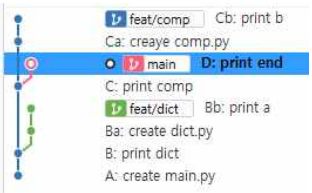
6주차 실습 시트

강 환수 교수

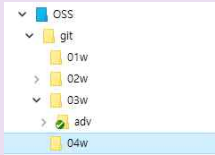
<table>
<tr><td>프로젝트</td><td>

branch & switch, checkout: 브랜치 생성과 이동 [교재 6장]

□ 브랜치 main에서 4 개의 커밋 생성
○ 브랜치 main에서 2개 커밋 후 이동, 브랜치 목록 보기
○ 브랜치 feat/dict 생성 후 새로운 파일 생성과 커밋 2개
○ 브랜치 main으로 이동 후 2개 커밋 후, 바로 이전 커밋에서 브랜치 feat/comp 생성 후 새로운 파일 생성과 커밋



</td></tr>
<tr><td>모듈1</td><td>□ 저장소 brch 생성, 브랜치 main에서 2개의 커밋 먼저 생성, 브랜치 목록 보기</td></tr>
<tr><td>모듈2</td><td>□ 브랜치 생성(branch, checkout –b)·삭제(branch -d)와 이동(switch, checkout)
□ 브랜치 feat/dict 생성 후 파일 dict.py 생성, 2개 커밋</td></tr>
<tr><td>모듈3</td><td>□ 브랜치 main에서 다시 2개 커밋(C, D), 총 4개 커밋</td></tr>
<tr><td>모듈4</td><td>□ 브랜치 main에서 바로 이전 커밋(C, head~)에서 새로운 브랜치 feat/comp를 생성
○ 브랜치 feat/comp에서 새로운 파일 comp.py 생성, 2개의 커밋 이력 생성</td></tr>
<tr><td>모듈5</td><td>□ 이전 커밋 이력으로 head를 이동시키는 checkout
○ ^^, ~~, ~1
□ 소스트리로 열어서 브랜치 확인</td></tr>
<tr><td>준비</td><td>□ 폴더 git/06w 하부에 저장소 brch 생성 후 시작



</td></tr>
<tr><td>모듈1
실습
(15분)</td><td>

□ 저장소 brch 생성, 브랜치 main에서 2개의 커밋 먼저 생성, 브랜치 목록 보기
○ 브랜치 main에서 파일 main.py 생성 후 2개의 커밋, 브랜치 목록 보기

<table>
<tr><td>커밋이력</td><td>A</td><td>B</td><td>C</td><td>D</td></tr>
<tr><td>파일</td><td>main.py</td><td>main.py</td><td>main.py</td><td>main.py</td></tr>
<tr><td>메시지</td><td>A: create main.py</td><td>B: print dict</td><td>C: print comp</td><td>D: print end</td></tr>
<tr><td>내용</td><td>print(‘start’)\n</td><td>print(‘dict’)\n</td><td>print(‘comp’)\n</td><td>print(‘end’)\n</td></tr>
</table>

<table>
<tr><td>

```
$ git init brch
Initialized empty Git repository in C:/OSS/git/06w/brch/.git/

$ cd brch

$ echo "print('start')" >> main.py

$ cat main.py
print('start')

$ git add main.py

$ git commit -m 'A: create main.py'
[main (root-commit) a521e7c] A: create main.py
1 file changed, 1 insertion(+)
create mode 100644 main.py

$ git sts

$ git log1
a521e7c (HEAD -> main) A: create main.py

$ ls
main.py
```

</td><td>커밋 이력 A 생성

첫 파일 생성 이후에 add와 커밋을 차례로 수행</td></tr>
<tr><td>

```
$ echo "print('dict')" >> main.py
```

</td><td>커밋 이력 B 생성</td></tr>
</table>

</td></tr>
</table>

<table>
<tr><td rowspan="2"></td><td>

```
$ cat main.py
print('start')
print('dict')

$ git commit -am 'B: print dict'
[main 9bdc0f2] B: print dict
1 file changed, 1 insertion(+)

$ git log1
9bdc0f2 (HEAD -> main) B: print dict
a521e7c A: create main.py
```

