

```
doGit.pragmatic({  
    eclipse : egit,  
    gitCore : commands  
});
```

프로젝트에서 Egit과 git command로
깃 실용적으로 사용하기.

기본지식

기본지식

버전관리시스템
(Version control system, VCS)

기본지식

- * 중앙집중형 버전관리
(CentralizedVCS)
- * 분산 버전관리
(DistributedVCS)

기본 지식

사용자가 변경한 모든 내용을 추적하는 공간

* 저장소
(Repository)

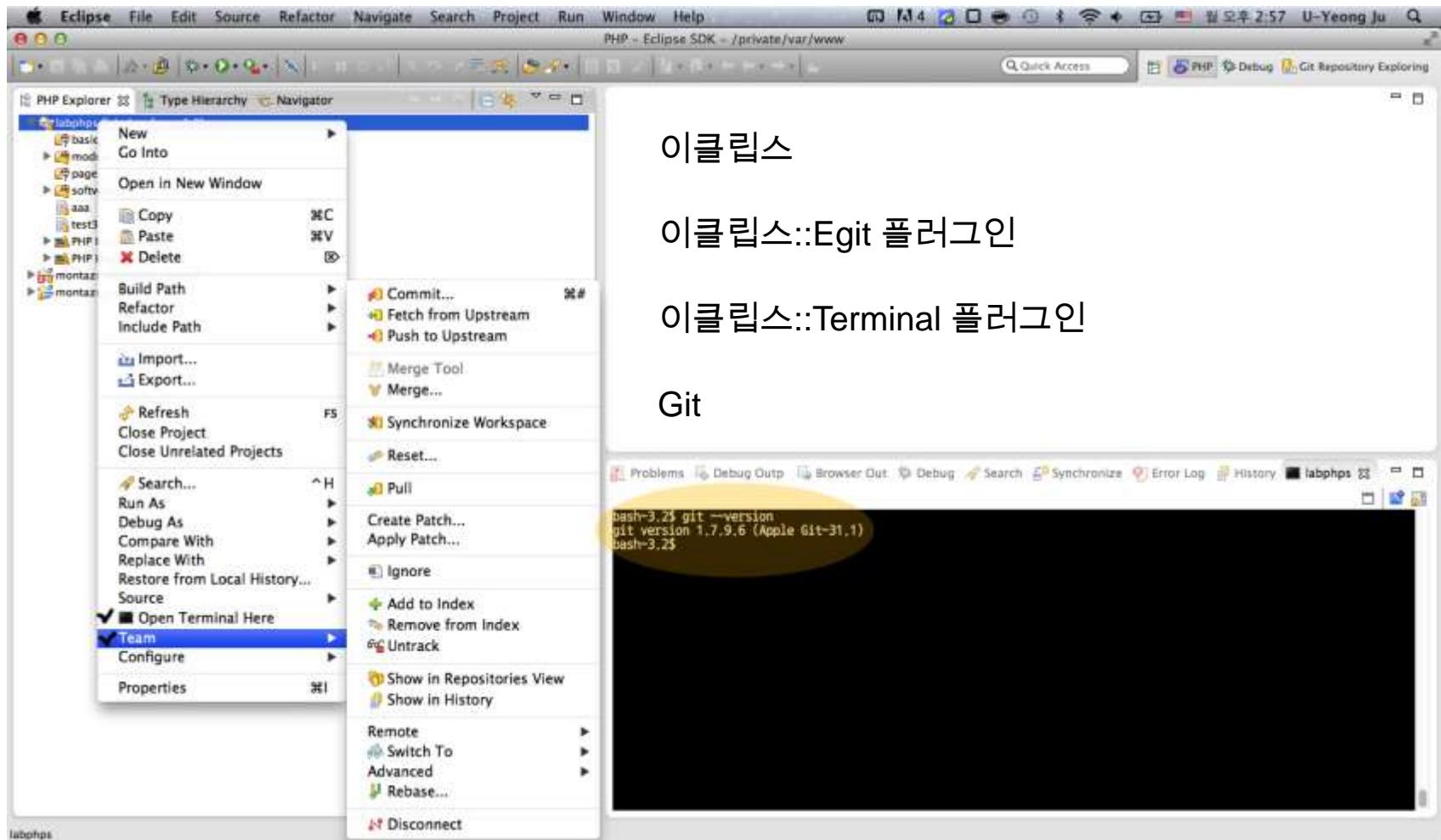
기본지식

모든 변경이 이루어지는 공간,
저장소를 바라보는 자신의 현재 시점

* 작업트리
(Working Tree)

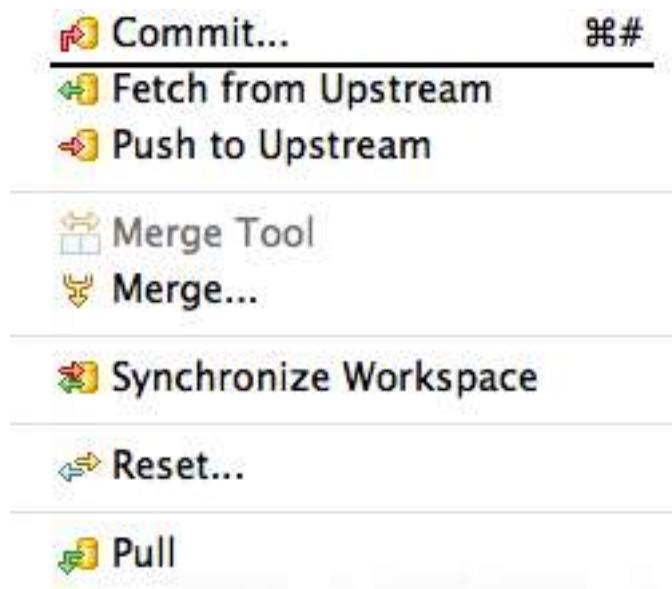
G I T

사용하는 프로그램



GIT

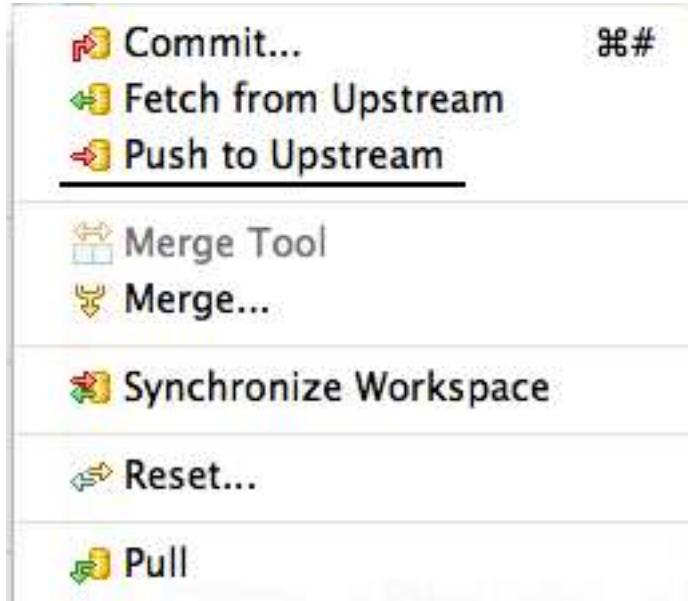
지역저장소에 변경사항을 적용하기 위해 새로운 리비전을 추가하고 무엇을 변경했는지 설명하는 로그메시지 까지 저장하는 기능



* 커밋
(Commit)

GIT

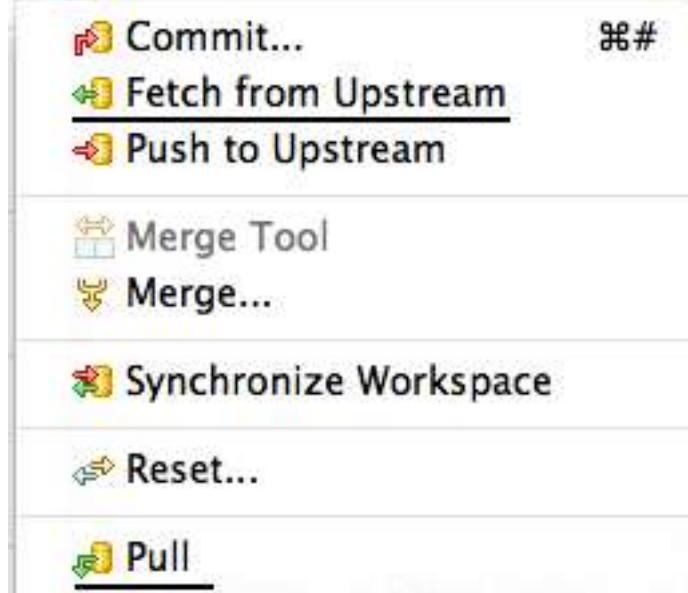
상위저장소(upstream repository)에 변경사항을 적용하는 기능, 상위저장소는 원격저장소라고 부른다.



* 푸시
(Push)

GIT

중앙집중식의 checkout와는 다른 동작을 보여준다.
풀링은 두 단계를 거친다. 상위저장소의
변경사항을 복사하고, 변경이력을 지역저장소와
병합한다.



* 풀

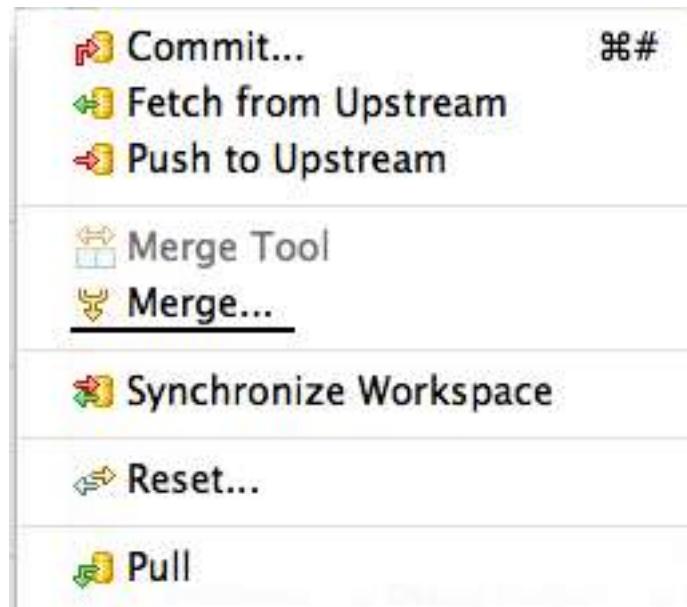
(Pull)

! Fetch

fetch는 pull과는 다르다.
원격저장소의 변경사항과 지역
저장소의 이력과 병합하기 전
까지만 하는 단계를 말한다.

GIT

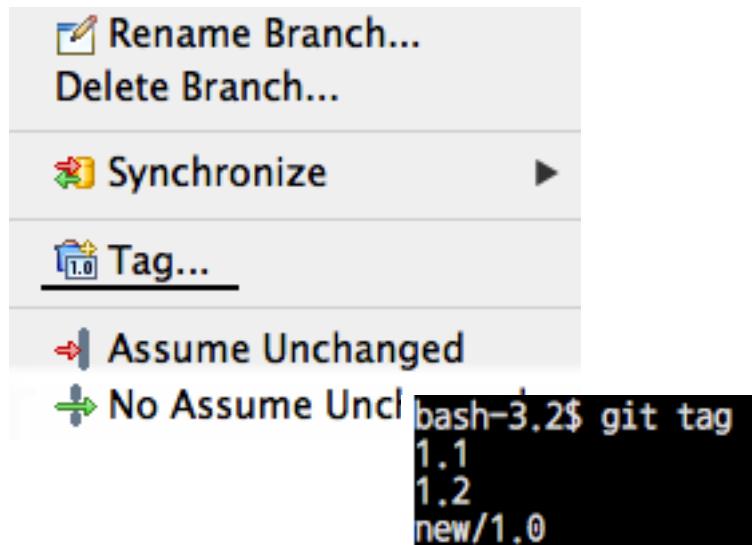
브랜치 이력을 하나로 합친다.



* 병합
(Merge)

GIT

저장소 이력의 특정위치를 기록하는 기능, 책갈피



* 태그
(Tag)

GIT

Commit객체를 가리키는 포인터, 하나의 단위

```
bash-3.2$ git branch
* from-1.0
  master
  new
```

* 브랜치
(Branch)

GIT

가장 최신의 리비전을 가리키는 포인터

```
[aspectj@U-Yeong-ui-MacBook-Pro .git]$ pwd  
/var/www/montazu/.git  
[aspectj@U-Yeong-ui-MacBook-Pro .git]$ ls -al  
total 112  
drwxr-xr-x  11 aspectj  staff   374 10 23 00:27 ./  
drwxr-xr-x  14 aspectj  staff   476 10 18 01:35 ../  
-rw-r--r--  1 aspectj  staff   126 10 18 05:02 FETCH_HEAD  
-rw-r--r--  1 aspectj  staff    23 10 23 00:27 HEAD  
drwxr-xr-x  2 aspectj  staff   68 10 18 01:34 branches/  
-rw-r--r--  1 aspectj  staff   262 10 18 01:35 config  
drwxr-xr-x  2 aspectj  staff   68 10 18 01:34 hooks/  
-rw-r--r--  1 aspectj  staff  44184 10 23 00:27 index  
drwxr-xr-x  4 aspectj  staff   136 10 18 01:35 logs/  
drwxr-xr-x 156 aspectj  staff  5304 10 22 18:03 objects/  
drwxr-xr-x   5 aspectj  staff   170 10 18 01:35 refs/  
[aspectj@U-Yeong-ui-MacBook-Pro .git]$ cat HEAD  
ref: refs/heads/master
```

* 헤드
(HEAD)

GIT의 디렉토리

GIT의 디렉토리

```
.git
├─ FETCH_HEAD
├─ HEAD
├─ ORIG_HEAD
├─ branches
├─ config
├─ description
├─ hooks
│   |-- applypatch-msg.sample
│   |-- commit-msg.sample
│   |-- post-update.sample
│   |-- pre-applypatch.sample
│   |-- pre-commit.sample
│   |-- pre-rebase.sample
│   |-- prepare-commit-msg.sample
│       '-- update.sample
├─ index
├─ info
│   '-- exclude
├─ logs
│   '-- HEAD
│       '-- refs
│           '-- heads
├─ objects
│   '-- info
│   '-- pack
├─ packed-refs
`-- refs
    '-- heads
    '-- remotes
        '-- origin
            '-- HEAD
    '-- tags
```

깃 저장소를 생성하면
생성되는 .git의 구조

GIT의 디렉토리

```
.git
|-- FETCH_HEAD
|-- HEAD
|-- ORIG_HEAD
|-- branches
|-- config
|-- description
|-- hooks
|   |-- applypatch-msg.sample
|   |-- commit-msg.sample
|   |-- post-update.sample
|   |-- pre-applypatch.sample
|   |-- pre-commit.sample
|   |-- pre-rebase.sample
|   |-- prepare-commit-msg.sample
|   '-- update.sample
|-- index
|-- info
|   '-- exclude
|-- logs
|   |-- HEAD
|   '-- refs
|       '-- heads
|-- objects
|   |-- info
|   '-- pack
|-- packed-refs
`-- refs
    '-- heads
    '-- remotes
        '-- origin
            '-- HEAD
    '-- tags
```

