Git 이해와 활용 기본

작성자:이응준
소속팀 / 상위부서: 기술전략팀/기술혁신센터
작성년월일: 2012. 12. 8
일반

0. 이 교육에 대해

목적

Git 으로 코드를 관리하는 프로젝트에 **개발자**로서 참여할 수 있다.

목표

Git의 개념과 동작 원리를 이해한다.
(Git의 메시지와 도움말을 **해석**하려면 필수)

다루는 것

기본적이고 많이 쓰이는 기능들
내부 동작
개념과 원리

다루지 않는 것

Git 저장소 운영
IDE 연동
SVN <-> Git

1. Git 소개

공식 정의

the stupid content tracker

알기 쉬운 설명

특정 시점에서의 파일들의 상태를, 저장하고 보여주는 소프트웨어

분류

분산형 버전관리 시스템 (DVCS)

만든 사람

Linus Torvalds

만든 때

2005년 4월

만든 배경

리눅스 커널 소스코드를 관리하기 위해 사용하던 BitKeeper가 상용화되어버리는 바람에 직접 버전관리시스템을 만듦

장점

속도가 빠르다
디스크를 적게 차지한다
좋은 기능이 많다

단점

어렵다

Mac OS X

기본적으로 설치되어있음. 없다면 아래 링크에서 다운로드 받아 설치

http://code.google.com/p/git-osx-installer/downloads/list?can=3

Linux

패키지 관리자를 통해 설치

데비안 계열(우분투 등): `$ sudo apt-get install git-core`
rpm 계열(Fedora, CentOS 등): `$ sudo yum install git`

혹은 아래 링크를 통해 패키지를 다운로드 받아 설치 (데비안)

http://packages.debian.org/stable/git

Windows

아래 링크를 통해 msysgit 1.7.9를 다운로드 받아 설치하고,

http://msysgit.googlecode.com/files/Git-1.7.9-preview20120201.exe
(주의: 1.7.10 버전은 한글이 입력되지 않으니 설치하지 말 것)

한글 입력을 위해 ~/.inputrc를 다음과 같이 수정

```
set output-meta on
set convert-meta off
```

이후 **Git Bash** 를 실행하여 Git을 사용할 수 있음

그 외의 운영체제, 혹은 위의 방법이 모두 불가능하면

아래 링크를 통해 소스코드를 다운로드 받아 컴파일하여 설치

http://code.google.com/p/git-core/downloads/detail?name=git-1.7.10.tar.gz

2. 저장소 만들기

현재 디렉토리를 Git으로 관리하기 시작하려면,

```
$ git init
```

새로 디렉토리를 만들고 그 디렉토리를 Git으로 관리하려면,

```
$ git init <디렉토리명>
```

2.2 실습 - init

```
$ mkdir hello
$ cd hello
$ git init
```

2.3 저장소 구조

Working Directory	Staging Area (Index)	Repository
./	.git/index	.git/

이름, 메일주소 설정 (커밋시 사용됨)

```
$ git config --global user.name 'yourname'
$ git config --global user.email 'you@mail.com ‘
```

crlf 자동변환 (Windows의 경우)

```
$ git config --global core.autocrlf input
```

인코딩 설정 (Windows의 경우)

```
$ git config --global i18n.logOutputEncoding cp949
```

3. 커밋하기

커밋?

특정 시점의 상태

커밋하다?

특정 시점의 상태를 저장소에 저장하다

커밋하는 법

```
$ git add file1 file2… (변경된 파일들에 대해)
$ git commit  -m ‘로그 메시지’
```

3.2 실습 - add, commit

```
$ echo '#include <stdio.h>

int main(int argc, const char** argv) {
     printf("Hello, World\n");
     return 0;
}' > hello.c
$ git add hello.c                          # hello.c 를 Staging Area 에 추가
$ git commit -m 'Hello, World'             # Staging Area 에 저장된 내용을 'Hello, World' 라는 로그메시지로 커밋
$ echo '#include <stdio.h>

int main(int argc, const char** argv) {
    printf("Hello, %s\n", argv[1]);
    return 0;
}' > hello.c
$ git add hello.c                                          # Staging Area 에 hello.c 를 추가
$ git commit -m '사용자가 안녕할 대상을 지정 가능하게 수정'    # Staging Area 를 커밋
$ echo ‘ This is the Helloworld example. ’ > README
$ git add README                                           # Staging Area 에 README 를 추가
$ git commit  - m ‘ 프로그램을 소개하는 README 추가’          # Staging Area 를 커밋
```

3.3 Git Internal - add, commit

Working Directory

hello.c (untracked)

```
#include <stdio.h>

int main(int argc, const char** argv) {
printf("Hello, World₩n");
}
```

Staging Area

Repository

hello.c 작성

3.3 Git Internal - add, commit

Working Directory

hello.c (staged)

```
#include <stdio.h>

int main(int argc, const char** argv) {
printf("Hello, World₩n");
}
```

