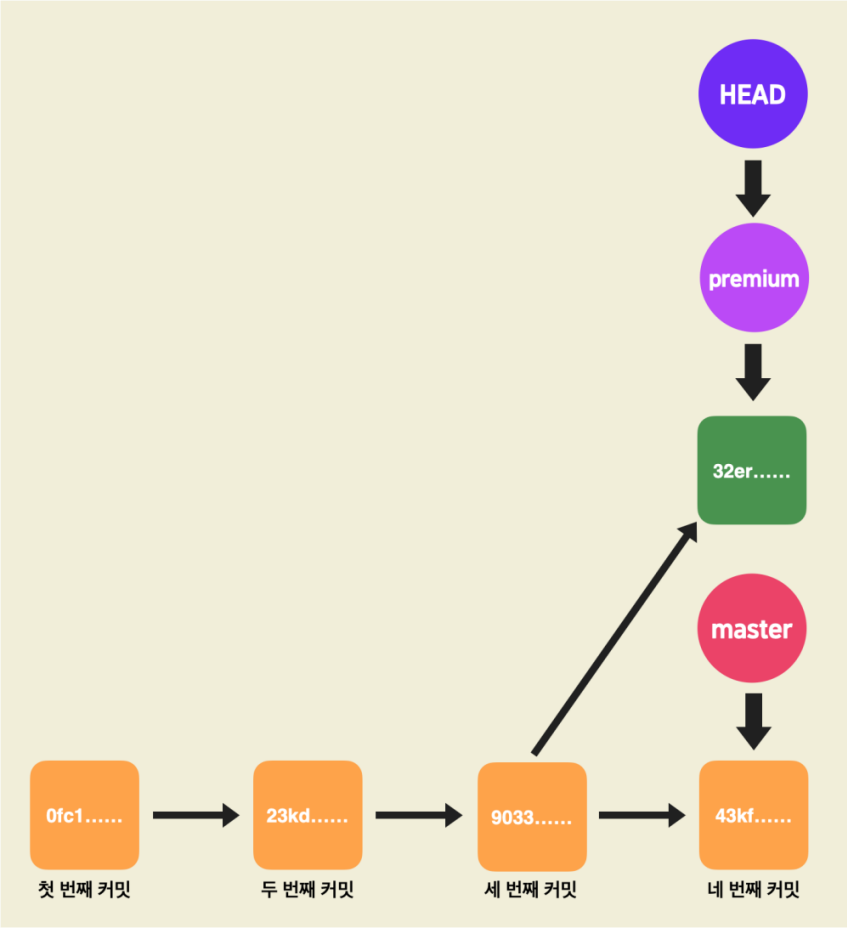
- Premium에서 commit을 하면

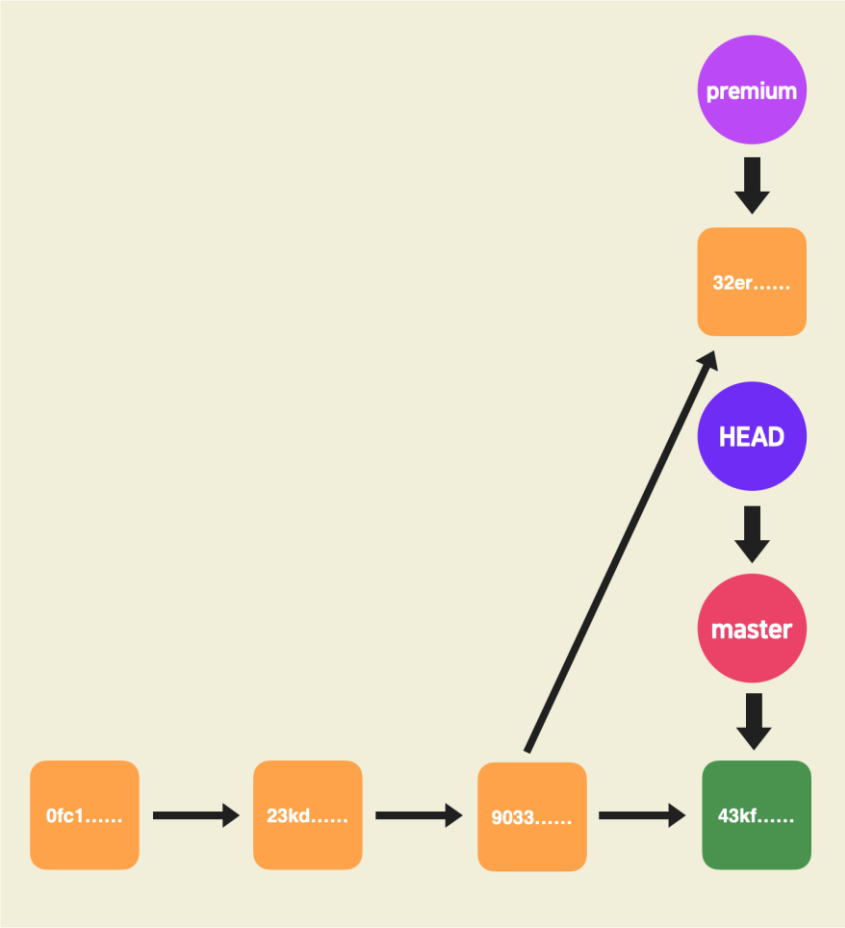




브랜치 이동

- **$ git checkout master**
  - Ptemium에서 master로 이동

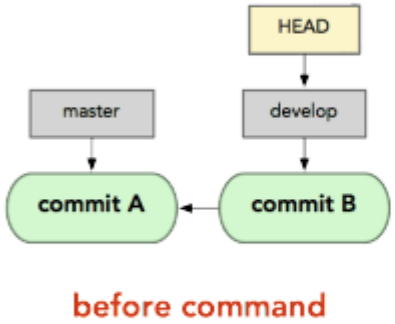


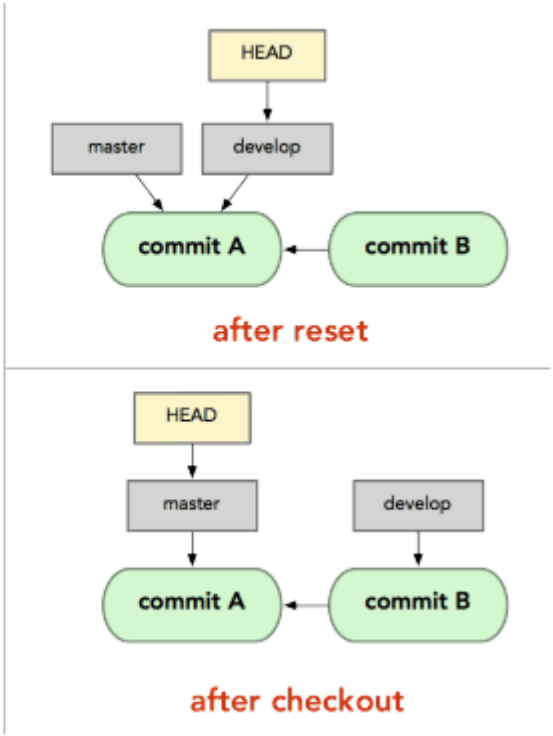


Reset과 checkout 비교

- **Checkout은 head를 이동하고 index와 작업공간도 수정**

git reset	git checkout
HEAD가 가리키던 브랜치가 다른 커밋을 가리키도록 한다	HEAD 자체가 다른 커밋이나 브랜치를 가리키도록 한다
HEAD도 결국 간접적으로 다른 커밋을 가리키게되는 효과가 생긴다	브랜치를 통하지 않고, 커밋을 직접적으로 가리키는 HEAD를 Detached HEAD라고 한다