HEAD포인터는 파일로
존재한다.

```
root@ip-10-160-229-163:/var/www/montazu/.git# cat HEAD
ref: refs/heads/montazu-1.0.0
```

```
root@ip-10-160-229-163:/var/www/montazu/.git/refs/heads# pwd
/var/www/montazu/.git/refs/heads
root@ip-10-160-229-163:/var/www/montazu/.git/refs/heads# ls
montazu-1.0.0
```

```
root@ip-10-160-229-163:/var/www/montazu/.git/refs/heads# cat montazu-1.0.0
50f95d89a0ba59e8504c91c092f429e8227c793e
```

GIT의 디렉토리

```
.git
├── FETCH_HEAD
├── HEAD
├── ORIG_HEAD
├── branches
├── config
├── description
└── hooks
    ├── applypatch-msg.sample
    ├── commit-msg.sample
    ├── post-update.sample
    ├── pre-applypatch.sample
    ├── pre-commit.sample
    ├── pre-rebase.sample
    ├── prepare-commit-msg.sample
    └── update.sample
├── index
├── info
│   '-- exclude
├── logs
│   '-- HEAD
│   '-- refs
│       '-- heads
└── objects
    '-- info
    '-- pack
├── packed-refs
└── refs
    '-- heads
    '-- remotes
        '-- origin
            '-- HEAD
    '-- tags
```

헤더정보를 포함하여 zlib
로 압축하여 저장

모든파일은 SHA방식의
해시로 표현된다.

```
objects/
├── 08
|   '-- 85706729123e24ef93d075576a1dc48a3d1729
|       ....
|       ....
├── d6
|   '-- d8dd3367be8791ba8ea252ba7b43ba6f448d9c
├── f5
|   '-- 696abaa65742fb407226b70dd45b4cbbf9d2ba
└── info
    '-- pack
        '-- pack-bc....idx
        '-- pack-bc....pack
```

GIT의 디렉토리

```
.git
├── FETCH_HEAD
├── HEAD
├── ORIG_HEAD
├── branches
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-rebase.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── index
├── info
│   '-- exclude
├── logs
│   '-- HEAD
│   '-- refs
│       '-- heads
├── objects
│   '-- info
│   '-- pack
├── packed-refs
└── refs
    '-- heads
    '-- remotes
        '-- origin
            '-- HEAD
    '-- tags
```

원격저장소의 정보도 역시 파일로 저장된다.

```
bash-3.2$ pwd
/private/var/www/labphps/.git/refs/remotes/origin
bash-3.2$ cat master
233e717aad00643db8813483bcf21b43a68737c9
```

GIT전략

브랜치 사용

실험적인 변경사항

성능이 향상되는지 알아보기 위해 알고리즘을
다시 작성하거나 코드의 일부를 특정 패턴으로
리팩토링 하려 할때 브랜치를 생성하여 작업

브랜치 사용

새로운 기능

새로운 기능을 추가할때마다 브랜치를 생성한다.
작업이 끝나면 브랜치를 합쳐야한다. 이때
브랜치의 전체 변경이력을 가져오거나 이력을
하나의 커밋으로 합칠 수 있다.

브랜치 사용

버그수정

아직 릴리즈 하지 않은 코드의 버그거나 릴리즈 하려고 태그를 붙힌 코드의 버그에 상관없이 버그와 관련된 변경사항을 추적할 수 있도록 브랜치를 생성한다. 새로운 기능을 추가하는 브랜치와 마찬가지로 수정한 부분을 기존의 코드에 반영할 때 매우 유연해진다.

브랜치 머지의 종류

바로합치기(straight merge)

하나의 브랜치와 다른 브랜치의 변경 이력
전체를 합친다.

브랜치 머지의 종류

커밋합치기(squashed commit)

브랜치의 이력을 압축하여 다른 브랜치의 최신
커밋 하나로 합친다.

브랜치 머지의 종류

선택합치기(sherry picking)

다른 브랜치에서 하나의 커밋을 가져와 현재 브랜치에 적용한다.

릴리즈 브랜치

- 릴리즈 브랜치는 프로젝트에서 이번 릴리즈에 포함하기로한 기능 구현이 끝나면 생성한다.
- 릴리즈 브랜치에서는 최소한의 변경만 발생하며, 버그나 로직의 수정에만 집중할 뿐 새로운 기능을 추가하지 않는다.
- 일반적으로 릴리즈브랜치에는 RB_라는 접두어를 붙히며, 그뒤에 릴리즈 번호를 붙힌다.

릴리즈 브랜치

- 릴리즈브랜치는 해당 릴리즈가 요구하는 마지막 테스트까지 통과하는 짧은 기간 동안만 존재한다.
- 릴리즈준비가 완료되면, 해당 릴리즈를 표시하는 태그를 붙이고 브랜치를 삭제한다.
- 태그가 해당 위치를 표시하고 있으므로 이력을 유지하기 위해 브랜치를 유지할 필요가 없다.

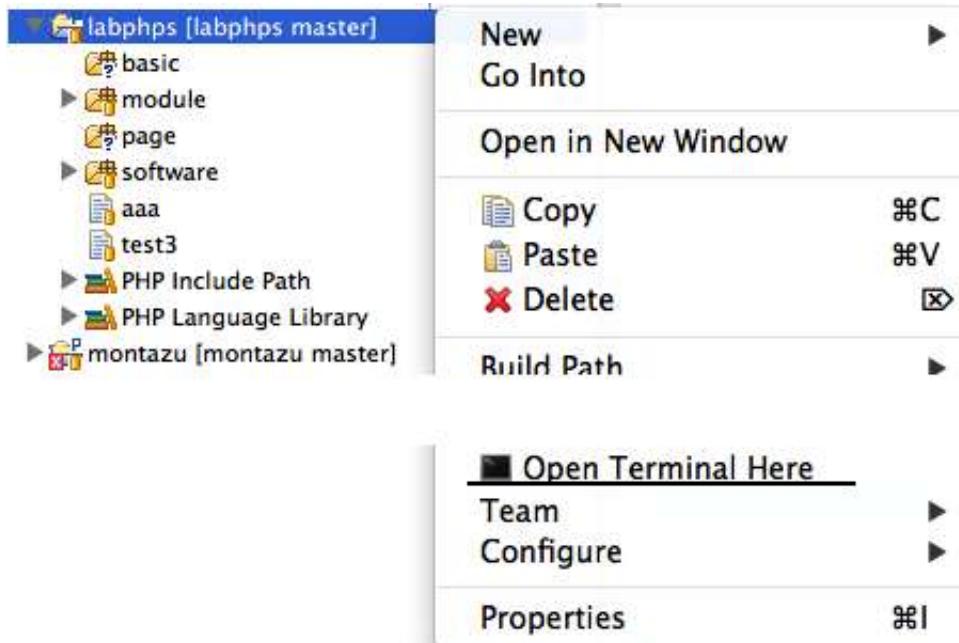
태그사용

프로젝트가 진행되면 마일스톤을 달성하게된다.
태그를 사용하면 마일스톤을 쉽게 표시할 수 있고
추후 원하는 마일스톤으로 돌아갈 수 있다. 태그는
포인터이며 저장소의 책갈피처럼 동작한다.

GIT시나리오 기능추가

GIT전략 시나리오 - 기능추가

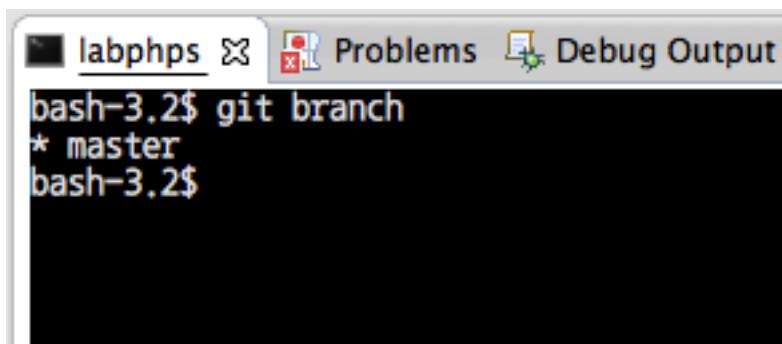
1. 브랜치 확인



터미널을 연다.

GIT전략 시나리오 - 기능추가

1. 브랜치 확인



A screenshot of a terminal window titled "labphps". The window has three tabs at the top: "Problems" (highlighted in red) and "Debug Output". The main area shows the command "git branch" being run in a bash shell. The output indicates that the current branch is "master".

```
labphps ✘ Problems Debug Output
bash-3.2$ git branch
* master
bash-3.2$
```

Git branch 명령어로
브랜치를 확인한다.

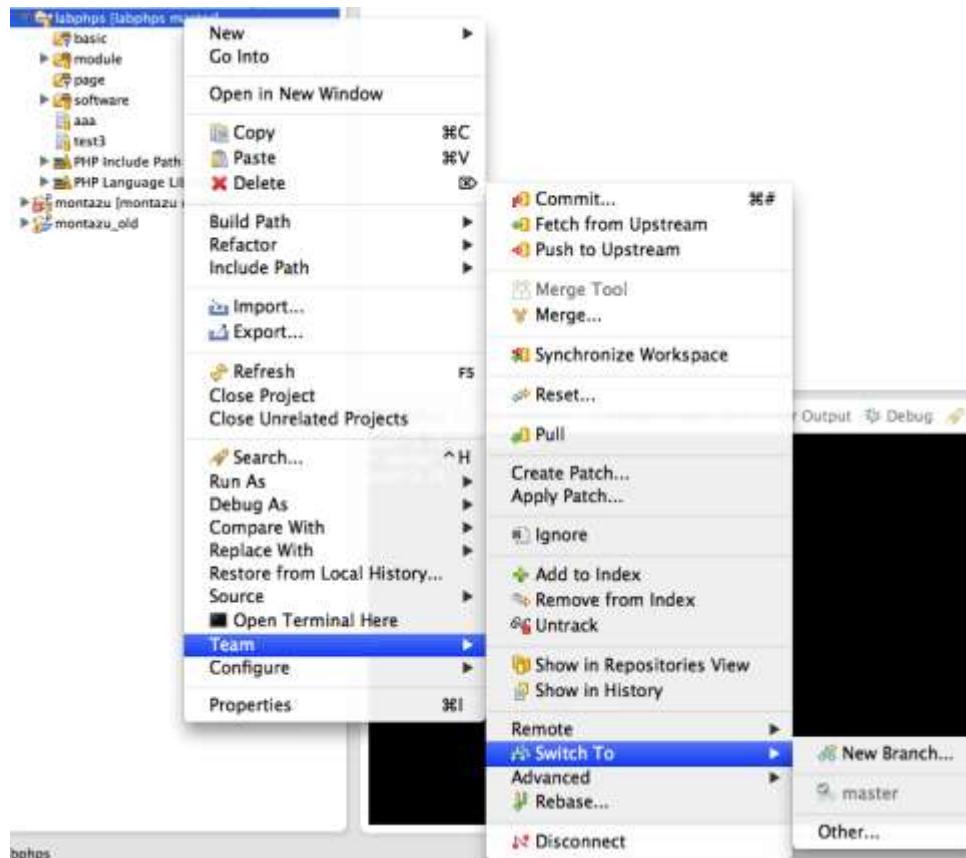
GIT전략 시나리오 - 기능추가

1. 브랜치 확인



GIT전략 시나리오 - 기능추가

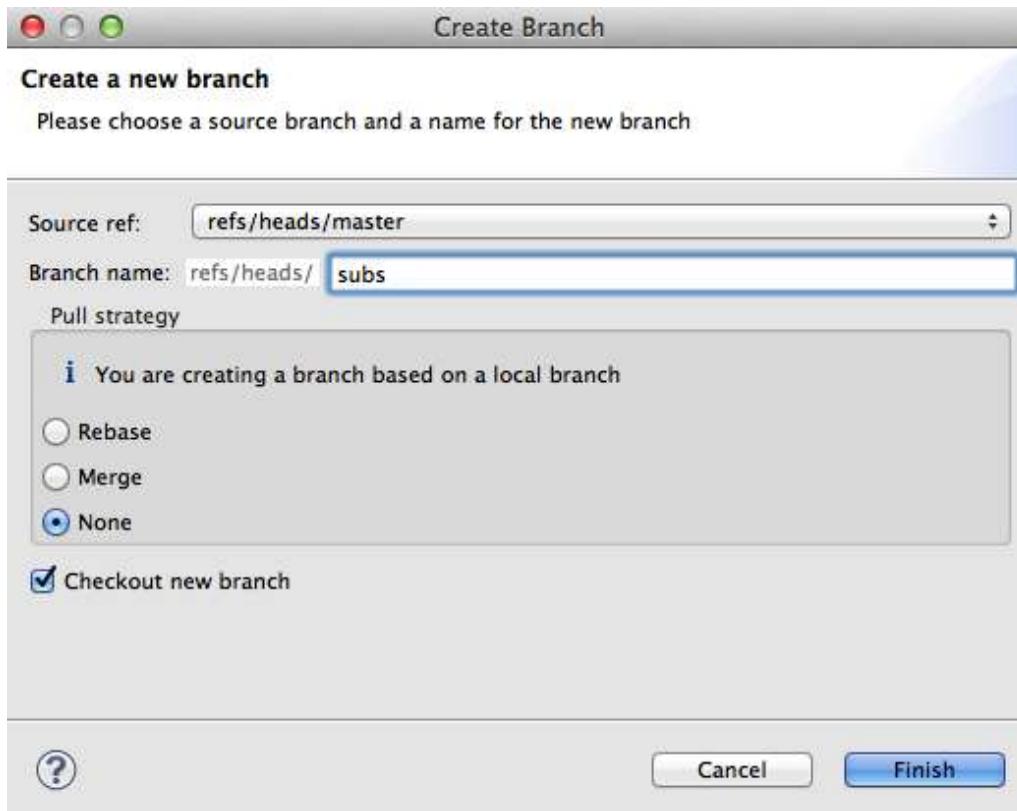
2. 브랜치 생성



New branch를
클릭한다.

GIT전략 시나리오 - 기능추가

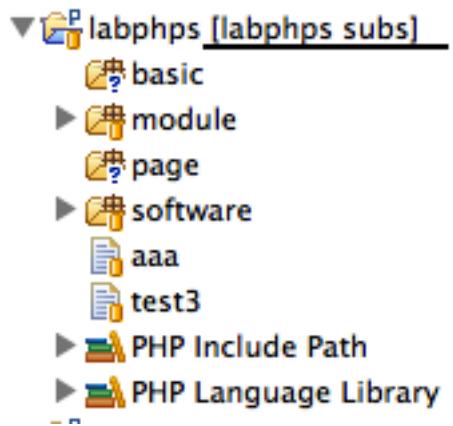
2. 브랜치 생성



Subs라는
브랜치를
생성한다.