</td><td></td></tr>
<tr><td>

```
$ git branch
* main
```

</td><td>브랜치 목록 검사</td></tr>
<tr><td rowspan="2">모듈2
실습
(20분)</td><td colspan="2">□ 브랜치 생성(branch, checkout –b)·삭제(branch -d)와 이동(switch, checkout)
○ 삭제 옵션: -d(미병합이면 삭제 불가), -D(병합과 무관하게 삭제)</td></tr>
<tr><td>

```
$ git branch hotfix/login          # 브랜치 생성

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
$ git branch                       # 브랜치 목록 보기
hotfix/login
* main

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
$ git switch hotfix/login          # 브랜치 이동
Switched to branch 'hotfix/login'

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (hotfix/login)
$ git branch -h
usage: git branch [<options>] [-r | -a] [--merged] [--no-merged]
or: git branch [<options>] [-l] [-f] <branch-name> [<start-point>]
or: git branch [<options>] [-r] (-d | -D) <branch-name>...
or: git branch [<options>] (-m | -M) [<old-branch>] <new-branch>
or: git branch [<options>] (-c | -C) [<old-branch>] <new-branch>
or: git branch [<options>] [-r | -a] [--points-at]
or: git branch [<options>] [-r | -a] [--format]

...

-a, --all             list both remote-tracking and local branches
-d, --delete          delete fully merged branch
-D                    delete branch (even if not merged)
-m, --move            move/rename a branch and its reflog
-M                    move/rename a branch, even if target exists
-c, --copy            copy a branch and its reflog
-C                    copy a branch, even if target exists
-l, --list            list branch names

...

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (hotfix/login)
$ git branch -d hotfix/login
error: Cannot delete branch 'hotfix/login' checked out at 'C:/OSS/git/06w/brch'

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (hotfix/login)
$ git checkout -
Switched to branch 'main'

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
$ git branch -d hotfix/login       # 빈 브랜치 삭제 가능
Deleted branch hotfix/login (was 9f74e28).

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
$ git branch
* main
```

</td><td>

현재 자신의 브랜치 삭제는 불가능, 다른 브랜치로 이동해서 삭제 가능

브랜치의 삭제 옵션 –d
옵션 –D: 작업하거나 병합이 되지 않은 브랜치의 삭제 무조건 삭제

브랜치의 이동
$ git branch bname
$ git checkout bname

checkout에서만 바로 이전 브랜치 이동 가능
$ git checkout -
$ git branch - # 오류

</td></tr>
</table>

□ 브랜치 feat/dict 생성 후 파일 dict.py 생성, 2개 커밋

커밋이력	Ba	Bb
파일	dict.py	dict.py
메시지	Ba: create dict.py	Bb: print a
내용	a = {1:1, 2:4, 3:9}	print(a)

```
$ git branch feat/dict

$ git branch
feat/dict
* main

$ git branch -v
feat/dict 9bdc0f2 B: print dict
* main       9bdc0f2 B: print dict

$ git switch feat/dict
Switched to branch 'feat/dict'

$ git branch
* feat/dict
  main

$ git checkout main
Switched to branch 'main'

$ git checkout -
Switched to branch 'feat/dict'

$ git branch
* feat/dict
  main
```

-v —verbose
자세한 브랜치 목록

브랜치 생성

브랜치 확인

브랜치 이동

- : 이전 브랜치

git branch ☞ 로컬 저장소의 브랜치 목록
git branch -r ☞ 원격 저장소의 브랜치 목록
git branch -a ☞ 로컬+원격 모든 브랜치 목록

```
$ echo 'a = {1:1, 2:4, 3:9}' > dict.py

$ ls
dict.py  main.py

$ cat dict.py
a = {1:1, 2:4, 3:9}

$ git add
Nothing specified, nothing added.
hint: Maybe you wanted to say 'git add .'?
hint: Turn this message off by running
hint: "git config advice.addEmptyPathspec false"

$ git add .

$ git commit -m 'Ba: create dict.py'
[feat/dict d7b794e] Ba: create dict.py
1 file changed, 1 insertion(+)
create mode 100644 dict.py

$ git log1
d7b794e (HEAD -> feat/dict) Ba: create dict.py
9bdc0f2 (main) B: print dict
a521e7c A: create main.py

$ echo 'print(a)' >> dict.py

$ cat dict.py
a = {1:1, 2:4, 3:9}
print(a)

$ git commit -am 'Bb: print a'
[feat/dict 81eaccf] Bb: print a
1 file changed, 1 insertion(+)

$ git log1
81eaccf (HEAD -> feat/dict) Bb: print a
d7b794e Ba: create dict.py
9bdc0f2 (main) B: print dict
a521e7c A: create main.py
```