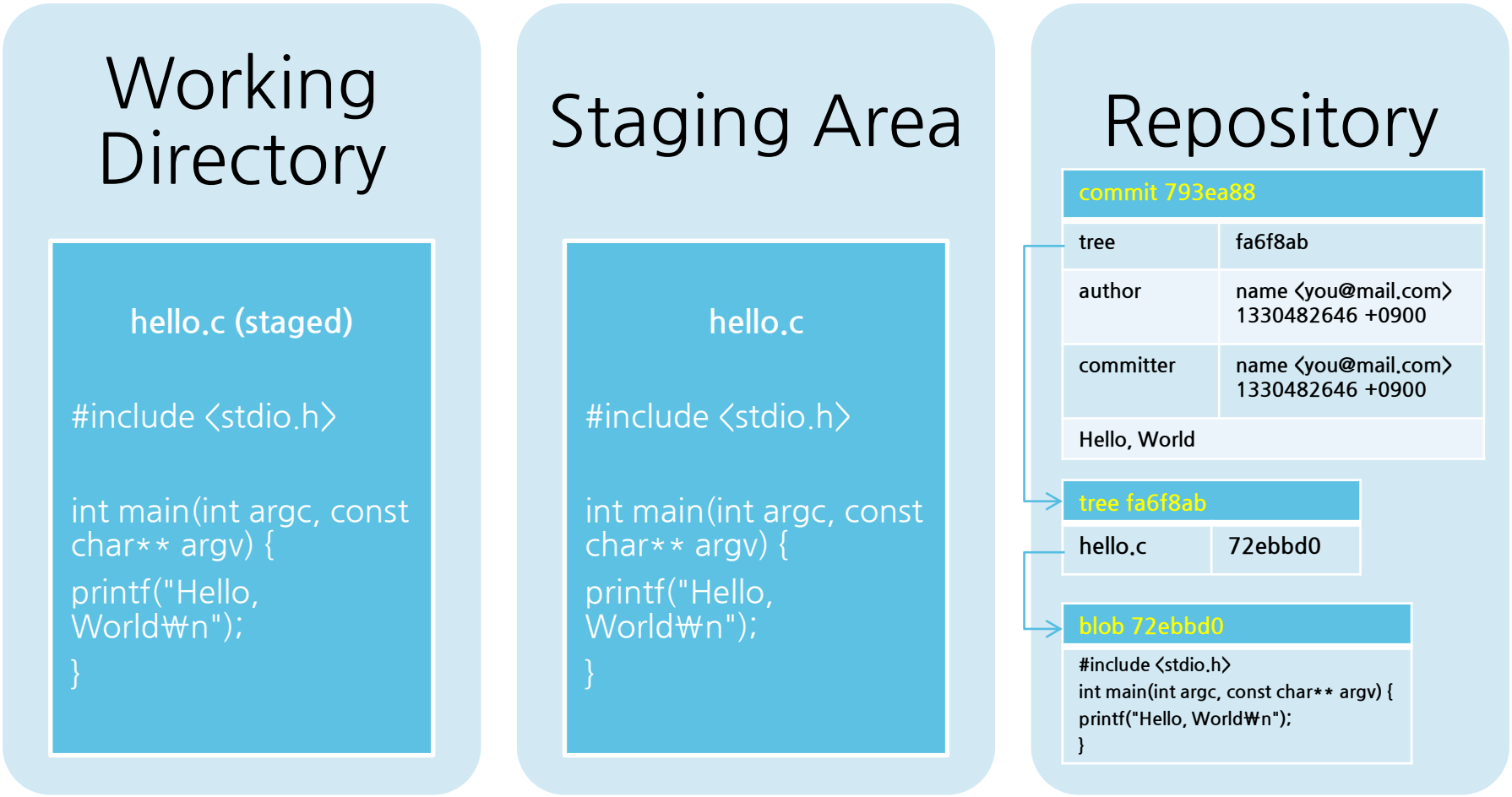
Staging Area

hello.c

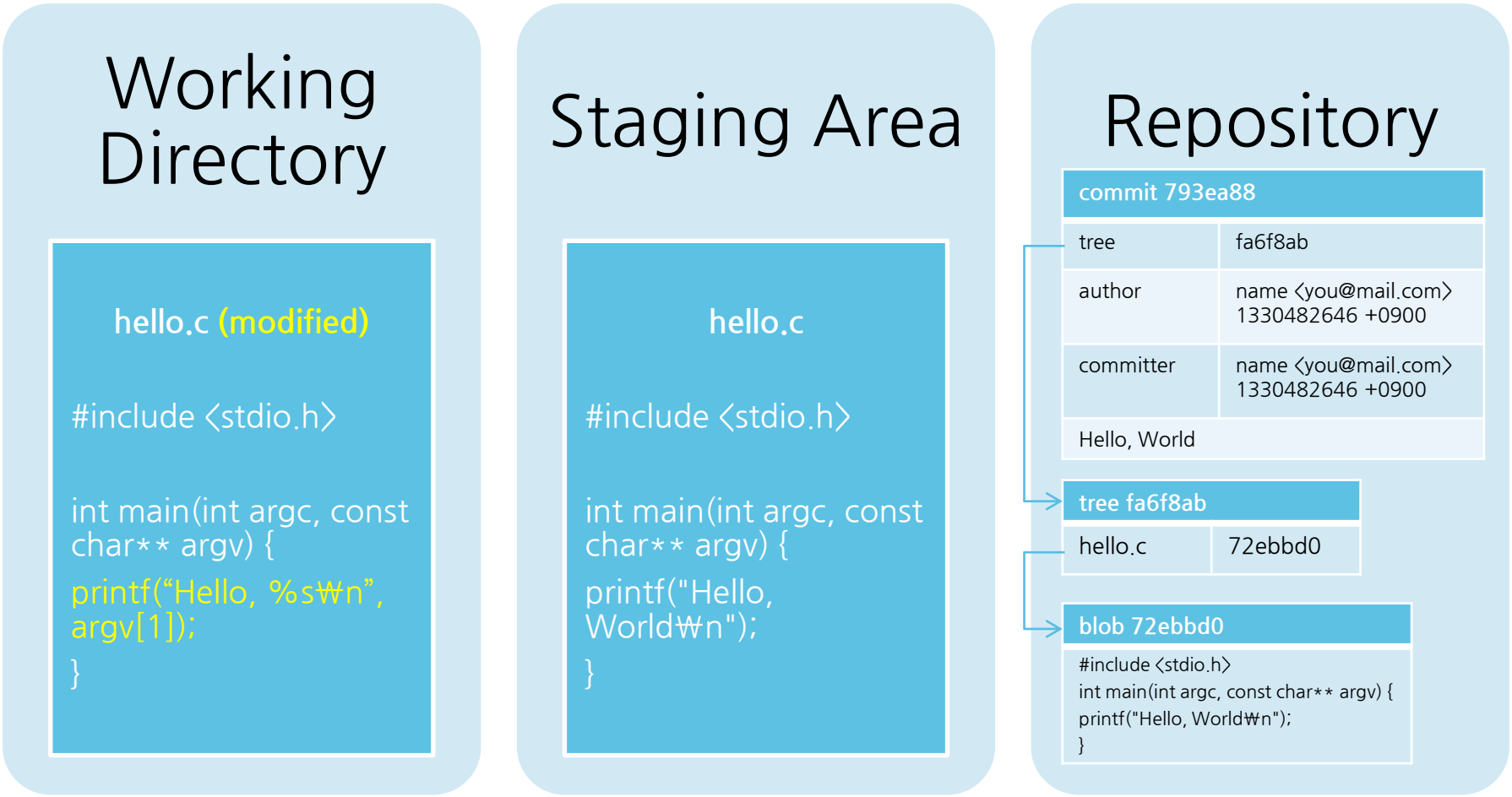
```
#include <stdio.h>

int main(int argc, const char** argv) {
printf("Hello, World₩n");
}
```