GIT전략 시나리오 - 기능추가

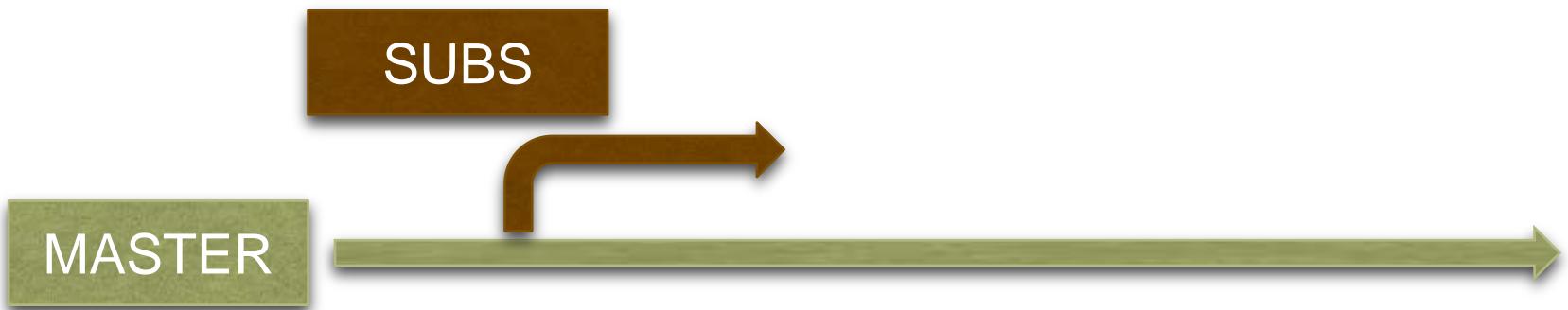
2. 브랜치 생성



포인터를 확인한다.

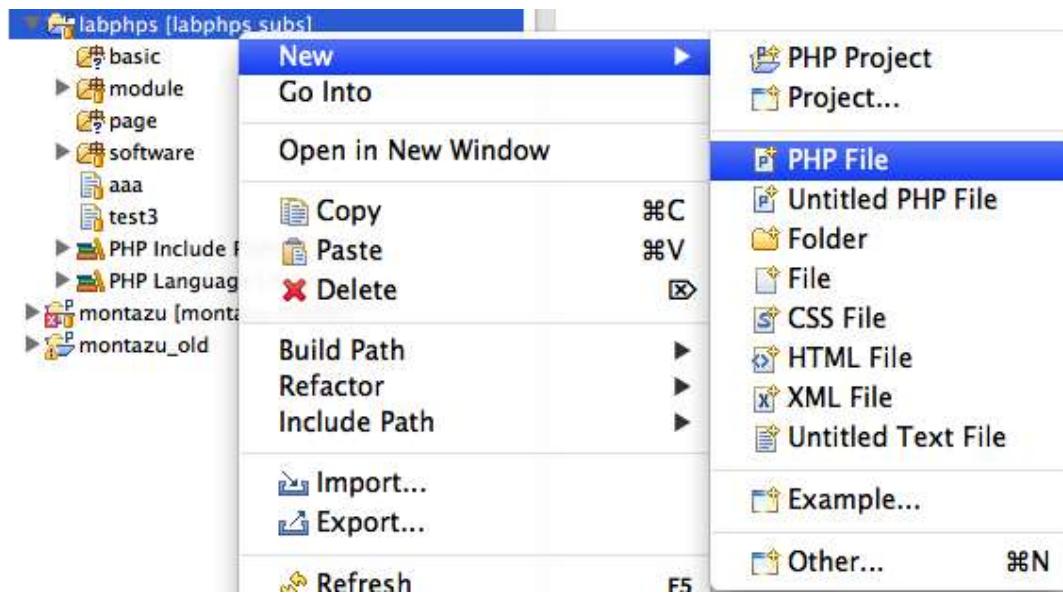
GIT전략 시나리오 - 기능추가

2. 브랜치 생성



GIT전략 시나리오 - 기능추가

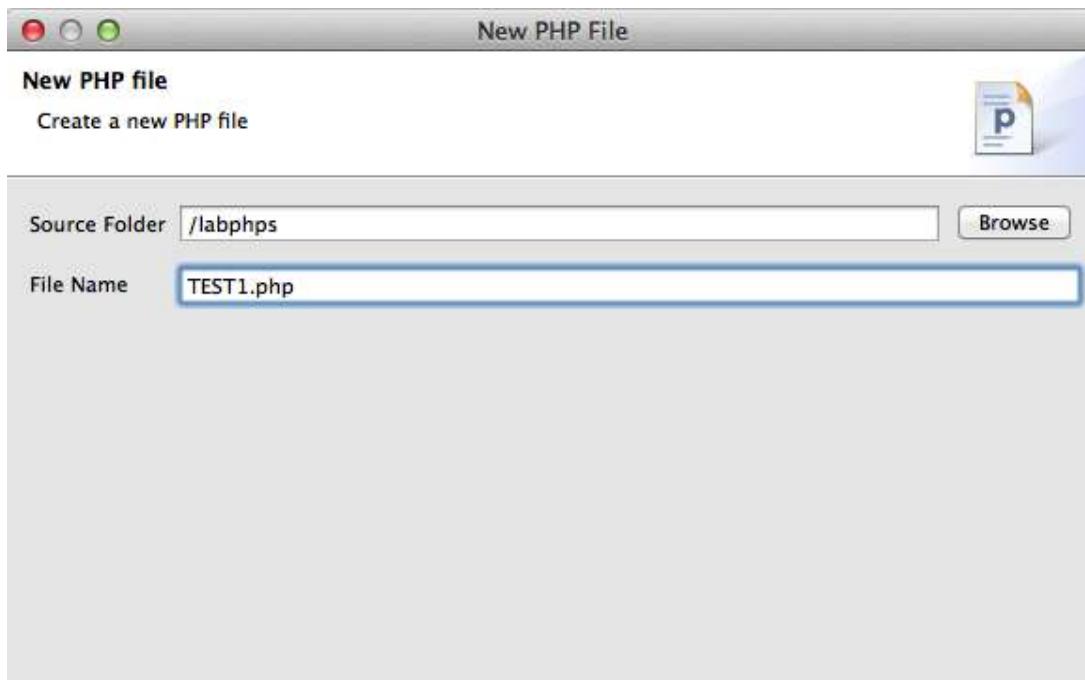
3. 파일추가 후 커밋



파일을 추가한다.

GIT전략 시나리오 - 기능추가

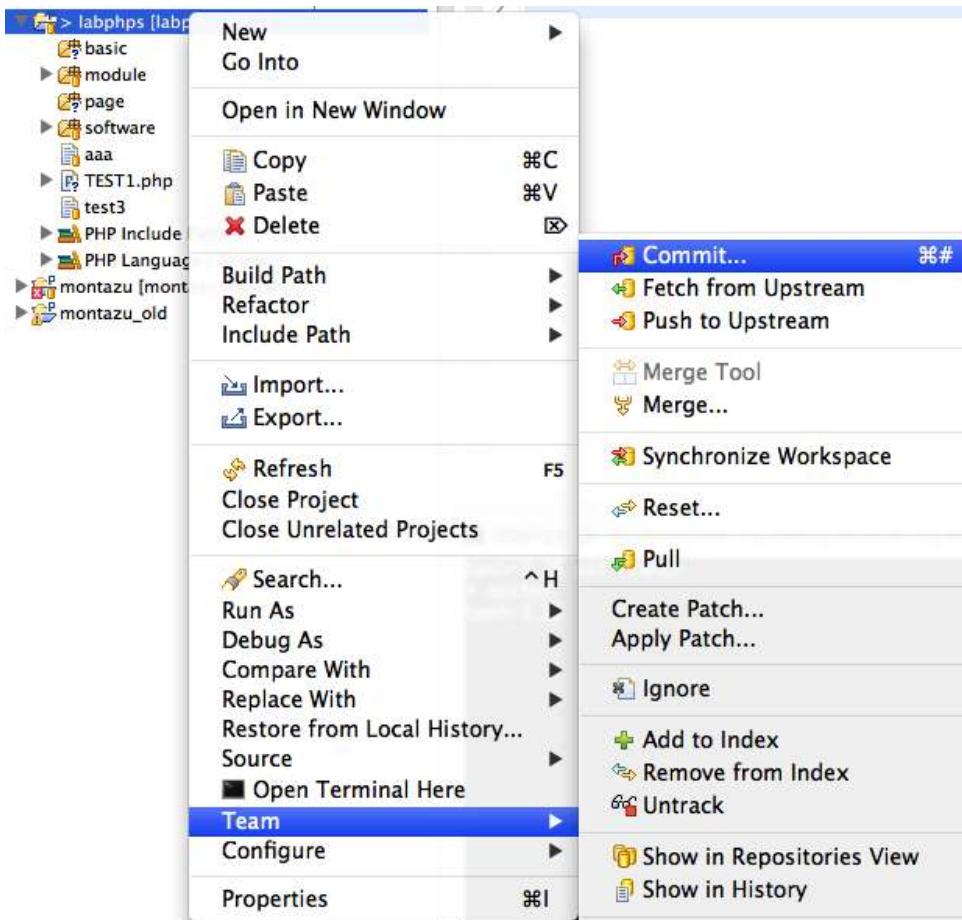
3. 파일추가 후 커밋



TEST1.PHP파일
을 추가해 본다.

GIT전략 시나리오 - 기능추가

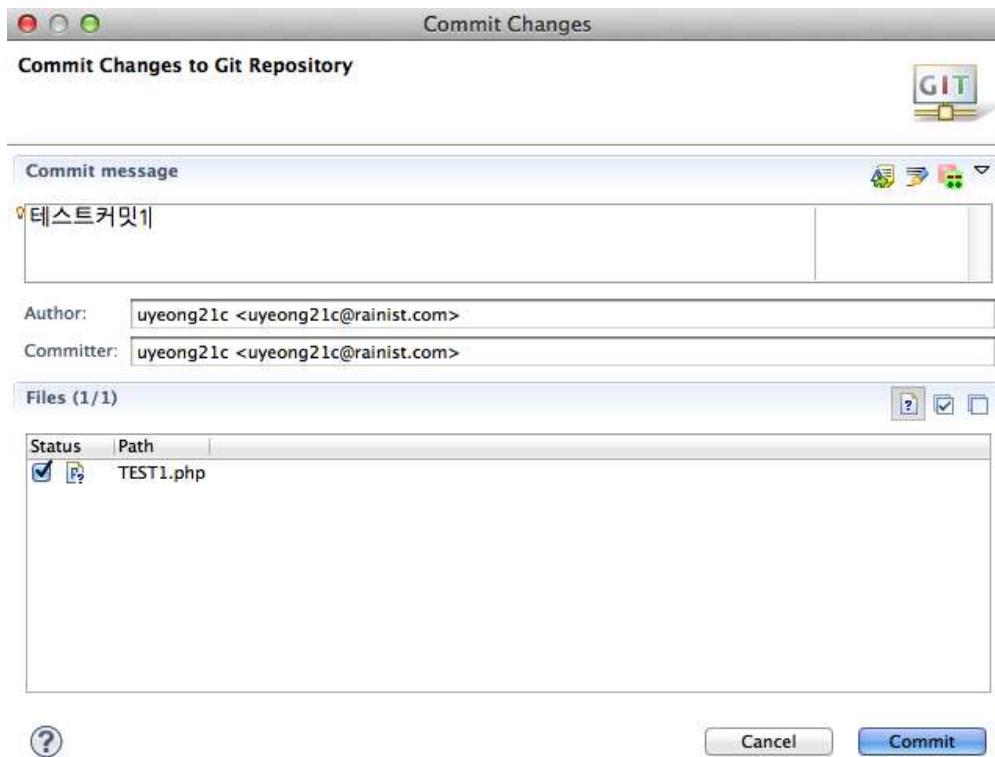
3. 파일추가 후 커밋



파일 추가가
다되었으면
커밋한다.

GIT전략 시나리오 - 기능추가

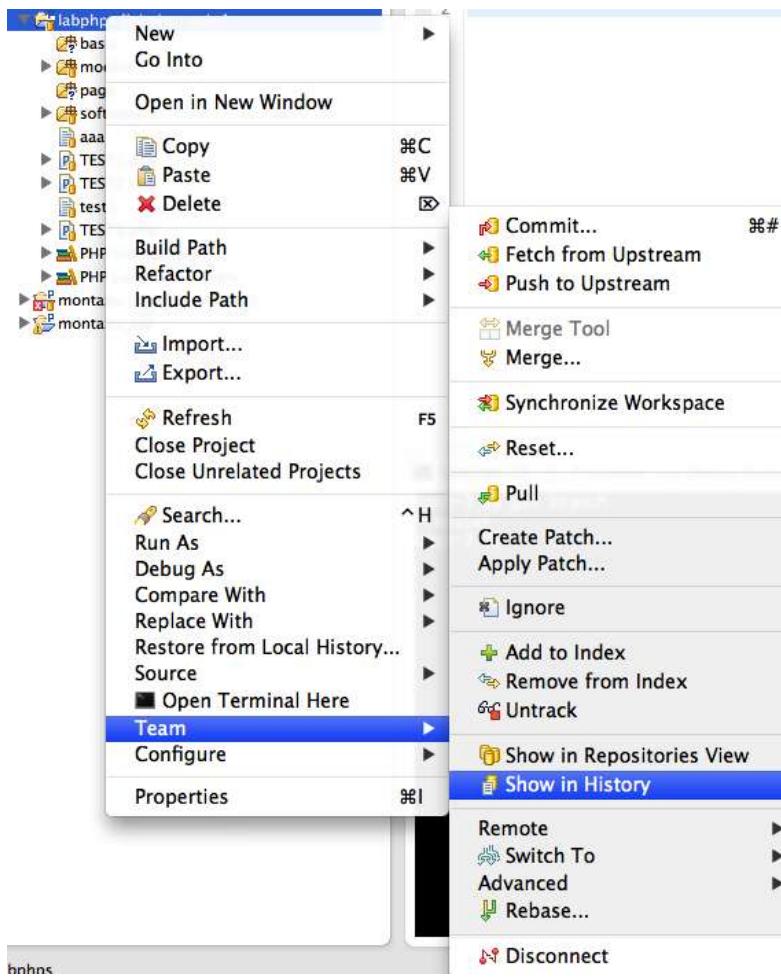
3. 파일추가 후 커밋



파일 추가가
다되었으면
커밋한다.

GIT전략 시나리오 - 기능추가

3. 파일추가 후 커밋



커밋 리스트를 확인해보기 위해 history를 켠다.