브랜치 확인 후 작업

다시 D로 복귀하기 위해
reset --hard 사용

<table>
<tr><td rowspan="4"></td><td colspan="2">□ 브랜치 main으로 이동, 파일 확인</td></tr>
<tr><td>

```
$ git branch
* feat/dict
  main

$ git checkout main
Switched to branch 'main'

$ git branch
feat/dict
* main

$ ls
main.py

$ git log1
9bdc0f2 (HEAD -> main) B: print dict
a521e7c A: create main.py
```

</td><td>다른 브랜치에서 생성한 파일은 보이지 않음

브랜치 main에서 파일 dict.py는 보이지 않음</td></tr>
<tr><td>

```
$ git log
commit 9bdc0f237ba9d44e3d34810cbde30ad204ea63a9 (HEAD -> main)
Author: hskang <ai7dnn@gmail.com>
Date:   Wed Sep 7 12:36:10 2022 +0900

B: print dict

commit a521e7ce3c12b6d13c77fb808ce93fc20f31ea53
Author: hskang <ai7dnn@gmail.com>
Date:   Wed Sep 7 12:27:48 2022 +0900

A: create main.py

$ git log --graph
* commit 9bdc0f237ba9d44e3d34810cbde30ad204ea63a9 (HEAD -> main)
| Author: hskang <ai7dnn@gmail.com>
| Date:   Wed Sep 7 12:36:10 2022 +0900
|
|     B: print dict
|
* commit a521e7ce3c12b6d13c77fb808ce93fc20f31ea53
  Author: hskang <ai7dnn@gmail.com>
Date:   Wed Sep 7 12:27:48 2022 +0900

A: create main.py
```

</td><td>커밋 이력 옵션: --graph</td></tr>
<tr><td>

```
$ git config --global alias.logg 'log --graph'

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
$ git logg
* commit 9bdc0f237ba9d44e3d34810cbde30ad204ea63a9 (HEAD -> main)
| Author: hskang <ai7dnn@gmail.com>
| Date:   Wed Sep 7 12:36:10 2022 +0900
|
|     B: print dict
|
* commit a521e7ce3c12b6d13c77fb808ce93fc20f31ea53
  Author: hskang <ai7dnn@gmail.com>
Date:   Wed Sep 7 12:27:48 2022 +0900

A: create main.py

$ git config --global alias.logg1 'log --graph --oneline'

$ git logg1
* 9bdc0f2 (HEAD -> main) B: print dict
* a521e7c A: create main.py
```

</td><td>별칭 명령 정의
$ git logg
$ git logg1</td></tr>
<tr><td>모듈3
실습
(15분)</td><td colspan="2">□ 브랜치 main에서 다시 2개 커밋(C, D), 총 4개 커밋</td></tr>
</table>

커밋이력	A	B	C	D
파일	main.py	main.py	main.py	main.py
메시지	A: create main.py	B: print dict	C: print comp	D: print end
내용	print(‘start’)\n	print(‘dict’)\n	print(‘comp’)\n	print(‘end’)\n

```
$ echo "print('comp')" >> main.py

$ git commit -am 'C: print comp'
[main 8444489] print comp
1 file changed, 1 insertion(+)

$ git logg1
* 43791fe (HEAD -> main) C: print comp
* 9bdc0f2 B: print dict
* a521e7c A: create main.py

$ echo "print('end')" >> main.py

$ git commit -am 'D: print end'
[main 9f74e28] D: print end
1 file changed, 1 insertion(+)

$ git logg1
* 9f74e28 (HEAD -> main) D: print end
* 43791fe C: print comp
* 9bdc0f2 B: print dict
* a521e7c A: create main.py
```