Repository

$ git add hello.c

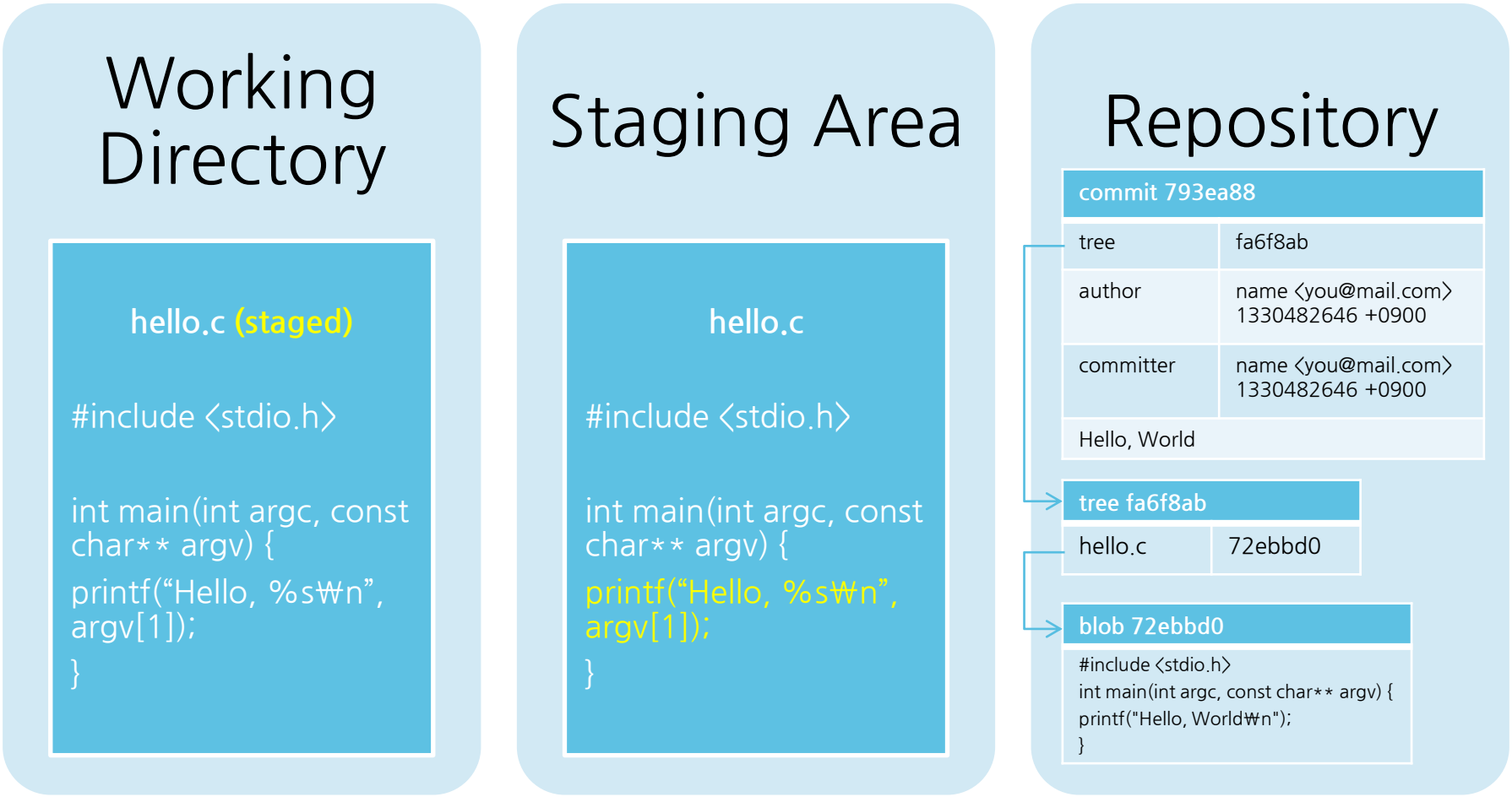


$ git commit hello.c -m ‘Hello, World’



hello.c 파일을 편집

3.3 Git Internal - add, commit



$ git add hello.c

Working Directory

hello.c (staged)

```
#include <stdio.h>

int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

Staging Area

hello.c

```
#include <stdio.h>

int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

Repository

$ git add hello.c

3.3 Git Internal - add, commit

Working Directory

hello.c (staged)

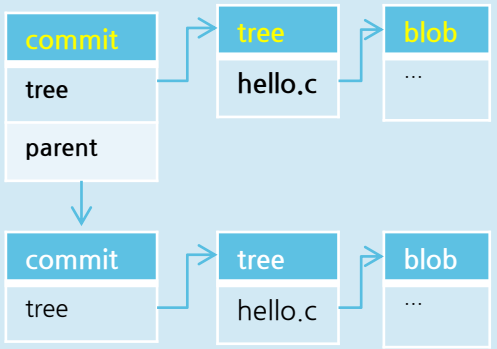
```
#include <stdio.h>

int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

Staging Area

hello.c

```
#include <stdio.h>

int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

Repository



$ git commit hello.c -m '안녕할 대상을 지정 가능하게'

Working Directory

hello.c (staged)

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README (untracked)

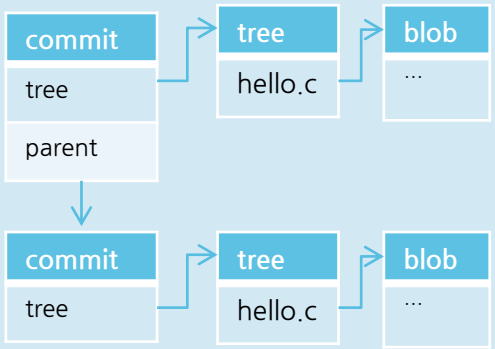
This is the helloworld example.

Staging Area

hello.c

```
#include <stdio.h>

int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

Repository



README 작성

3.3 Git Internal - add, commit

Working Directory

hello.c (staged)

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README (staged)

This is the helloworld example.

Staging Area

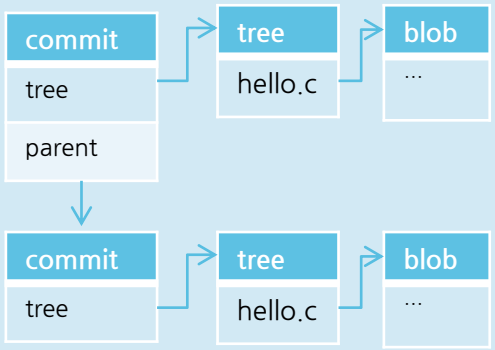
hello.c

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README

This is the helloworld example.

Repository



$ git add README

3.3 Git Internal - add, commit

Working Directory

hello.c (staged)

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README (staged)

This is the helloworld example.

Staging Area

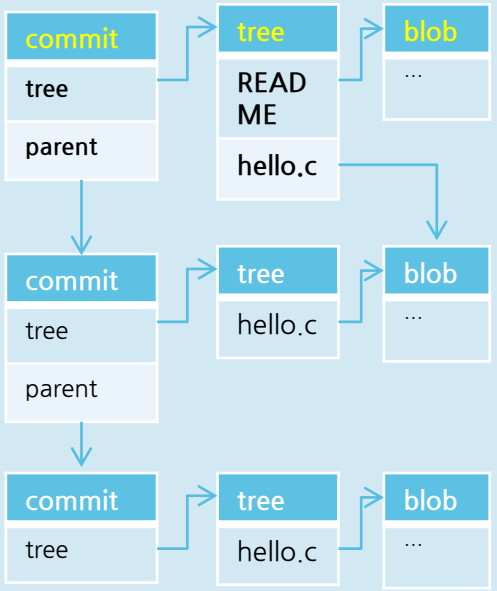
hello.c

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README

This is the helloworld example.

Repository



$ git commit -m '프로그램을 소개하는 README 추가'

파일 지우기

```
$ git rm <파일명>
$ git commit
```

파일 옮기기 (이름 바꾸기)

```
$ mv <파일명> <새 파일명>
$ git rm <파일명>
$ git add <새 파일명>
$ git commit
```

이렇게 해도 같음