GIT전략 시나리오 - 기능추가

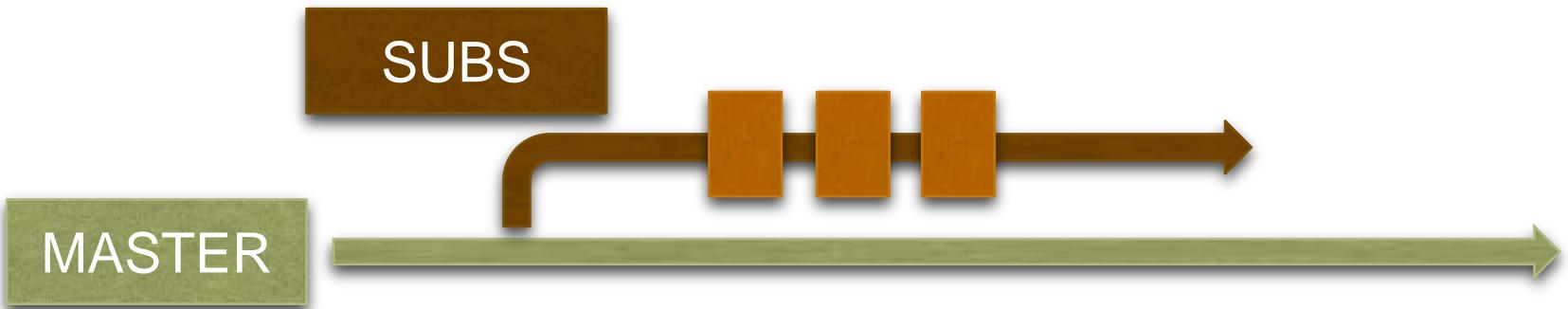
3. 파일추가 후 커밋

Project: labphps [labphps]	
Id	Message
198fa0e	subs [HEAD] 테스트커밋3
9e89abf	테스트커밋2
dfe0df0	테스트커밋1

커밋리스트가 3개 추가된 것을 확인한다.

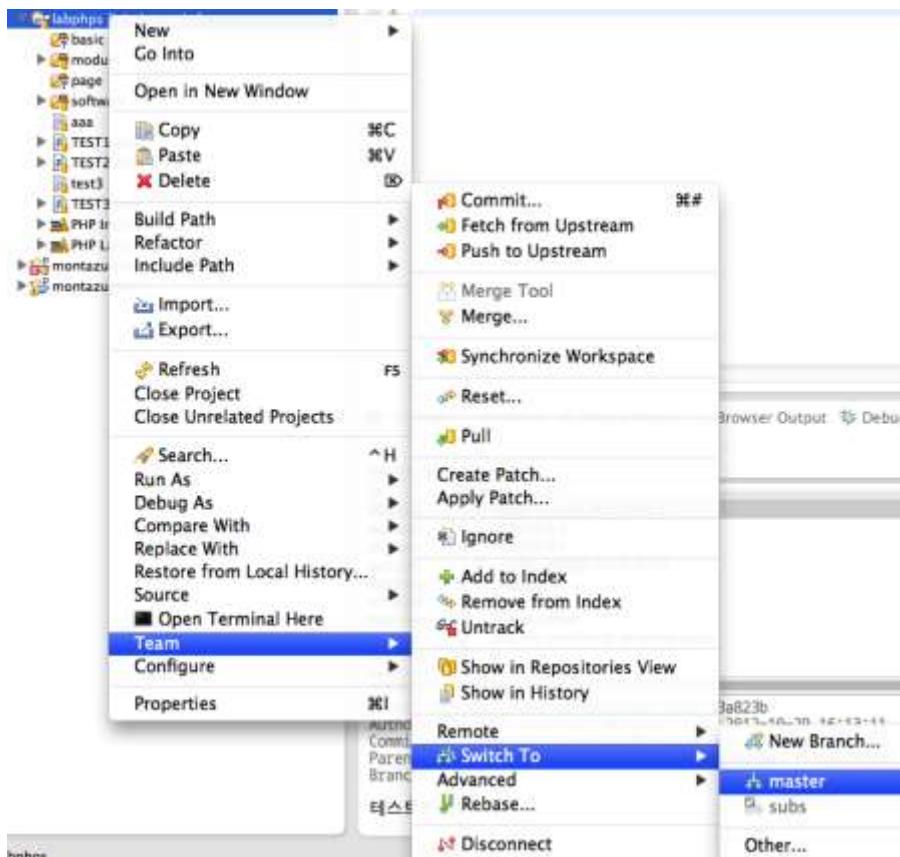
GIT전략 시나리오 - 기능추가

3. 파일추가 후 커밋



GIT전략 시나리오 - 기능추가

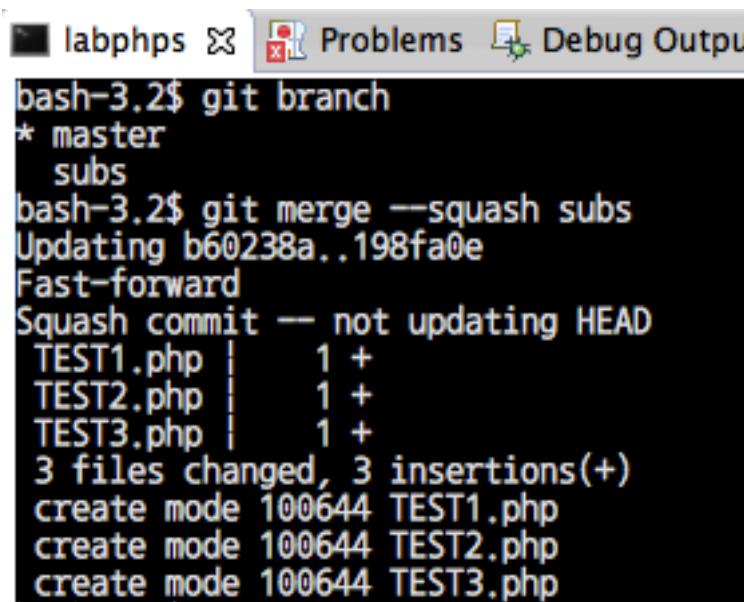
4. 커밋합치기



Master 브랜치로
포인터 이동

GIT전략 시나리오 - 기능추가

4. 커밋합치기



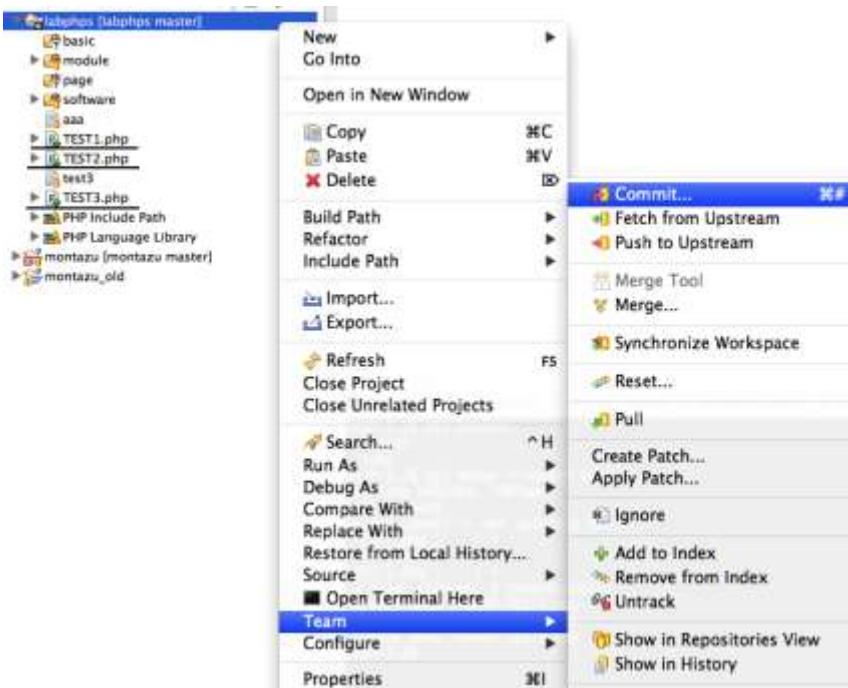
The screenshot shows a terminal window with the title bar "labphps" and tabs "Problems" and "Debug Output". The terminal content is as follows:

```
bash-3.2$ git branch
* master
  subs
bash-3.2$ git merge --squash subs
Updating b60238a..198fa0e
Fast-forward
  Squash commit — not updating HEAD
    TEST1.php      1 +
    TEST2.php      1 +
    TEST3.php      1 +
  3 files changed, 3 insertions(+)
  create mode 100644 TEST1.php
  create mode 100644 TEST2.php
  create mode 100644 TEST3.php
```

Squash 전달인자로
subs브랜치로부터 병합을
진행한다.

GIT전략 시나리오 - 기능추가

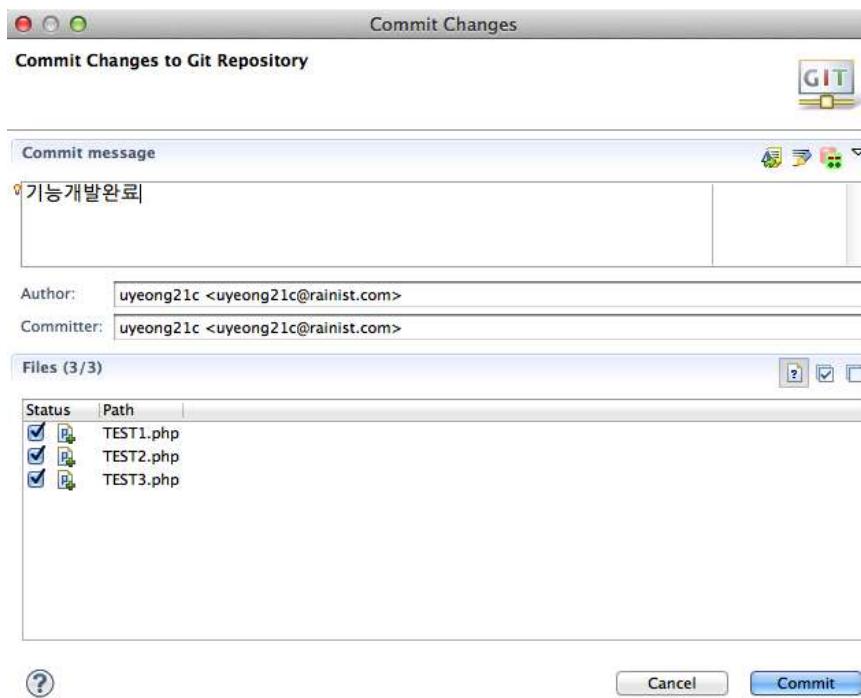
4. 커밋합치기



Subs 브랜치에 있는
커밋이 스테이징 되었지만
커밋은 되지 않은
상태이므로 커밋을
진행한다.

GIT전략 시나리오 - 기능추가

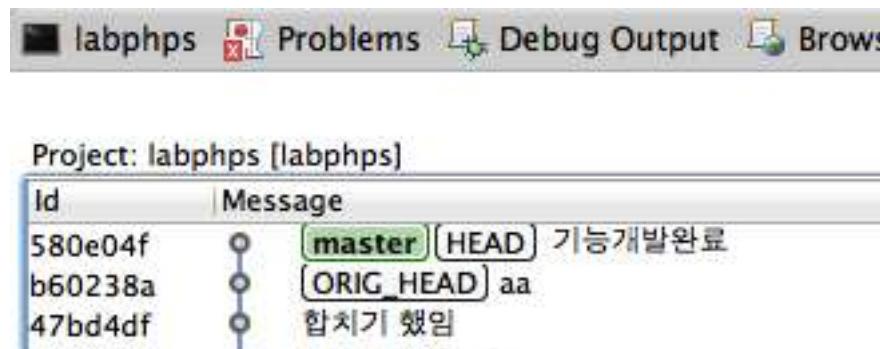
4. 커밋합치기



Subs 브랜치에 있는
커밋이 스테이징 되었지만
커밋은 되지 않은
상태이므로 커밋을
진행한다.

GIT전략 시나리오 - 기능추가

4. 커밋합치기

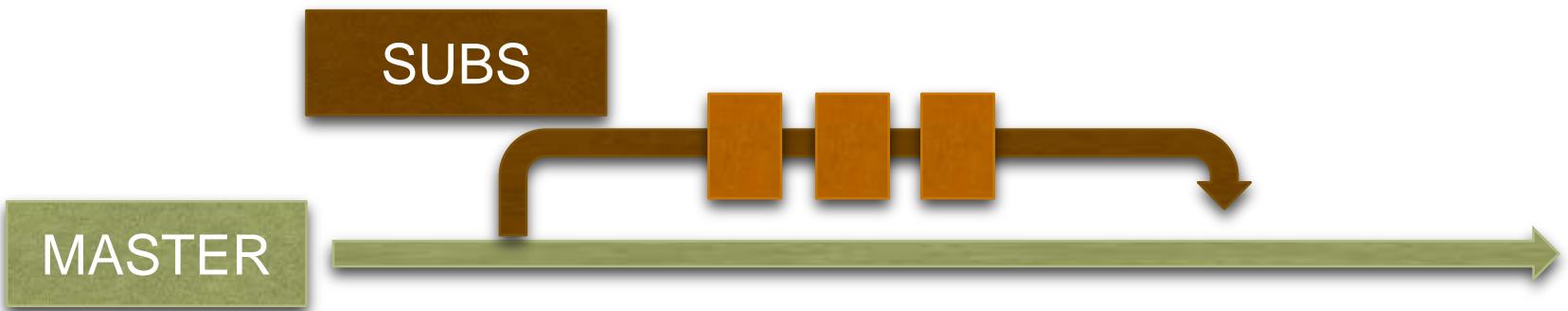


Id	Message
580e04f	master [HEAD] 기능개발완료
b60238a	ORIG_HEAD aa
47bd4df	합치기 했임

커밋이력을 확인해본다.