커밋 이력 C, D 생성

```
$ git checkout feat/dict
Switched to branch 'feat/dict'

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (feat/dict)
$ git logg1
* 81eaccf (HEAD -> feat/dict) Bb: print a
* d7b794e Ba: create dict.py
* 9bdc0f2 B: print dict
* a521e7c A: create main.py

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (feat/dict)
$ git logg1 --all
* 16c3272 (main) D: print end
* 9461d58 C: print comp
| * 3351dd4 (HEAD -> feat/dict) Bb: print a
| * 0a6c23e Ba: create dict.py
|/
* e929d09 B: print dict
* d0f3137 A: create main.py
```

브랜치 이동 후

로그 정보 확인

어느 브랜치에 있든 모든 브랜치의 커밋 이력 보기 옵션 --all

모듈4 실습 (15분)

□ 브랜치 main에서 바로 이전 커밋(C, head~)에서 새로운 브랜치 feat/comp를 생성

○ git checkout –b 생성, 생성할 위치를 마지막 인자로 사용

```
$ git checkout main
Switched to branch 'main'

$ git logg1
* 9f74e28 (HEAD -> main) D: print end
* 43791fe C: print comp
* 9bdc0f2 B: print dict
* a521e7c A: create main.py

$ git checkout -b feat/comp HEAD~
Switched to a new branch 'feat/comp'

$ git logg1
* 43791fe (HEAD -> feat/comp) C: print comp
* 9bdc0f2 B: print dict
* a521e7c A: create main.py
```

새로운 브랜치 feat/comp를 생성

□ 브랜치 feat/comp에서 새로운 파일 comp.py 생성, 2개의 커밋 이력 생성

커밋이력	Ca	Cb
파일	comp.py	comp.py
메시지	Ca: create comp.py	Cb: print b
내용	b = {i:i**2 for i in range(1, 4)}	print(b)

	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/br (feat/comp)
	$ echo 'b = {i:i**2 for i in range(1, 4)}' >> comp.py
	
	$ cat comp.py
	b = {i:i**2 for i in range(1, 4)}
	
	$ git add .
	
	$ ls
	comp.py main.py
	
	$ git commit -m 'Ca: create comp.py'
	[feat/comp bcc9b64] Ca: create comp.py
	1 file changed, 1 insertion(+)
	create mode 100644 comp.py
	
	$ git logg1
	* bcc9b64 (HEAD -> feat/comp) Ca: create comp.py
	* 43791fe C: print comp
	* 9bdc0f2 B: print dict
	* a521e7c A: create main.py
		파일 생성
		
		커밋 이력 모두 보기

	$ echo 'print(b)' >> comp.py
	
	$ cat comp.py
	b = {i:i**2 for i in range(1, 4)}
	print(b)
	
	$ git commit -am 'Cb: print b'
	[feat/comp 8de58d7] Cb: print b
	1 file changed, 1 insertion(+)
	
	$ git logg1
	* 8de58d7 (HEAD -> feat/comp) Cb: print b
	* bcc9b64 Ca: creaye comp.py
	* 43791fe C: print comp
	* 9bdc0f2 B: print dict
	* a521e7c A: create main.py
	
	$ git branch
	* feat/comp
	  feat/dict
	  main
		커밋 추가

	$ git switch main
	Switched to branch 'main'
	
	$ git logg1
	* 9f74e28 (HEAD -> main) D: print end
	* 43791fe C: print comp
	* 9bdc0f2 B: print dict
	* a521e7c A: create main.py
	
	$ ls
	main.py
	
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/br (main)
	$ git logg1 --all
	* c3d31b8 (feat/comp) Cb: print b
	* 47493b4 Ca: create comp.py
	| * 16c3272 (HEAD -> main) D: print end
	|/
	* 9461d58 C: print comp
	| * 3351dd4 (feat/dict) Bb: print a
	| * 0a6c23e Ba: create dict.py
	|/
	* e929d09 B: print dict
	* d0f3137 A: create main.py
		브랜치 main에서는 파일이 main.py만 보임
		