```
$ git mv <파일명> <새 파일명>
$ git commit
```

3.5 실습 - rm, mv

```
$ mv hello.c main.c
$ git rm hello.c
$ git add main.c
$ git commit -m ‘hello.c를 main.c로 이름 바꾸기’
$ mkdir src
$ git mv main.c src/main.c
$ git commit -m ‘소스파일을 src 디렉토리로 옮김’
```

3.6 Git Internal - rm, mv

Working Directory

main.c (untracked)

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README (staged)

This is the helloworld example.

Staging Area

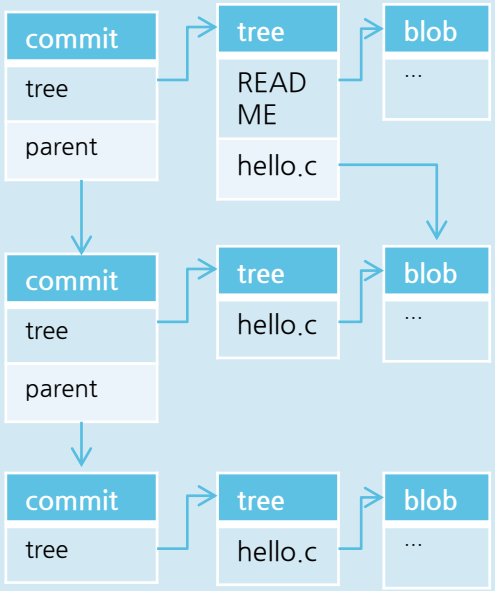
hello.c

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README

This is the helloworld example.

Repository




```
$ mv hello.c main.c
```

3.6 Git Internal - rm, mv

Working Directory

main.c (untracked)

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README (staged)

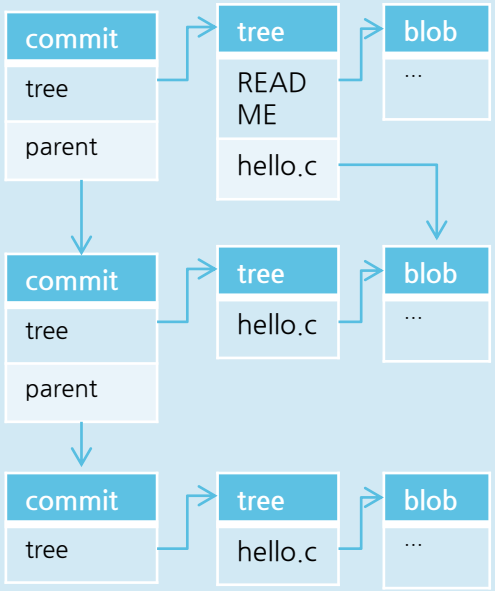
This is the helloworld example.

Staging Area

README

This is the helloworld example.

Repository




```
$ git rm hello.c
```

3.6 Git Internal - rm, mv

Working Directory

main.c (staged)

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README (staged)

This is the helloworld example.

Staging Area

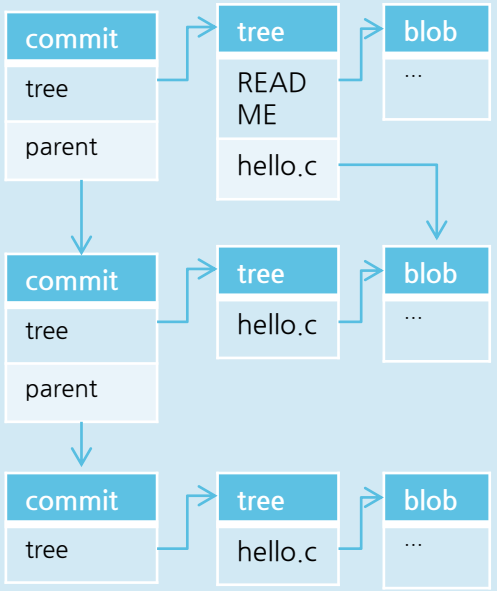
main.c

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README

This is the helloworld example.

Repository




```
$ git add main.c
```

3.6 Git Internal - rm, mv

Working Directory

main.c (staged)

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README (staged)

This is the helloworld example.

Staging Area

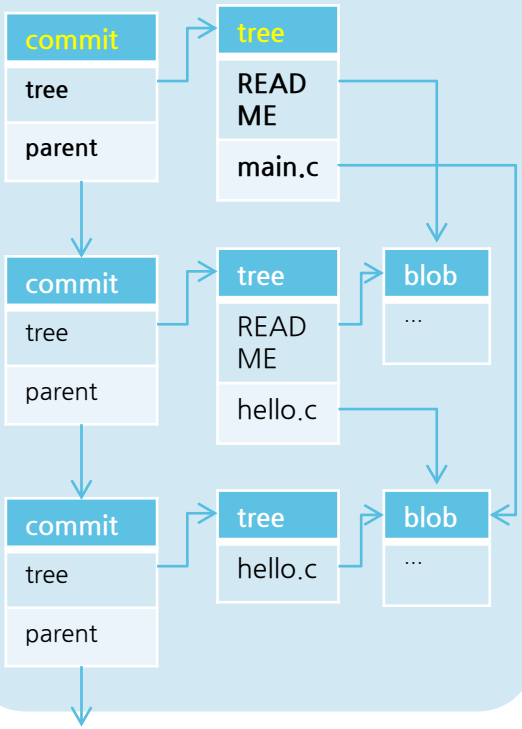
main.c

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README

This is the helloworld example.

Repository



$ git commit -m 'hello.c를 main.c로 이름 바꾸기'

3.6 Git Internal - rm, mv

Working Directory

main.c (staged)

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README (staged)

This is the helloworld example.

Staging Area

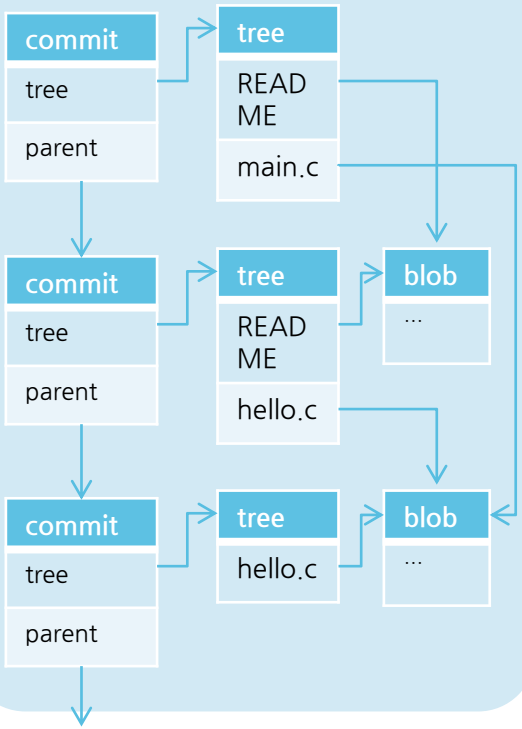
main.c

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README

This is the helloworld example.

Repository



$ mkdir src

3.6 Git Internal - rm, mv

Working Directory

src/main.c (staged)

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README (staged)

This is the helloworld example.

Staging Area

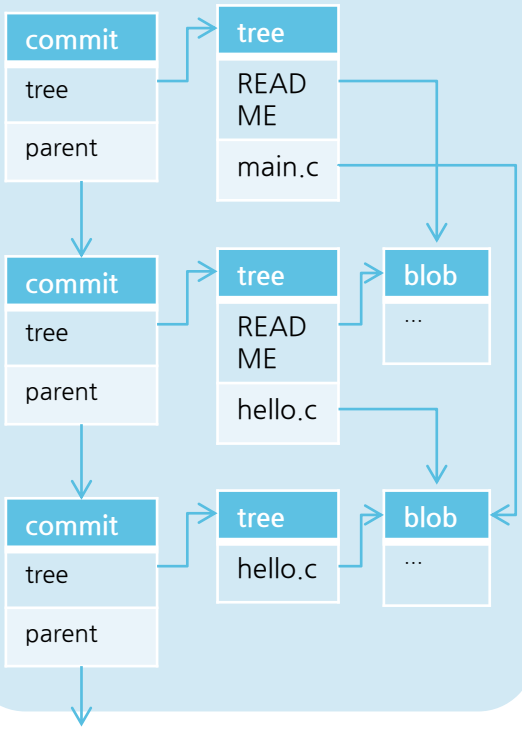
src/main.c

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README

This is the helloworld example.

Repository