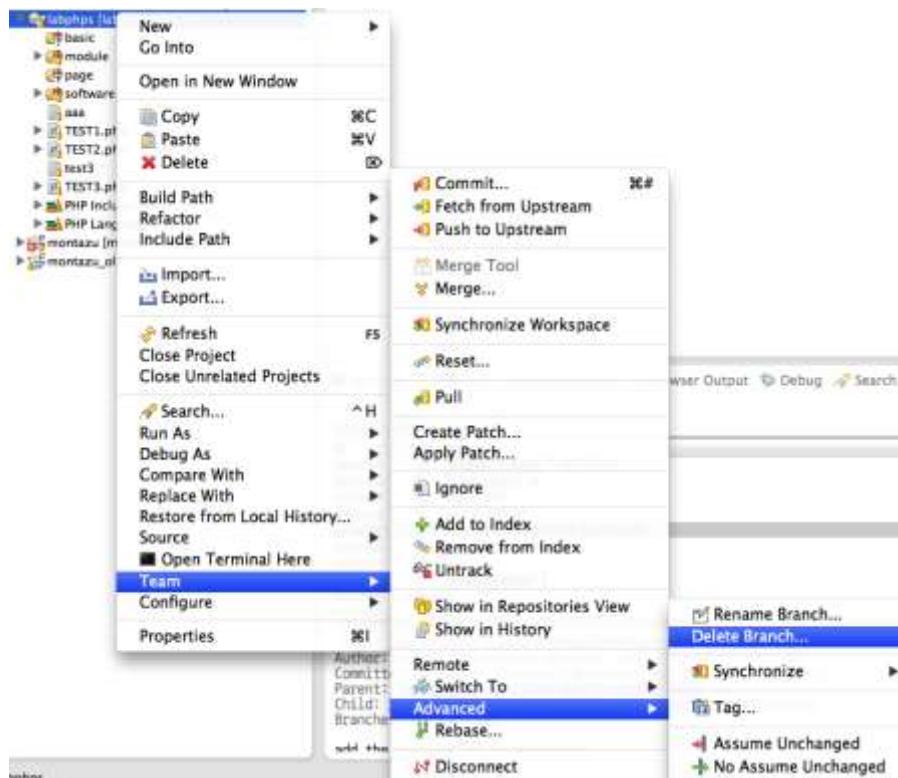
GIT전략 시나리오 - 기능추가

4. 파일추가 후 커밋



GIT전략 시나리오 - 기능추가

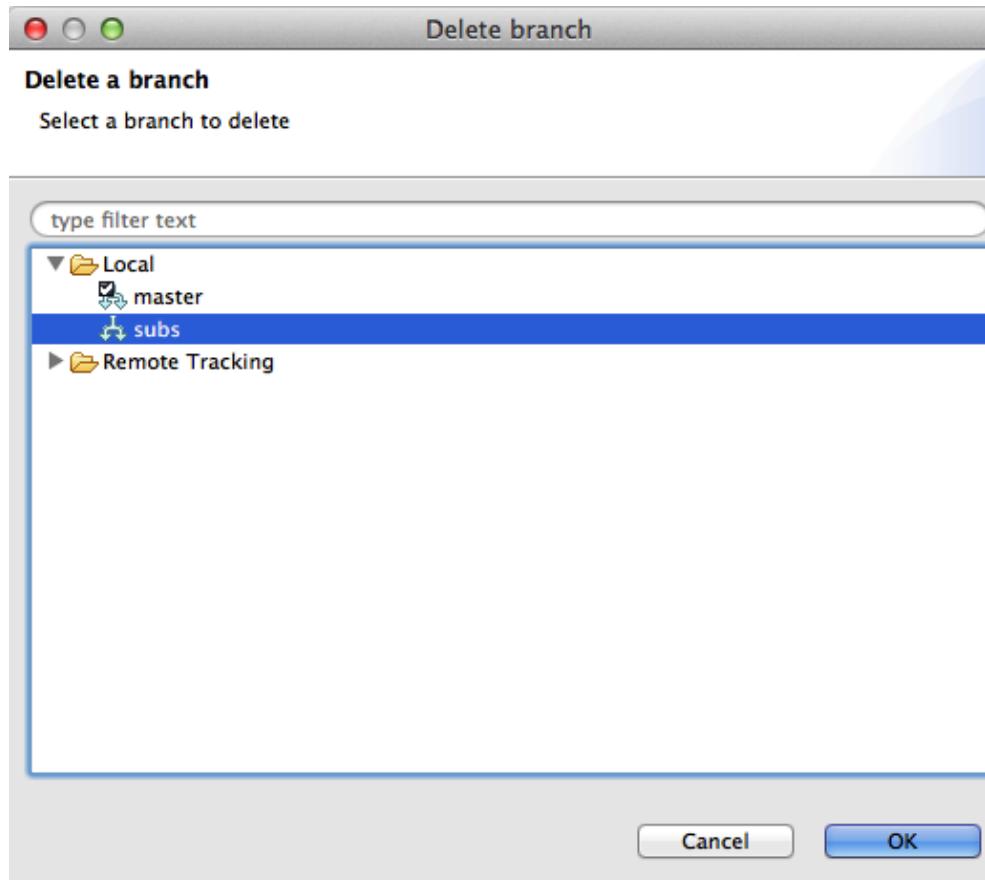
5. 기능추가브랜치 삭제



이 클립스의 브랜치 삭제
기능을 이용한다.

GIT전략 시나리오 - 기능추가

5. 기능추가브랜치 삭제



삭제한다.

GIT전략 시나리오 – 기능추가(완)

5. 기능추가 브랜치 삭제

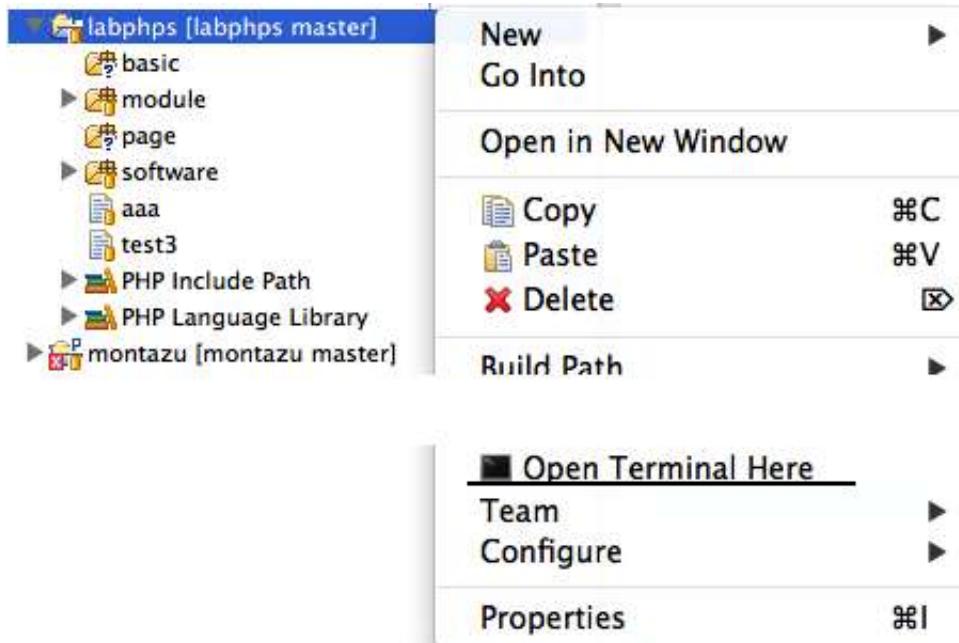
MASTER



GIT시나리오 버그수정

GIT전략 시나리오 - 버그수정

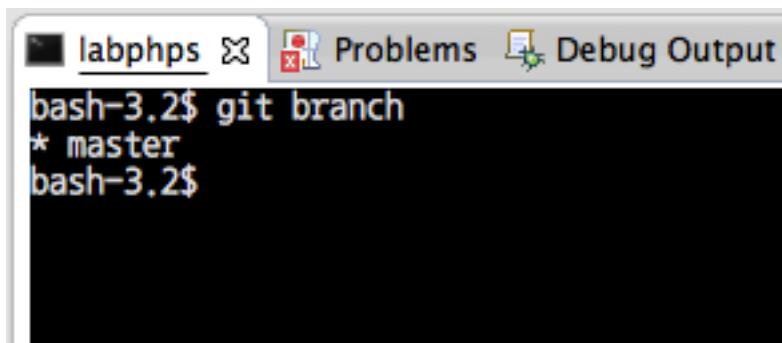
1. 브랜치 확인



터미널을 연다.

GIT전략 시나리오 - 버그수정

1. 브랜치 확인



A screenshot of a terminal window titled "labphps". The window has three tabs at the top: "Problems" (highlighted in red) and "Debug Output". The main area shows the command "git branch" being run in a bash shell. The output indicates that the current branch is "master".

```
labphps ✘ Problems Debug Output
bash-3.2$ git branch
* master
bash-3.2$
```

Git branch 명령어로
브랜치를 확인한다.

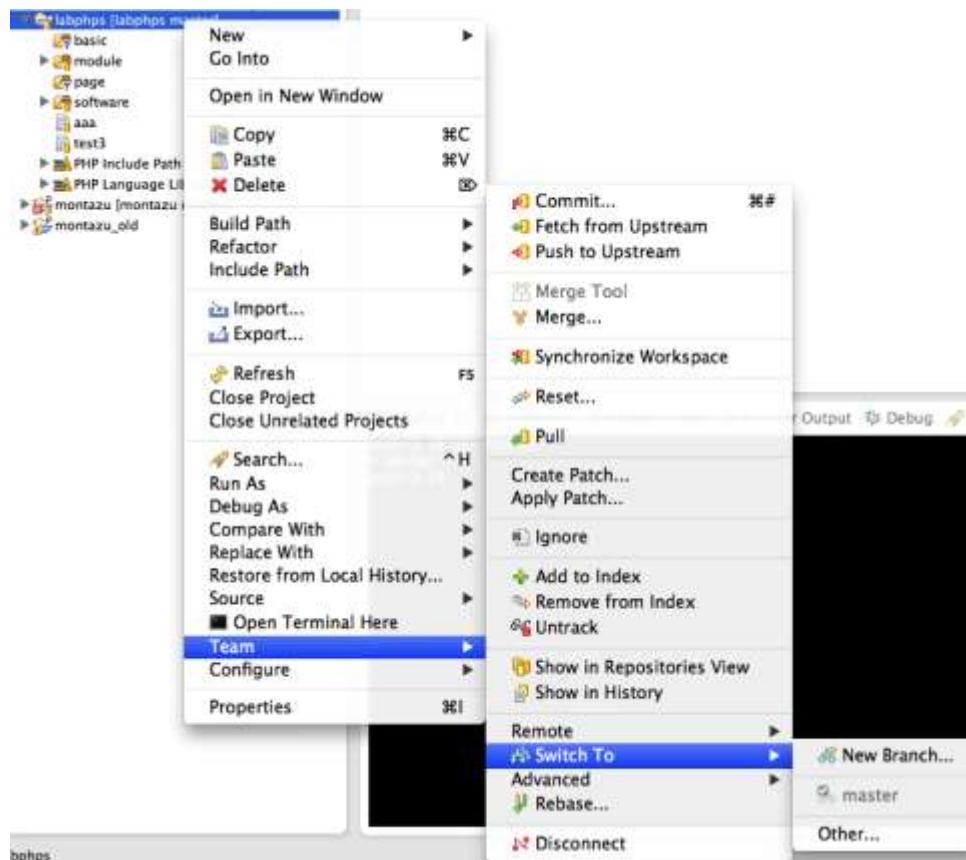
GIT전략 시나리오 - 버그수정

1. 브랜치 확인



GIT전략 시나리오 - 버그수정

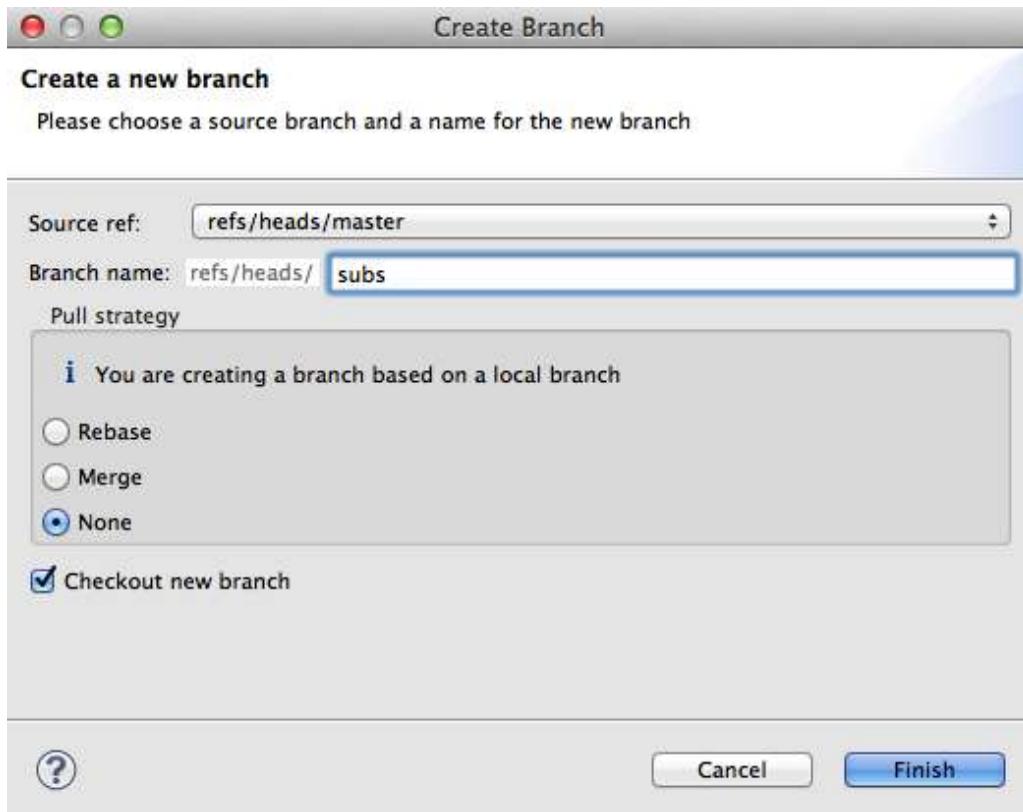
2. 브랜치 생성



New branch를
클릭한다.

GIT전략 시나리오 - 버그수정

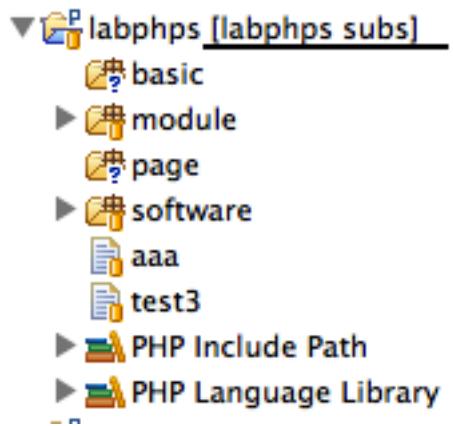
2. 브랜치 생성



Subs라는
브랜치를
생성한다.

GIT전략 시나리오 - 버그수정

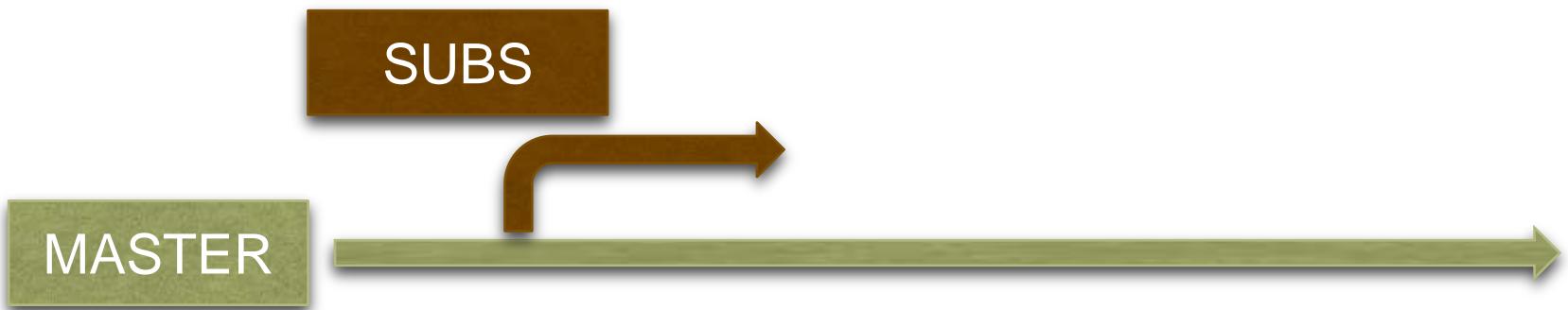
2. 브랜치 생성



포인터를 확인한다.