		# 어느 브랜치에 있든 모든 브랜치의 커밋 이력 보기
		옵션 --all

	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
	$ git logg1
	* 9f74e28 (HEAD -> main) D: print end
	* 43791fe C: print comp
	* 9bdc0f2 B: print dict
	* a521e7c A: create main.py
	
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
	$ git log --all --graph --oneline
		모든 브랜치의 모든 로그 정보
		$ git log —all
		
		HEAD가 가리키는 곳: 현재 브랜치의 마지막 커밋

<table>
<tr><td rowspan="2"></td><td>

```
* 8de58d7 (feat/comp) Cb: print b
* bcc9b64 Ca: creaye comp.py
| * 9f74e28 (HEAD -> main) D: print end
|/
* 43791fe C: print comp
| * 81eaccf (feat/dict) Bb: print a
| * d7b794e Ba: create dict.py
|/
* 9bdc0f2 B: print dict
* a521e7c A: create main.py

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
$ git logg1 --all
* 8de58d7 (feat/comp) Cb: print b
* bcc9b64 Ca: creaye comp.py
| * 9f74e28 (HEAD -> main) D: print end
|/
* 43791fe C: print comp
| * 81eaccf (feat/dict) Bb: print a
| * d7b794e Ba: create dict.py
|/
* 9bdc0f2 B: print dict
* a521e7c A: create main.py
```

</td><td></td></tr>
<tr><td>

```
PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (feat/dict)
$ git config --global alias.logg1a 'logg1 --all'

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (feat/dict)
$ git logg1a
* 8de58d7 (feat/comp) Cb: print b
* bcc9b64 Ca: creaye comp.py
| * 9f74e28 (main) D: print end
|/
* 43791fe C: print comp
| * 81eaccf (HEAD -> feat/dict) Bb: print a
| * d7b794e Ba: create dict.py
|/
* 9bdc0f2 B: print dict
* a521e7c A: create main.py
```

</td><td>깃 별칭 만들기
logg1a</td></tr>
<tr><td rowspan="2">모듈5
실습
(10분)</td><td colspan="2">□ 이전 커밋 이력으로 head를 이동시키는 checkout
○ ^^, ~~, ~1</td></tr>
<tr><td>

```
$ git logg1
* 9f74e28 (HEAD -> main) D: print end
* 43791fe C: print comp
* 9bdc0f2 B: print dict
* a521e7c A: create main.py

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
$ git checkout head^^
Note: switching to 'head^^'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 9bdc0f2 B: print dict

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch
```

</td><td>HEAD: 현재 브랜치의 마지막 커밋을 가지키는 포인터, checkout으로 이동 가능</td></tr>
</table>

	((9bdc0f2...))
	$ git logg1
	* 9bdc0f2 (HEAD) B: print dict
	* a521e7c A: create main.py
	
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch ((9bdc0f2...))
	$ git checkout
	
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch ((9bdc0f2...))
	$ git logg1
	* 9bdc0f2 (HEAD) B: print dict
	* a521e7c A: create main.py
	
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch ((9bdc0f2...))
	$ git checkout main
	Previous HEAD position was 9bdc0f2 B: print dict
	Switched to branch 'main'
	
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
	$ git logg1
	* 9f74e28 (HEAD -> main) D: print end
	* 43791fe C: print comp
	* 9bdc0f2 B: print dict
	* a521e7c A: create main.py	