https://rateye.tistory.com/1260

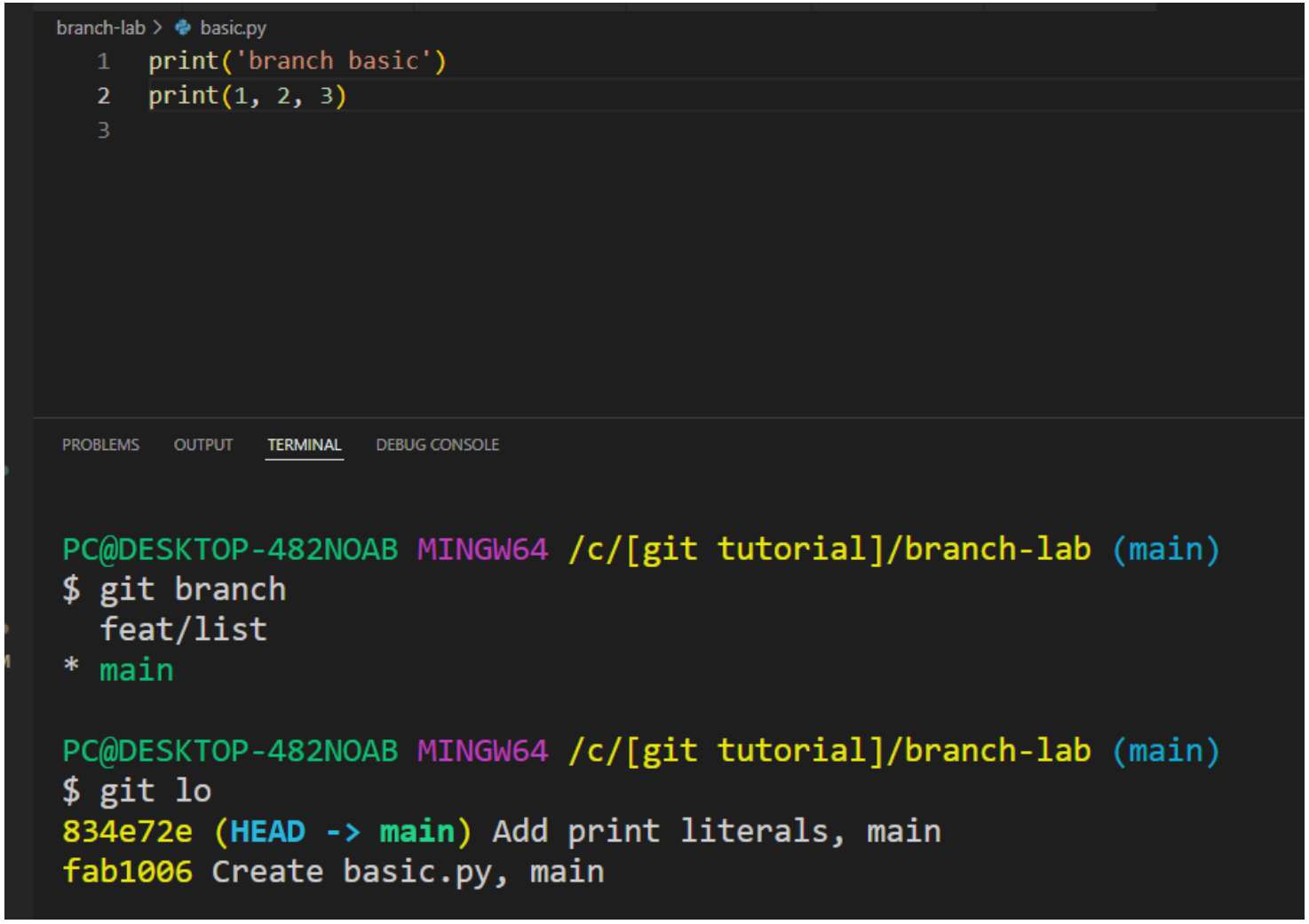
이동 실습

- **$ git checkout main**
  - 브랜치 마지막 커밋으로 이동

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ git lo
56361e0 (HEAD -> feat/list) Add print list of range, feat/list
5be70c1 Add print list, feat/list
834e72e (main) Add print literals, main
fab1006 Create basic.py, main

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/list)
$ git checkout main
Switched to branch 'main'

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git branch
  feat/list
* main

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (main)
$ git show HEAD
commit 834e72e6ba2c21f921f345e4b2a6e15c93792171 (HEAD -> main)
Author: edu4py <edu4py@outlook.kr>
Date:   Sun May 29 09:53:34 2022 +0900

    Add print literals, main

diff --git a/basic.py b/basic.py
index 09a9402..03f262f 100644
--- a/basic.py
+++ b/basic.py
@@ -1 +1,2 @@
 print('branch basic')
+print(1, 2, 3)
```

main 브랜치 확인

- 2개의 커밋 확인



중간으로 HEAD 이동

- **$ git checkout 5be70**
  - 스테이징 영역과 작업 디렉토리에도 복사

```
HEAD is now at 5be70c1 Add print list, feat/list

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab ((5be70
c1...))
$ git lo
5be70c1 (HEAD) Add print list, feat/list
834e72e (main) Add print literals, main
fab1006 Create basic.py, main
```

다시 feat/list로 이동

```
PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab ((5be70
c1...))
$ git switch feat/list
Previous HEAD position was 5be70c1 Add print list, feat/list
Switched to branch 'feat/list'

PC@DESKTOP-482NOAB MINGW64 /c/[git tutorial]/branch-lab (feat/l
ist)
$ git lo
56361e0 (HEAD -> feat/list) Add print list of range, feat/list
5be70c1 Add print list, feat/list
834e72e (main) Add print literals, main
fab1006 Create basic.py, main
```