GIT전략 시나리오 - 버그수정

2. 브랜치 생성



GIT전략 시나리오 - 버그수정

3. 파일수정

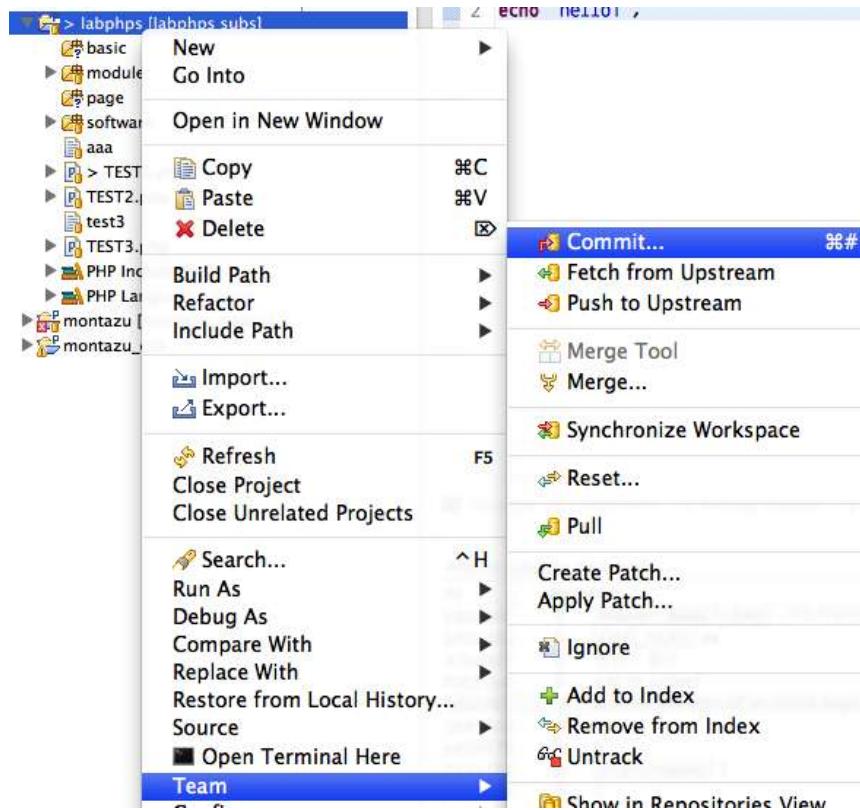
The screenshot shows a PHP code editor interface. On the left, there is a sidebar with a tree view of project files. The tree includes a root folder 'labphps' containing subfolders 'basic', 'module', 'page', 'software', and 'aaa'. Below these are several PHP files: 'TEST1.php', 'TEST2.php', 'test3', 'TEST3.php', 'PHP Include Path', 'PHP Language Library', 'montazu' (containing 'montazu master'), and 'montazu_old'. The 'TEST1.php' file is currently selected and open in the main editor area. The code in the editor is:

```
<?php  
echo "hello1";
```

TEST1.php 파일을 열어
echo hello문구를
추가하여 파일 수정을
한다.

GIT전략 시나리오 - 버그수정

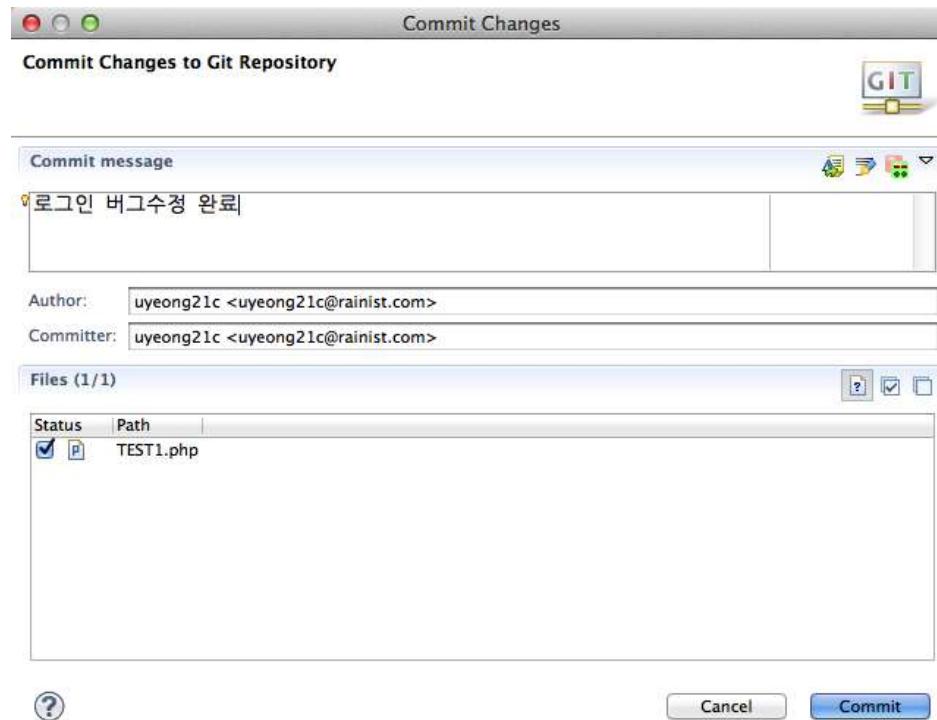
3. 파일수정



커밋을 진행한다.

GIT전략 시나리오 - 버그수정

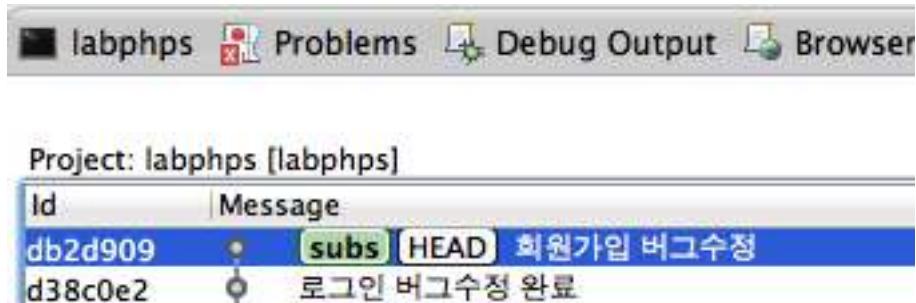
3. 파일수정



커밋을 진행한다.

GIT전략 시나리오 - 버그수정

3. 파일수정

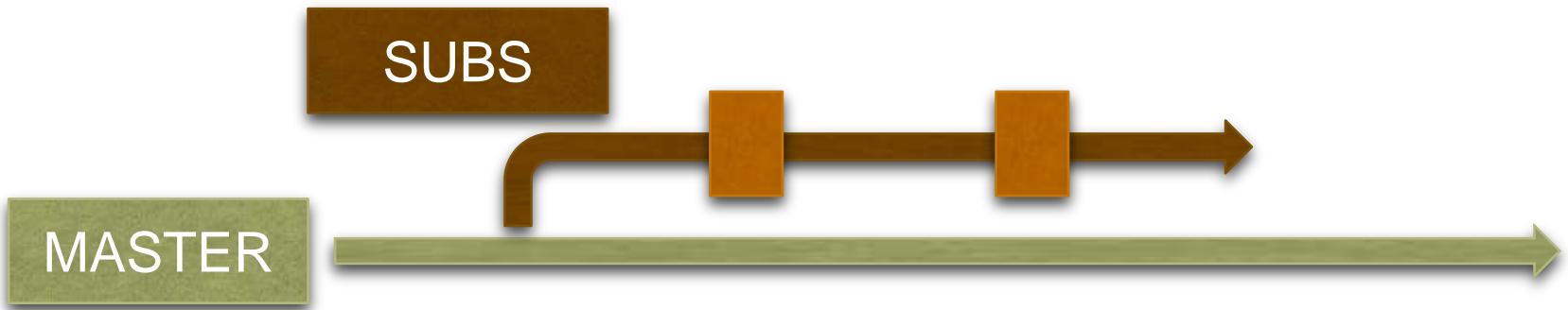


Id	Message
db2d909	subs HEAD 회원가입 버그수정
d38c0e2	로그인 버그수정 완료

커밋이력을 확인한다.

GIT전략 시나리오 - 기능추가

3. 버그수정



GIT전략 시나리오 - 버그수정

4. 선택합치기

버그수정 이력중에서 몇몇 가지만 적용하고
싶을지도 모른다. 많은 버그수정 이력중 검증받은
버그수정은 1개밖에 없을지도 모르기 때문이다.

여기서는 로그인 버그수정 완료 라는 이력만
선택하여 병합시키겠다.

GIT전략 시나리오 - 버그수정

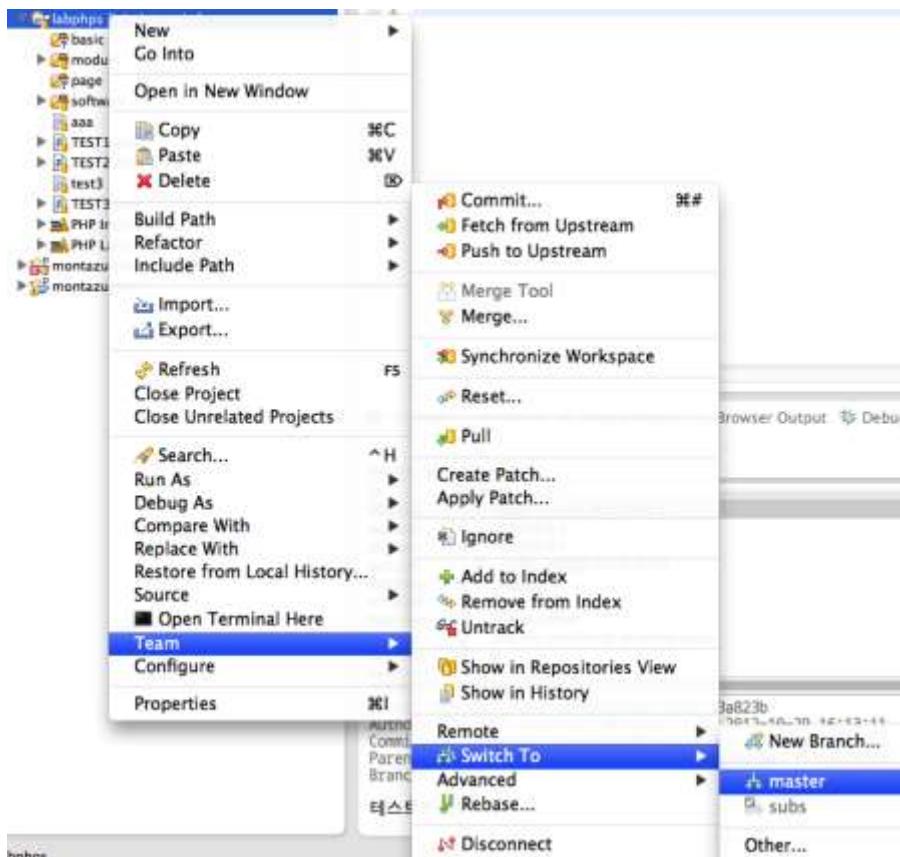
4. 선택합치기

Id	Message
db2d909	[subs] (HEAD) 회원가입 버그수정
d38c0e2	로그인 버그수정 완료
580e04f	[master] 기능개발완료

커밋하려는 버그수정한
커밋이력의 id를
기억한다.

GIT전략 시나리오 - 버그수정

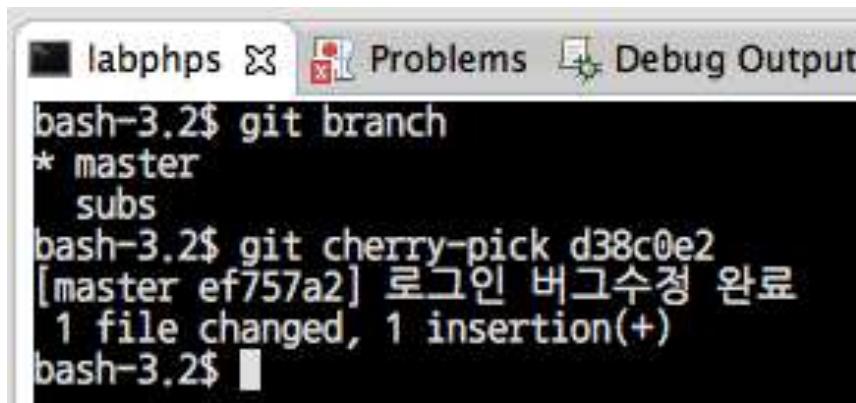
4. 선택합치기



Master 브랜치로
포인터 이동

GIT전략 시나리오 - 버그수정

4. 선택합치기



```
labphps labphps bash-3.2$ git branch
* master
  subs
labphps labphps bash-3.2$ git cherry-pick d38c0e2
[master ef757a2] 로그인 버그수정 완료
 1 file changed, 1 insertion(+)
labphps labphps bash-3.2$
```

커밋이력 고유id를 활용하여 master에 선택병합 이 된 것을 확인할 수 있다.

GIT전략 시나리오 - 버그수정

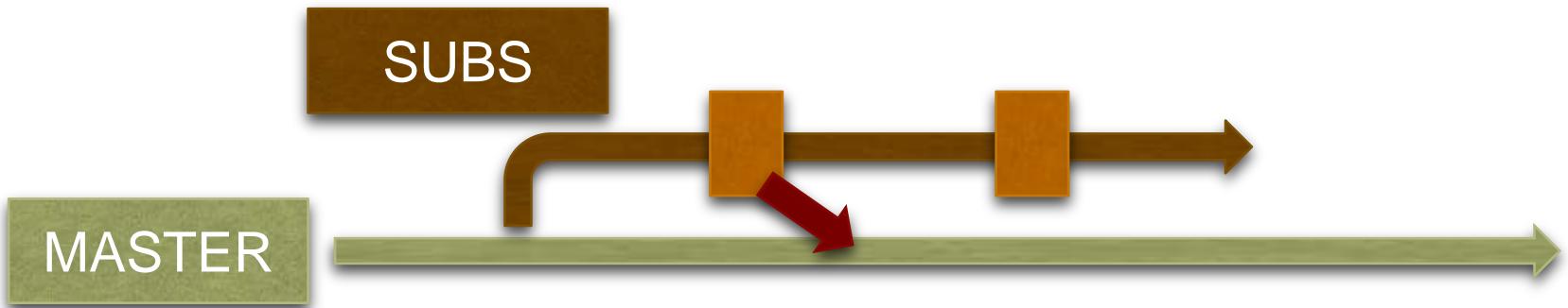
4. 선택합치기

Id	Message
ef757a2	[master] HEAD 로그인 버그수정 완료
580e04f	기능개발완료
b60238a	[ORIG_HEAD] aa
47bd4df	합치기 했임

커밋이력 고유id를 활용하여 master에 선택병합 이 된 것을 확인할 수 있다.