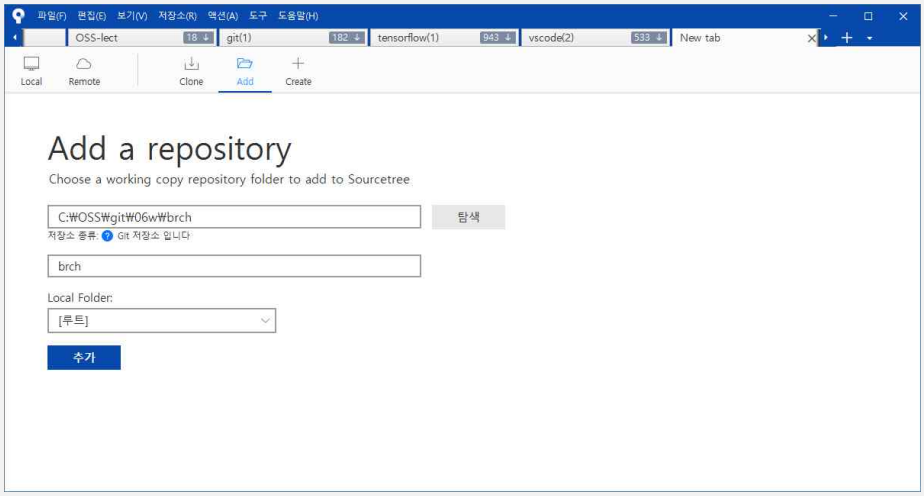
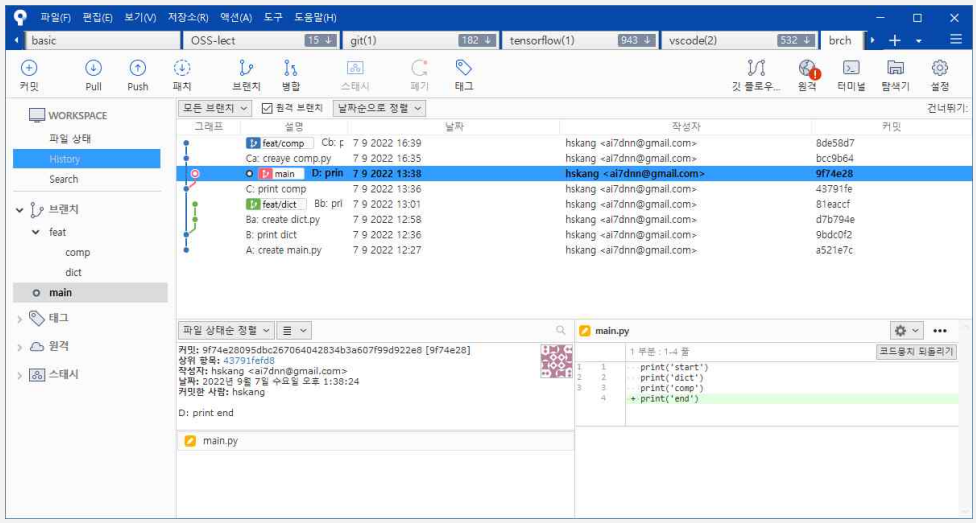
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
	$ git checkout head^
	Note: switching to 'head^'.
	
	You are in 'detached HEAD' state. You can look around, make experimental
	changes and commit them, and you can discard any commits you make in this
	state without impacting any branches by switching back to a branch.
	
	If you want to create a new branch to retain commits you create, you may
	do so (now or later) by using -c with the switch command. Example:
	
	git switch -c <new-branch-name>
	
	Or undo this operation with:
	
	git switch -
	
	Turn off this advice by setting config variable advice.detachedHead to false
	
	HEAD is now at 43791fe C: print comp
	
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch ((43791fe...))
	$ git logg1
	* 43791fe (HEAD) C: print comp
	* 9bdc0f2 B: print dict
	* a521e7c A: create main.py
	
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch ((43791fe...))
	$ cat main.py
	print('start')
	print('dict')
	print('comp')
	
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch ((43791fe...))
	$ git checkout head~2
	Previous HEAD position was 43791fe C: print comp
	HEAD is now at a521e7c A: create main.py
	
	PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch ((a521e7c...))
	$ git logg1
		head를 이전으로 이동하면 그 당시의 파일 내용 확인이 가능

```
* a521e7c (HEAD) A: create main.py

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch
((a521e7c...))
$ cat main.py
print('start')

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch
((a521e7c...))
$ git checkout main
Previous HEAD position was a521e7c A: create main.py
Switched to branch 'main'

PC@DESKTOP-482NOAB MINGW64 /c/OSS/git/06w/brch (main)
$ git logg1
* 9f74e28 (HEAD -> main) D: print end
* 43791fe C: print comp
* 9bdc0f2 B: print dict
* a521e7c A: create main.py
```

□ 소스트리로 열어서 브랜치 확인

○ new tab에서 메뉴 add 선택