```
$ git mv main.c src/main.c
```

3.6 Git Internal - rm, mv

Working Directory

src/main.c (staged)

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README (staged)

This is the helloworld example.

Staging Area

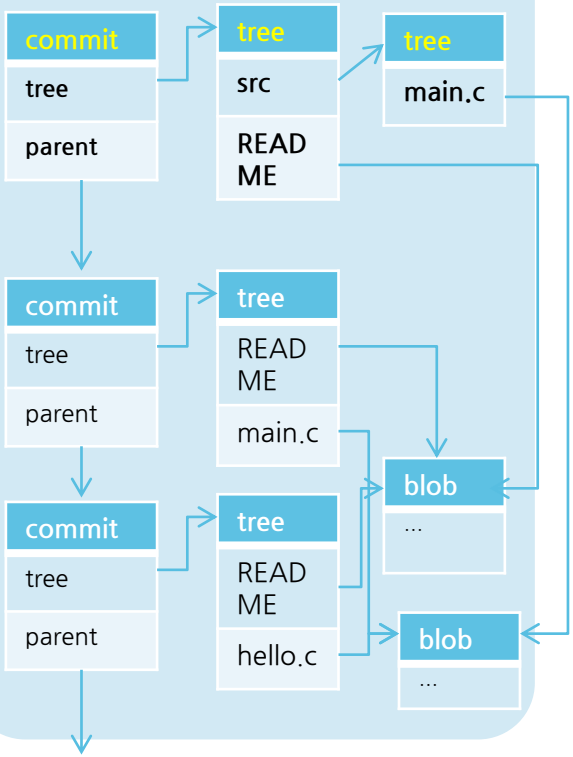
src/main.c

```
#include <stdio.h>
int main(int argc, const char** argv) {
printf("Hello, %s₩n", argv[1]);
}
```

README

This is the helloworld example.

Repository



$ git commit -m '소스파일을 src 디렉토리로 옮김'

Git Repository

Git Object의 key-value 저장소

Git Object

commit, tree(디렉토리), blob(파일), tag

Working Directory

작업할 파일들이 있는 곳

Staging Area (Index)

커밋될 파일들이 있는 곳

4. 되돌리기

되돌리기