GIT전략 시나리오 - 버그수정

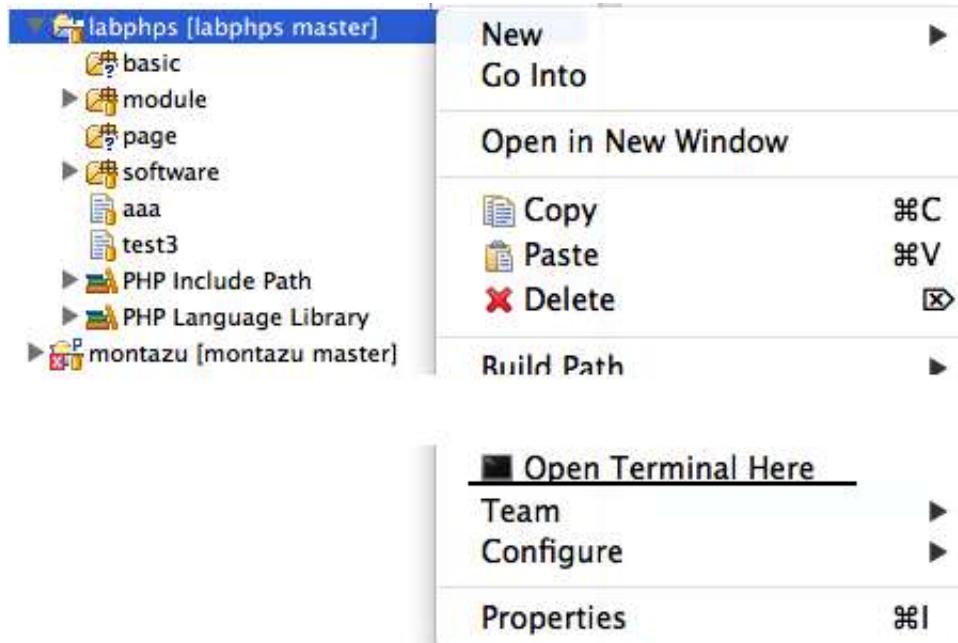
4. 선택합치기



GIT시나리오 릴리즈브랜치

GIT전략 시나리오 - 릴리즈브랜치

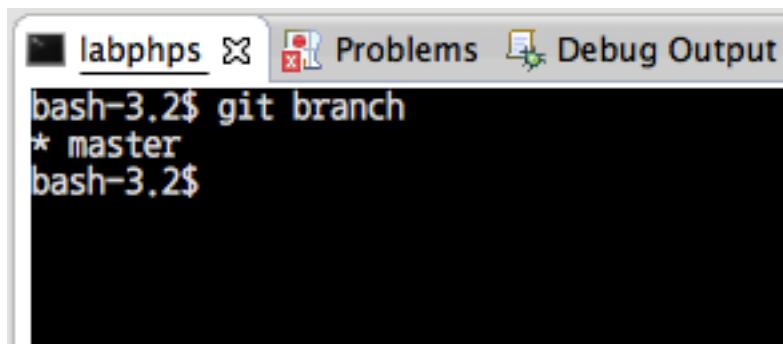
1. 브랜치 확인



터미널을 연다.

GIT전략 시나리오 - 릴리즈브랜치

1. 브랜치 확인



A screenshot of a terminal window titled "labphps". The window shows the following command and its output:
bash-3.2\$ git branch
* master
bash-3.2\$

Git branch 명령어로
브랜치를 확인한다.

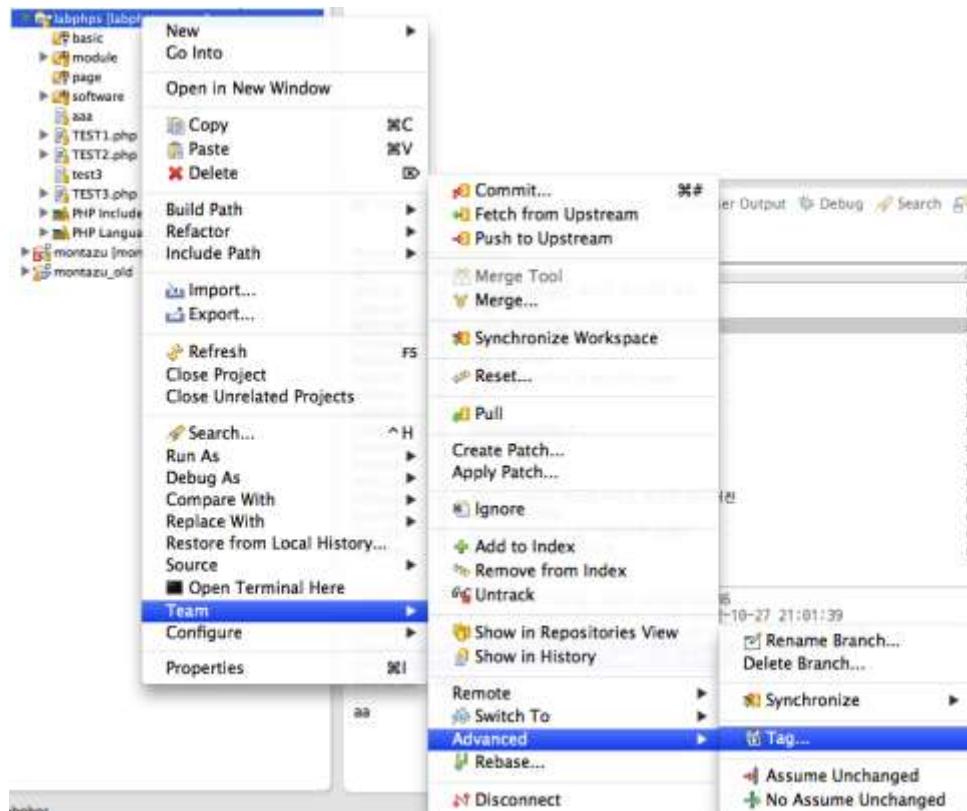
GIT전략 시나리오 - 릴리즈브랜치

1. 브랜치 확인



GIT전략 시나리오 - 릴리즈브랜치

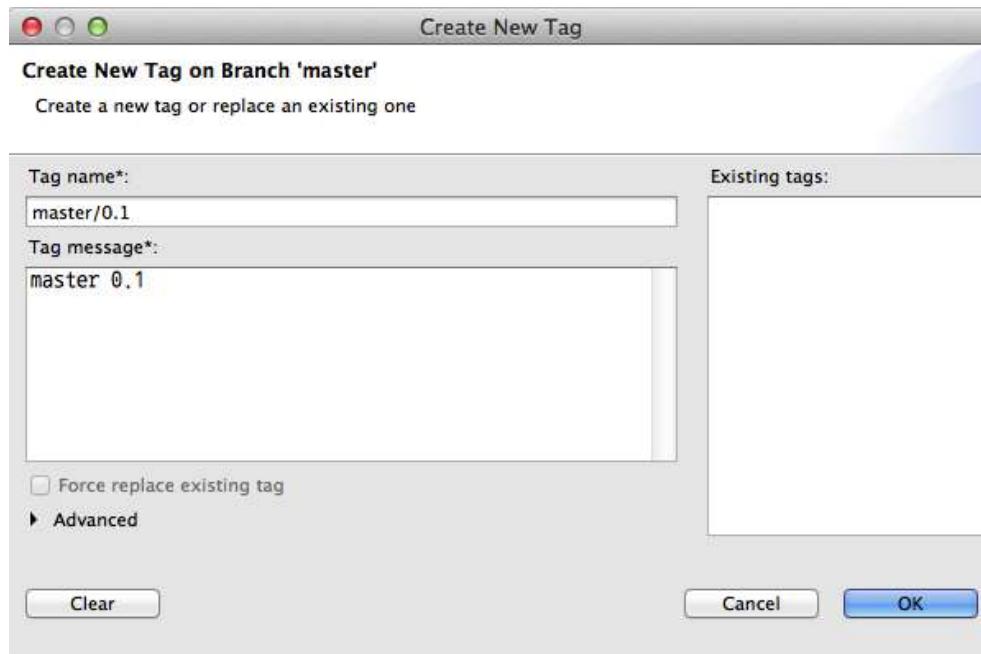
2. 릴리즈 시점에서 태그



Egit태그기능으로
태그 한다.

GIT전략 시나리오 - 릴리즈브랜치

2. 릴리즈 시점에서 태그



태그명을 입력한 후
Ok를 클릭한다.

GIT전략 시나리오 - 릴리즈브랜치

2. 릴리즈 시점에서 태그

Id	Message	Status
ef757a2	로그인 버그수정 완료	master/0.1 master HEAD
580e04f	기능개발완료	
b60238a	(ORIG_HEAD) aa	

이력을 확인하여 정원하는 시점에 태그가 생성되는지 확인한다.

GIT전략 시나리오 - 릴리즈브랜치

2. 릴리즈 시점에서 태그



GIT전략 시나리오 - 릴리즈브랜치

3. 태그로부터 브랜치

```
bash-3.2$ git tag  
master/0.1  
bash-3.2$ git checkout master/0.1  
Note: checking out 'master/0.1'.  
  
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 performing another checkout.  
  
If you want to create a new branch to retain commits you create, you may  
do so (now or later) by using -b with the checkout command again. Example:  
  
  git checkout -b new_branch_name  
  
HEAD is now at ef757a2... 로그인 버그수정 완료  
bash-3.2$ git branch  
* (no branch)  
  master  
bash-3.2$ █
```

터미널을 이용하여
태그로 포인터를
옮긴다.

GIT전략 시나리오 - 릴리즈브랜치

3. 태그로부터 브랜치

태그는 하나의 포인터로 브랜치가 아니므로 읽기만 가능하다. No branch로써 설명해주고 있다. 따라서 현재 시점에서 브랜치를 빼내어야 한다.

GIT전략 시나리오 - 릴리즈브랜치

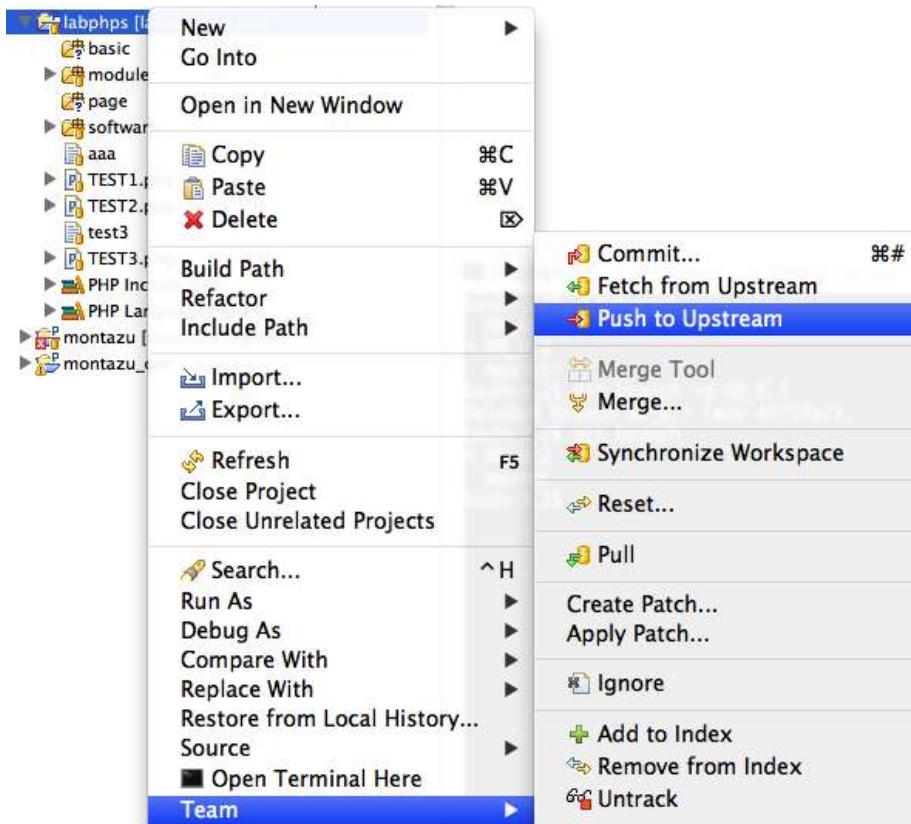
3. 태그로부터 브랜치

```
bash-3.2$ git checkout -b RB_0.1
Switched to a new branch 'RB_0.1'
bash-3.2$ git branch
* RB_0.1
  master
bash-3.2$
```

-b 전달인자로 포인터가
가리킨 태그를 기준으로
브랜치를 생성한다.

GIT전략 시나리오 - 릴리즈브랜치

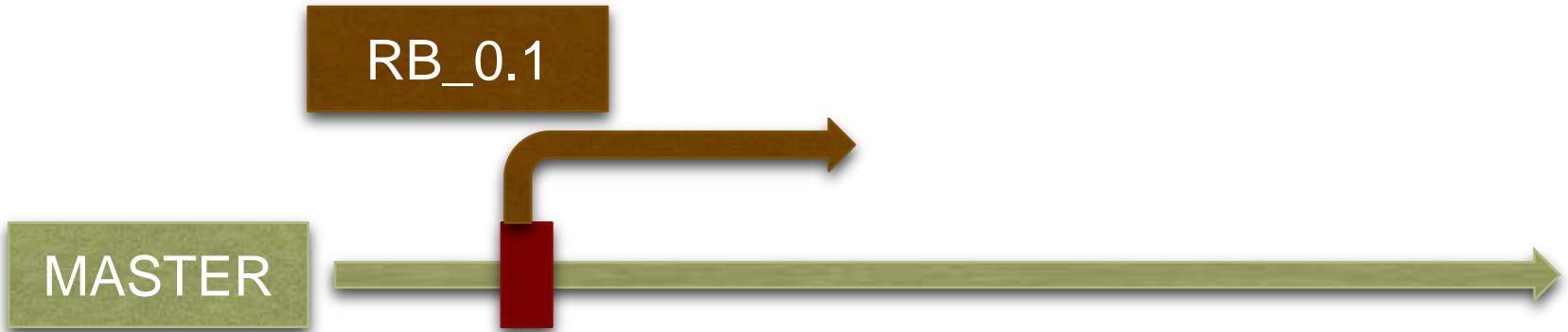
3. 태그로부터 브랜치



그리고 원격저장소에 push한다.

GIT전략 시나리오 - 릴리즈브랜치

3. 태그로부터 브랜치



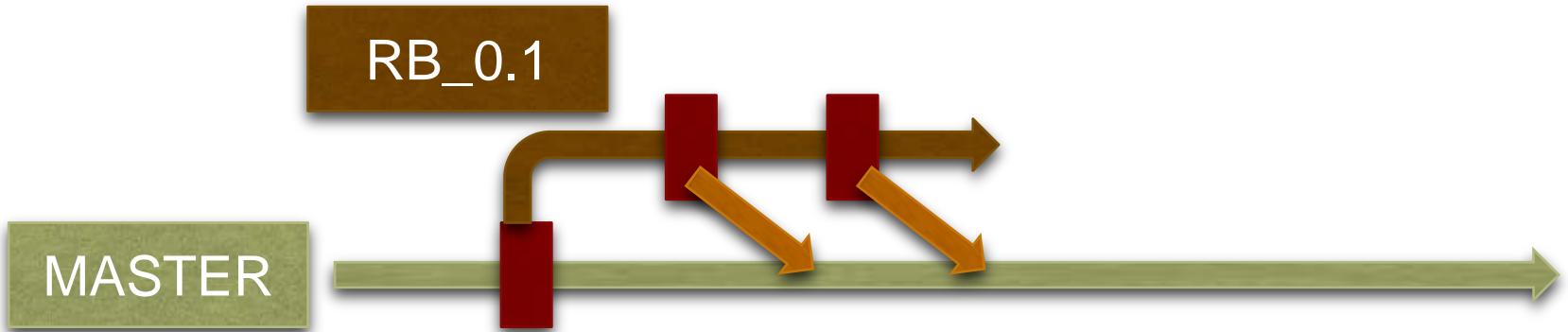
GIT전략 시나리오 - 릴리즈브랜치

4. 릴리즈브랜치에서 버그수정

릴리즈 브랜치에서는 최소한의 변경만 해야한다.
버그나 로직의 수정에만 집중할 뿐 새로운 기능을
추가하지 않는다.

GIT전략 시나리오 - 릴리즈브랜치

4. 릴리즈브랜치에서 버그수정



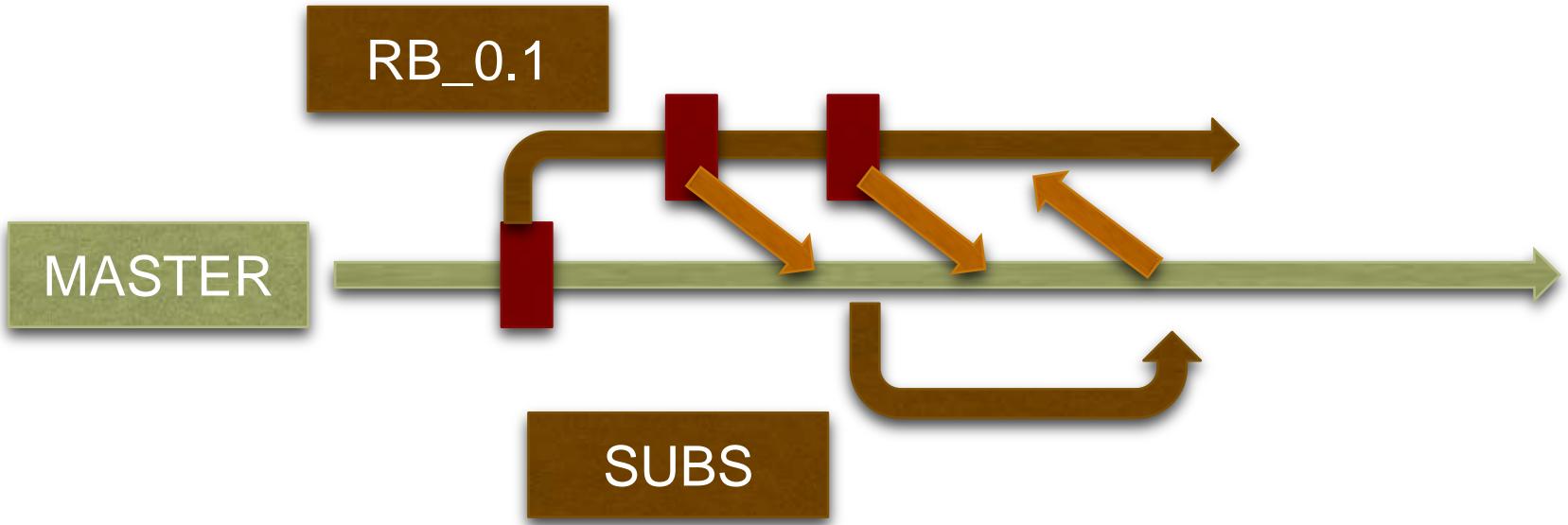
GIT전략 시나리오 - 릴리즈브랜치

5. 기능추가

릴리즈 브랜치에서는 기능추가를 하지 않는다.
기능추가시 또 다른브랜치를 생성하여
기능추가하고 마스터브랜치에만 병합하거나
필요시 릴리즈브랜치에도 병합한다.

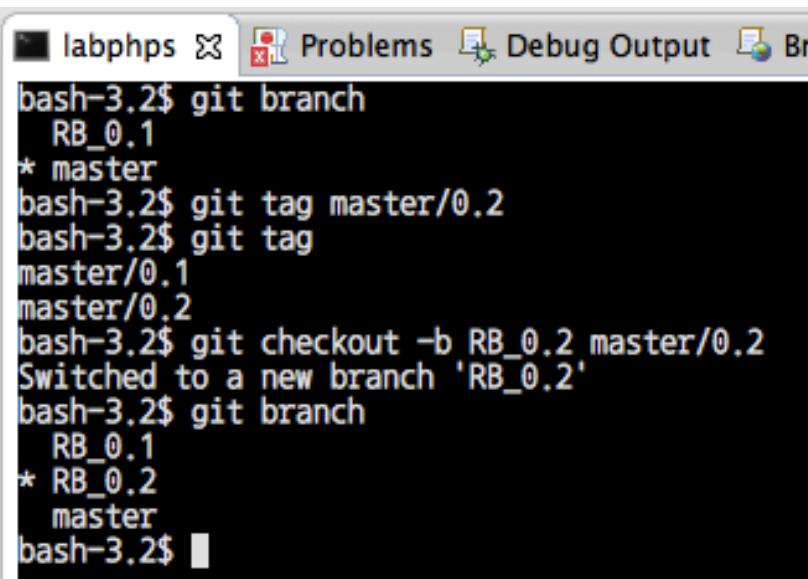
GIT전략 시나리오 - 릴리즈브랜치

5. 기능추가



GIT전략 시나리오 - 릴리즈브랜치

6. 버전업 릴리즈



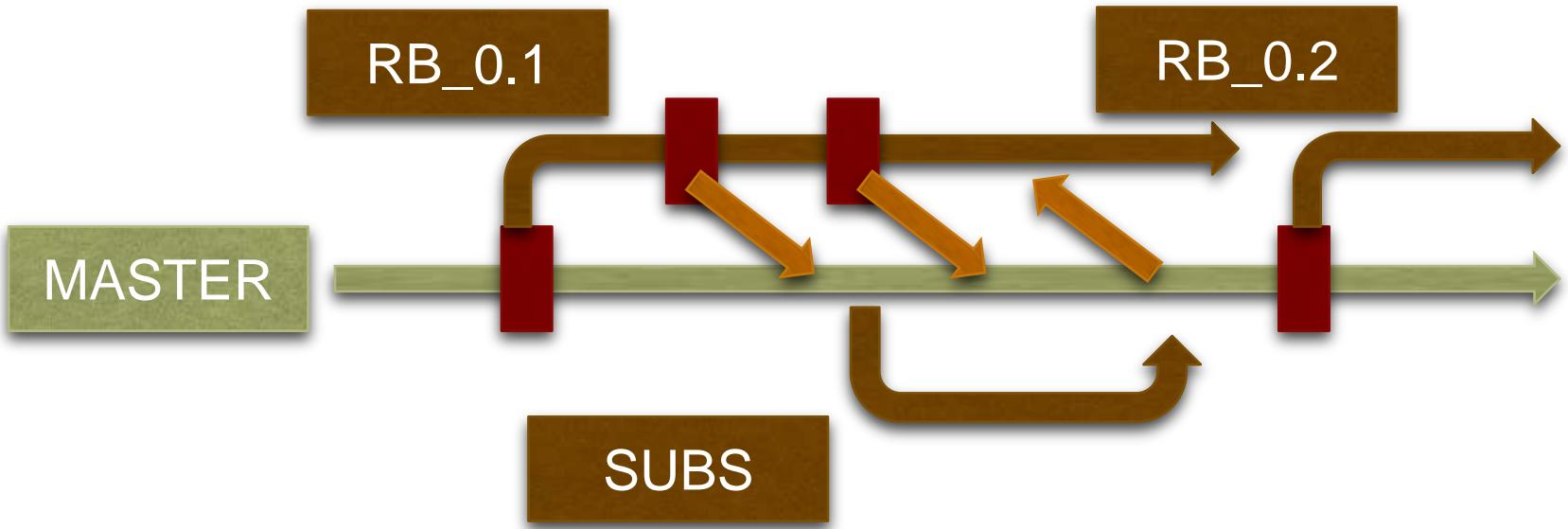
```
labphps ✘ Problems Debug Output Br
bash-3.2$ git branch
  RB_0.1
* master
bash-3.2$ git tag master/0.2
bash-3.2$ git tag
master/0.1
master/0.2
bash-3.2$ git checkout -b RB_0.2 master/0.2
Switched to a new branch 'RB_0.2'
bash-3.2$ git branch
  RB_0.1
* RB_0.2
  master
bash-3.2$ ■
```

Master브랜치의 일정수준
기능이 구현되면 또다른
버전이 배포될 것이다.

그 시점에서 새로운 태그를
생성하고 브랜치를
생성한다.

GIT전략 시나리오 - 릴리즈브랜치

6. 버전업 릴리즈



GIT전략 시나리오 - 릴리즈브랜치

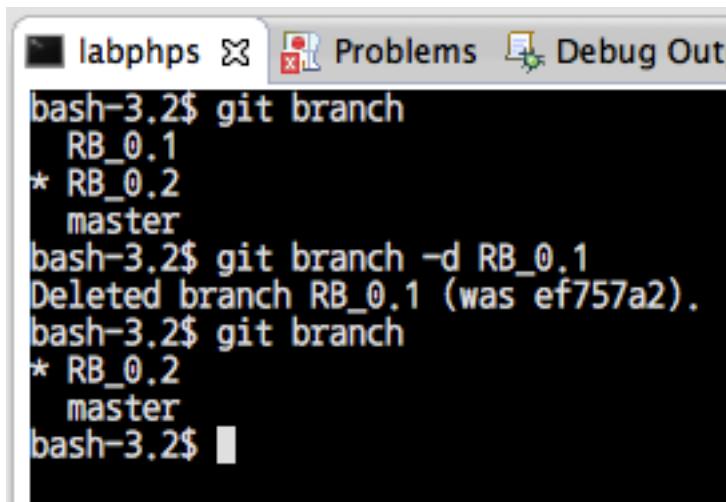
7. 구 릴리즈브랜치 삭제

새로운 릴리즈브랜치가 적용되었다면 구 릴리즈의
이력은 태그가 위치를 표시하고 있으므로 이력을
유지하기 위해 브랜치를 유지할 필요가 없다.

따라서 구 릴리즈브랜치를 삭제한다.

GIT전략 시나리오 - 릴리즈브랜치

7. 구 릴리즈브랜치 삭제



The screenshot shows a terminal window within the Eclipse IDE interface. The title bar of the terminal window says "labphps". The terminal content is as follows:

```
bash-3.2$ git branch
  RB_0.1
* RB_0.2
  master
bash-3.2$ git branch -d RB_0.1
Deleted branch RB_0.1 (was ef757a2).
bash-3.2$ git branch
* RB_0.2
  master
bash-3.2$
```

커マン드를 이용해도
되고, egit을 이용해도
된다.

브랜치는 삭제되었다.

GIT전략 시나리오 - 릴리즈브랜치

7. 구 릴리즈브랜치 삭제

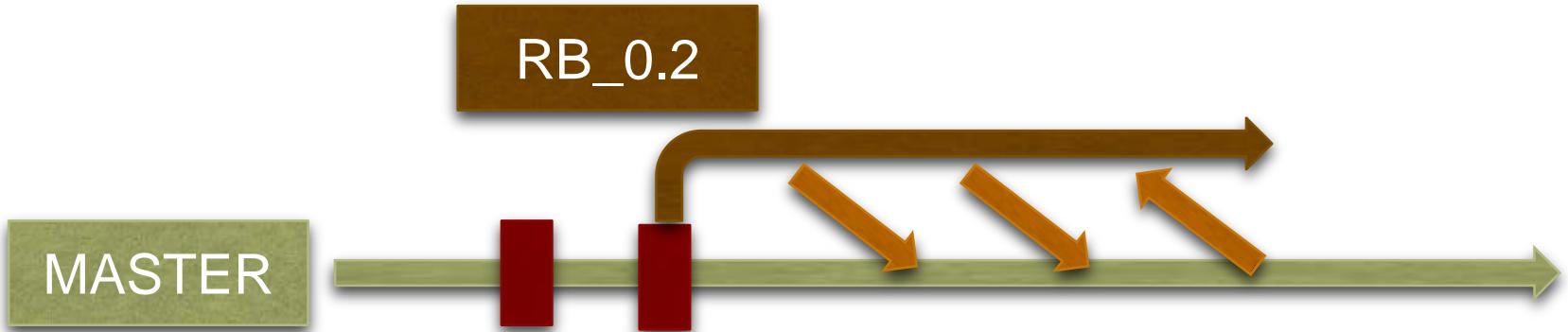
```
bash-3.2$ git checkout master
Switched to branch 'master'
bash-3.2$ git branch -d RB_0.2
Deleted branch RB_0.2 (was ef757a2).
bash-3.2$ git push origin :RB_0.2
Password for 'https://uyeong@bitbucket.org':
remote: bb/acl: uyeong is allowed. accepted payload.
To https://uyeong@bitbucket.org/uyeong/labphps.git
 - [deleted]          RB_0.2
bash-3.2$
```

원격저장소의 브랜치도
삭제한다.

스샷은 0.2로 되어있음

GIT전략 시나리오 - 릴리즈브랜치

6. 버전업 릴리즈



기타

참고하면 좋은 사이트

http://wiki.redgolems.com/03.server:git:01.git에_대하여

[http://www.slideshare.net/ajaxiankr/
2011-kth-h3-track-b-4-advanced-git-by-aj](http://www.slideshare.net/ajaxiankr/2011-kth-h3-track-b-4-advanced-git-by-aj)

<http://blog.outsider.ne.kr/572>

PPT작성자

Redgolems – UYEONG

uyeong21c@gmail.com

감사합니다.