```
$ git reset --hard <커밋>
$ git reset --hard HEAD^   # 직전 커밋으로 되돌리기
$ git reset --hard HEAD~1  # 직전 커밋으로 되돌리기
$ git reset --hard HEAD~2  # 전전 커밋으로 되돌리기
```

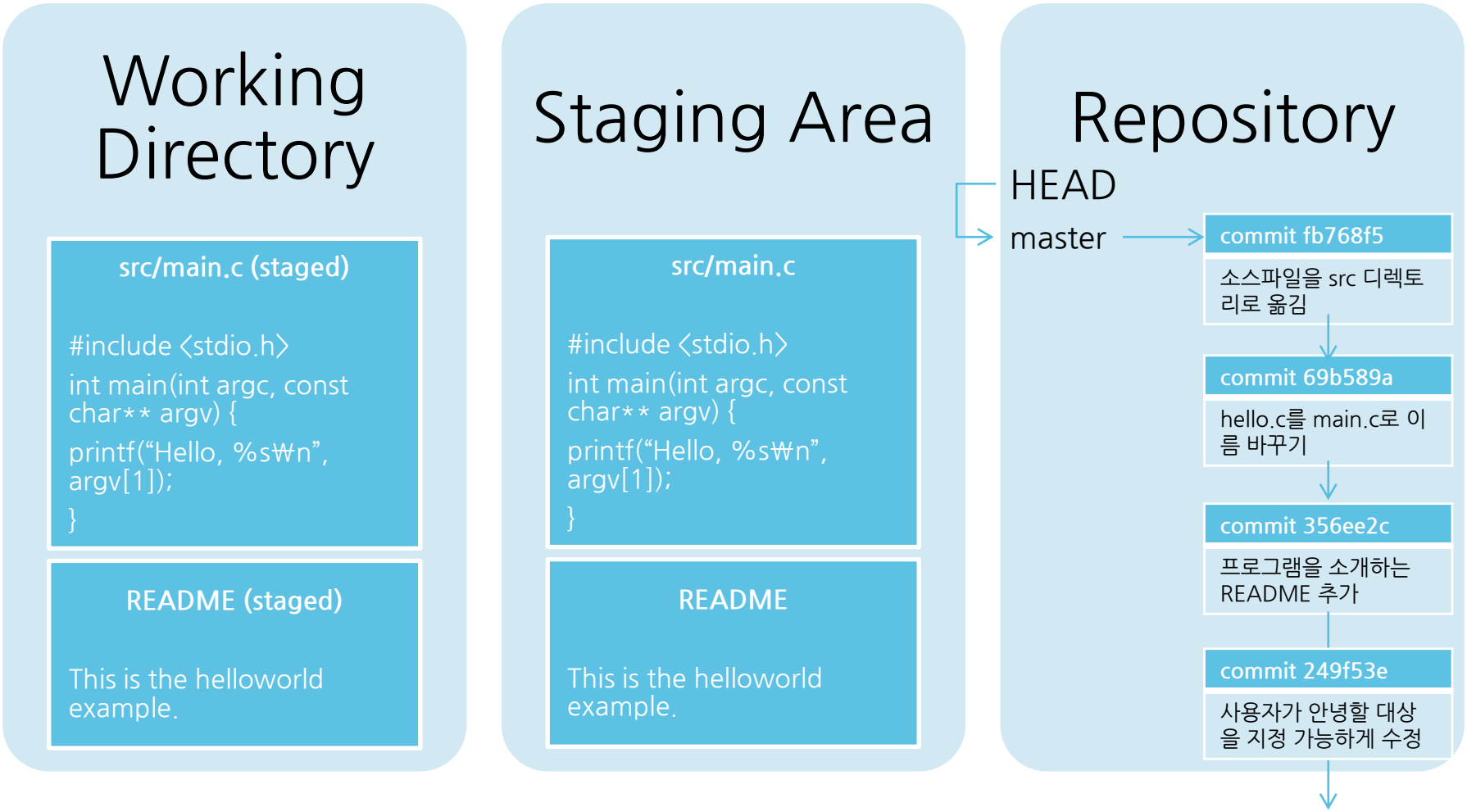
옵션에 따라 되돌리는 영역이 다름

	Working Directory	Staging Area	Repository
--hard	되돌림	되돌림	되돌림
--mixed	유지	되돌림	되돌림
--soft	유지	유지	되돌림

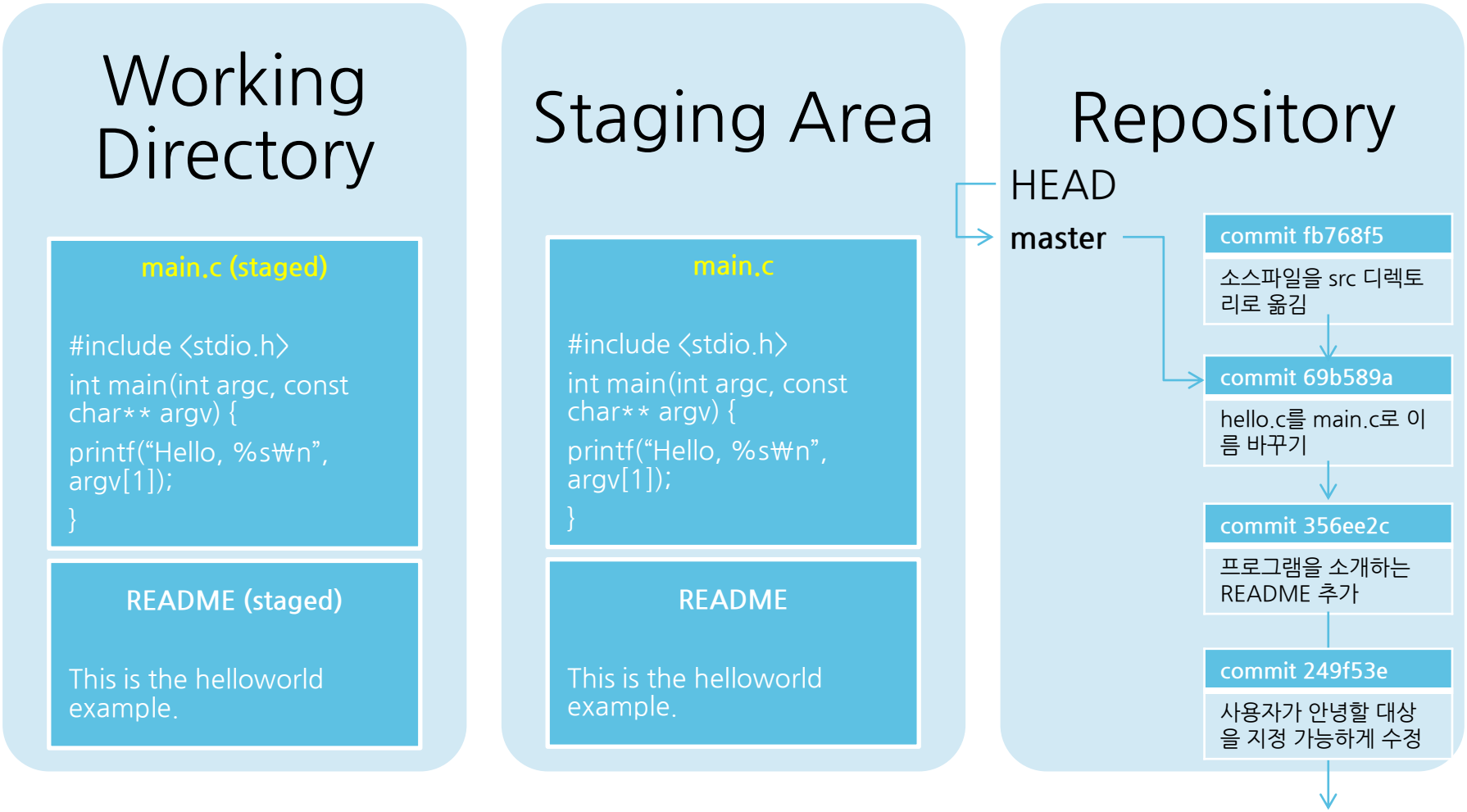
4.2 실습 - reset

```
$ git reset --hard HEAD^    # 직전 커밋으로 되돌림
$ git log                   # 되돌려졌는지 확인
$ git status                # 저장소 상태 확인
$ git reset --soft HEAD^    # 직전 커밋으로 되돌림
$ git log                   # 되돌려졌는지 확인
$ git status                # 저장소 상태 확인
$ git reset --mixed HEAD^   # 직전 커밋으로 되돌림
$ git log                   # 되돌려졌는지 확인
$ git status                # 저장소 상태 확인
```

4.3 Git Internal - reset

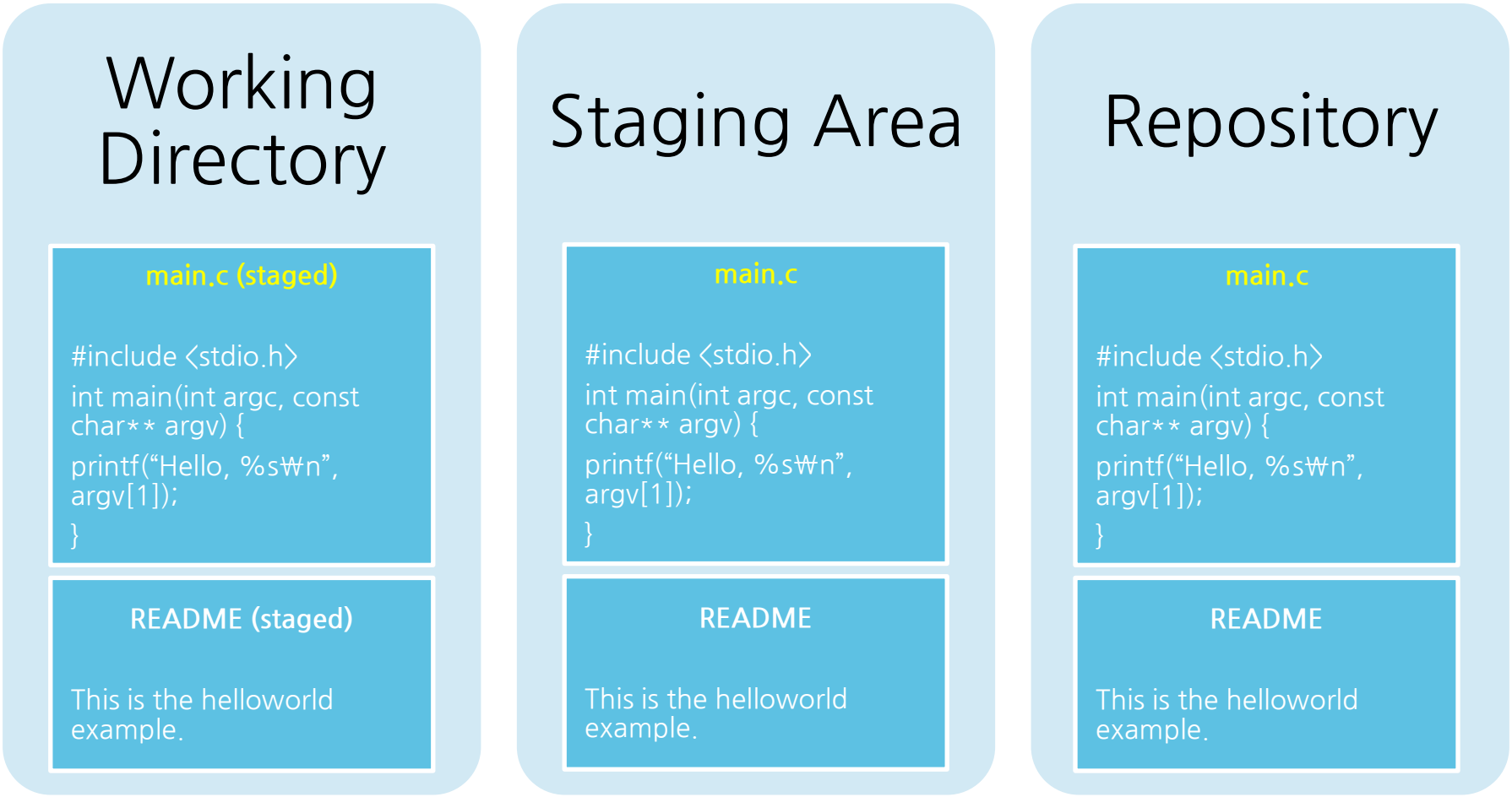


4.3 Git Internal - reset

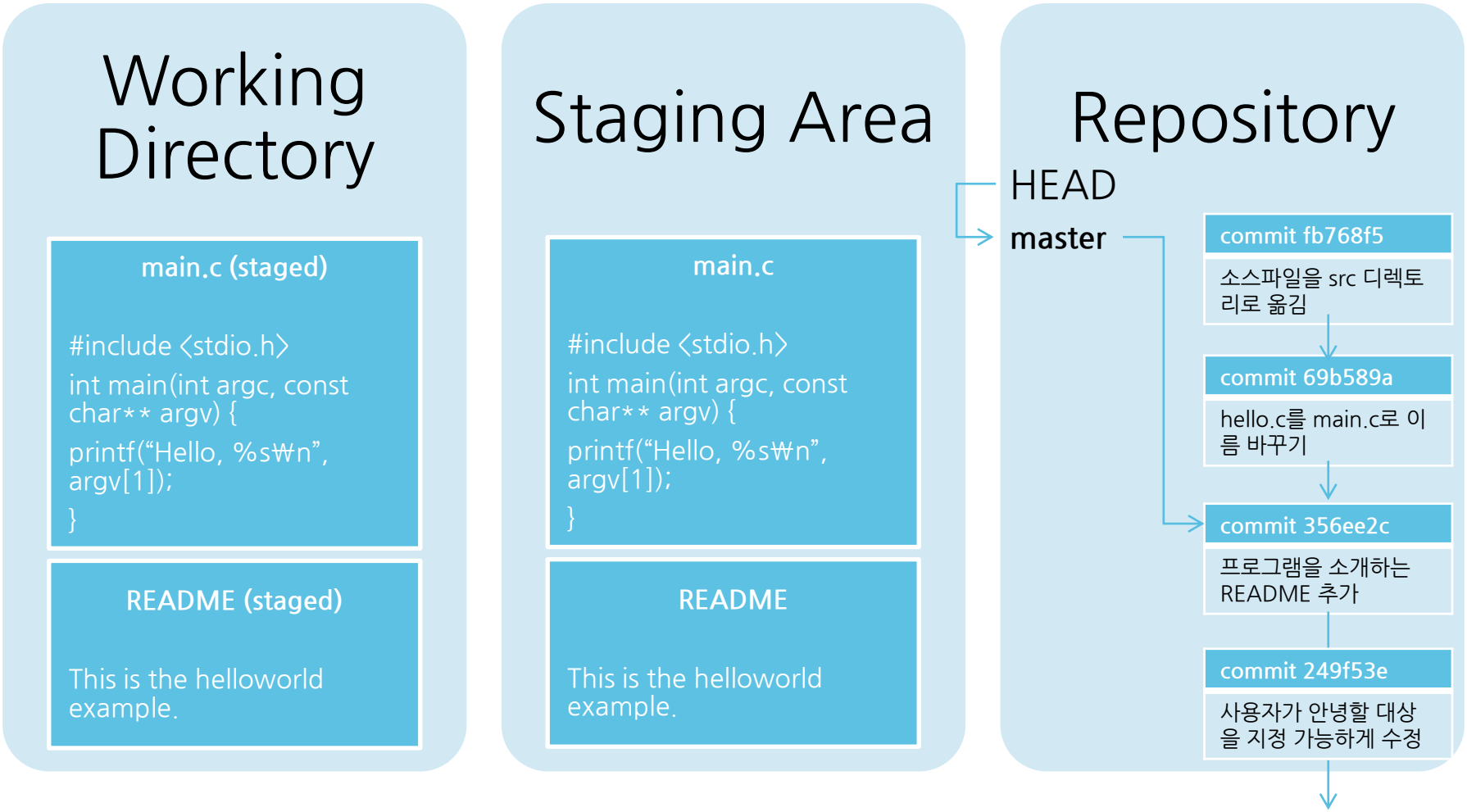


$ git reset --hard HEAD^

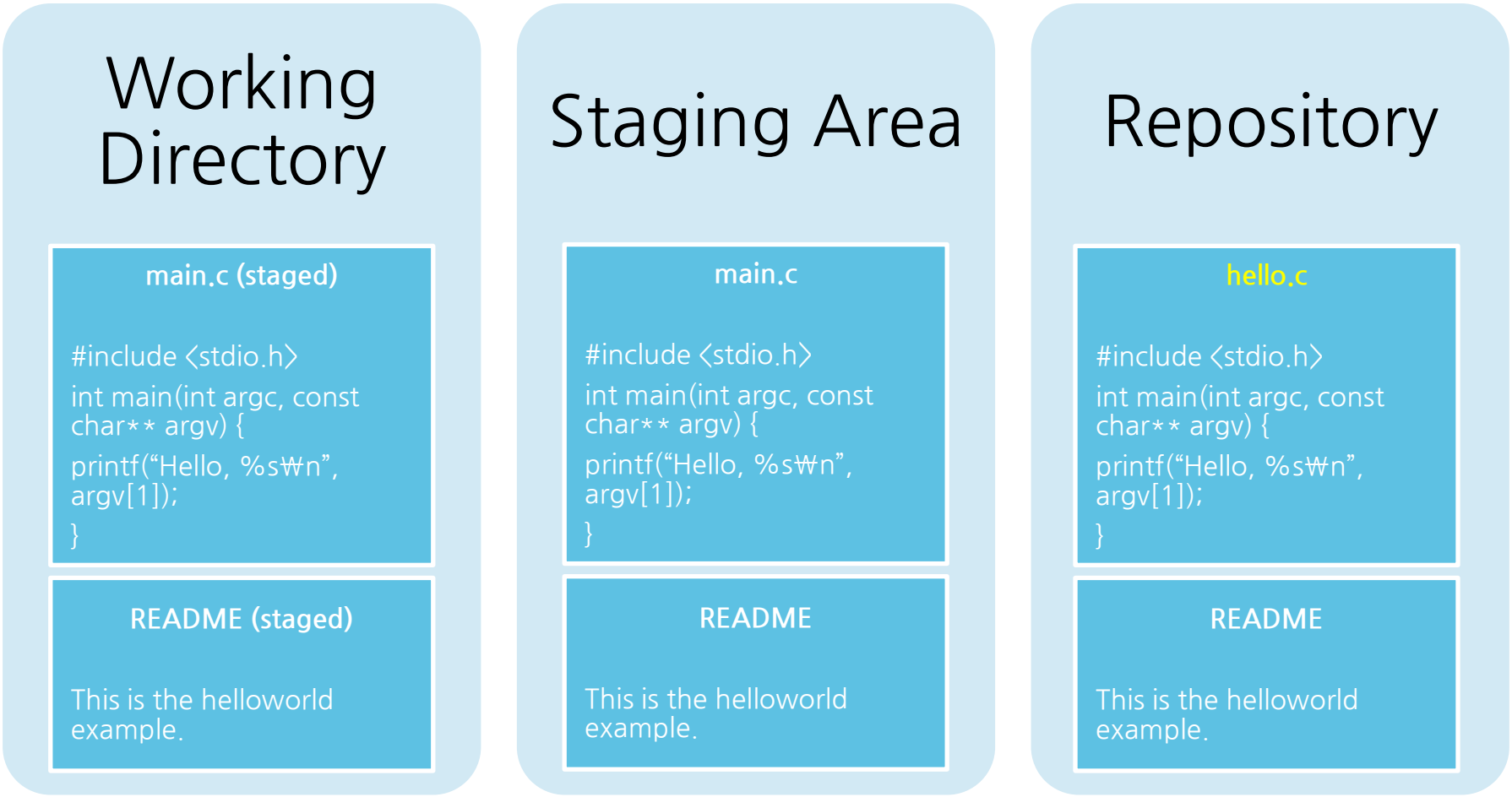
4.3 Git Internal - reset



nothing to commit

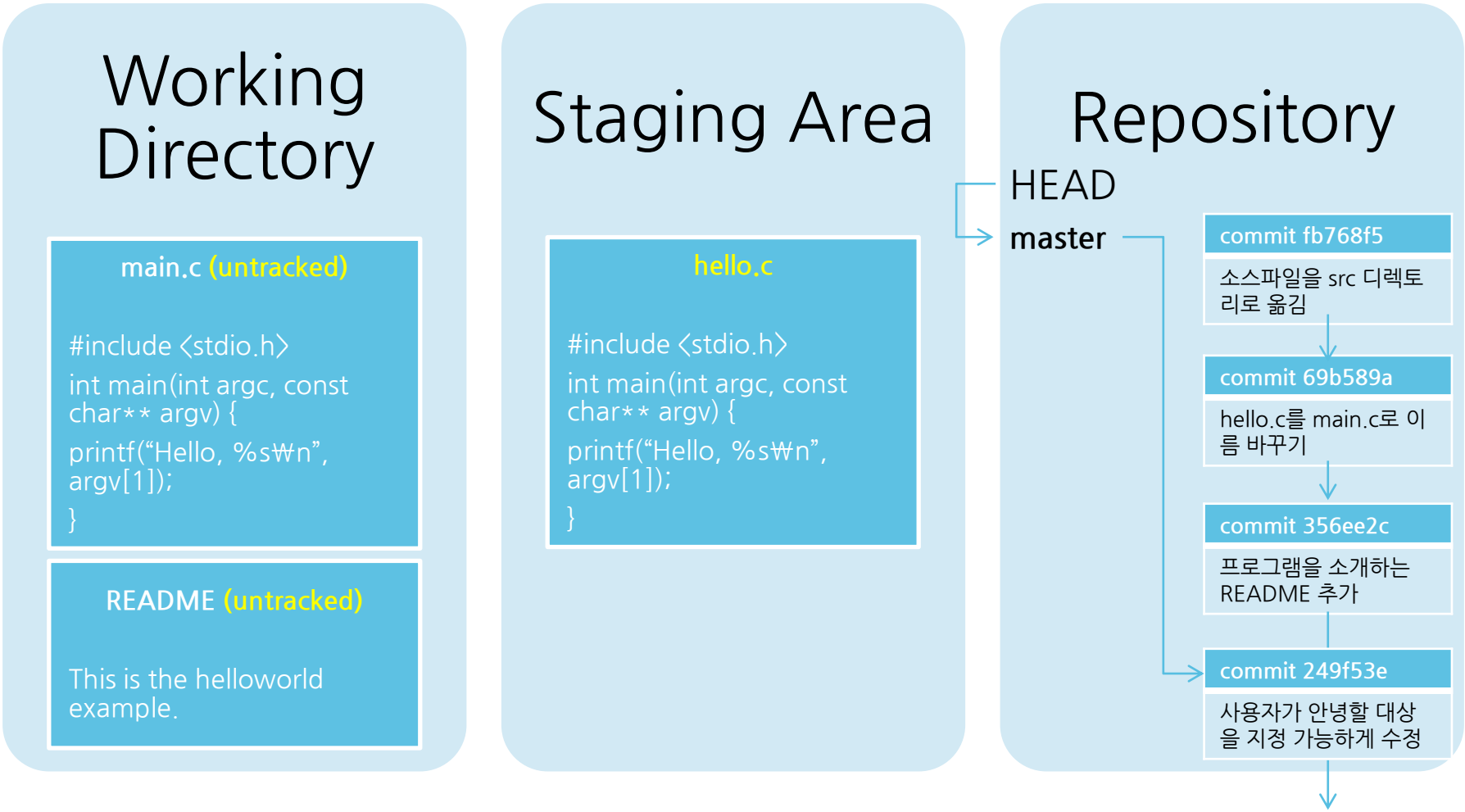


$ git reset --soft HEAD^



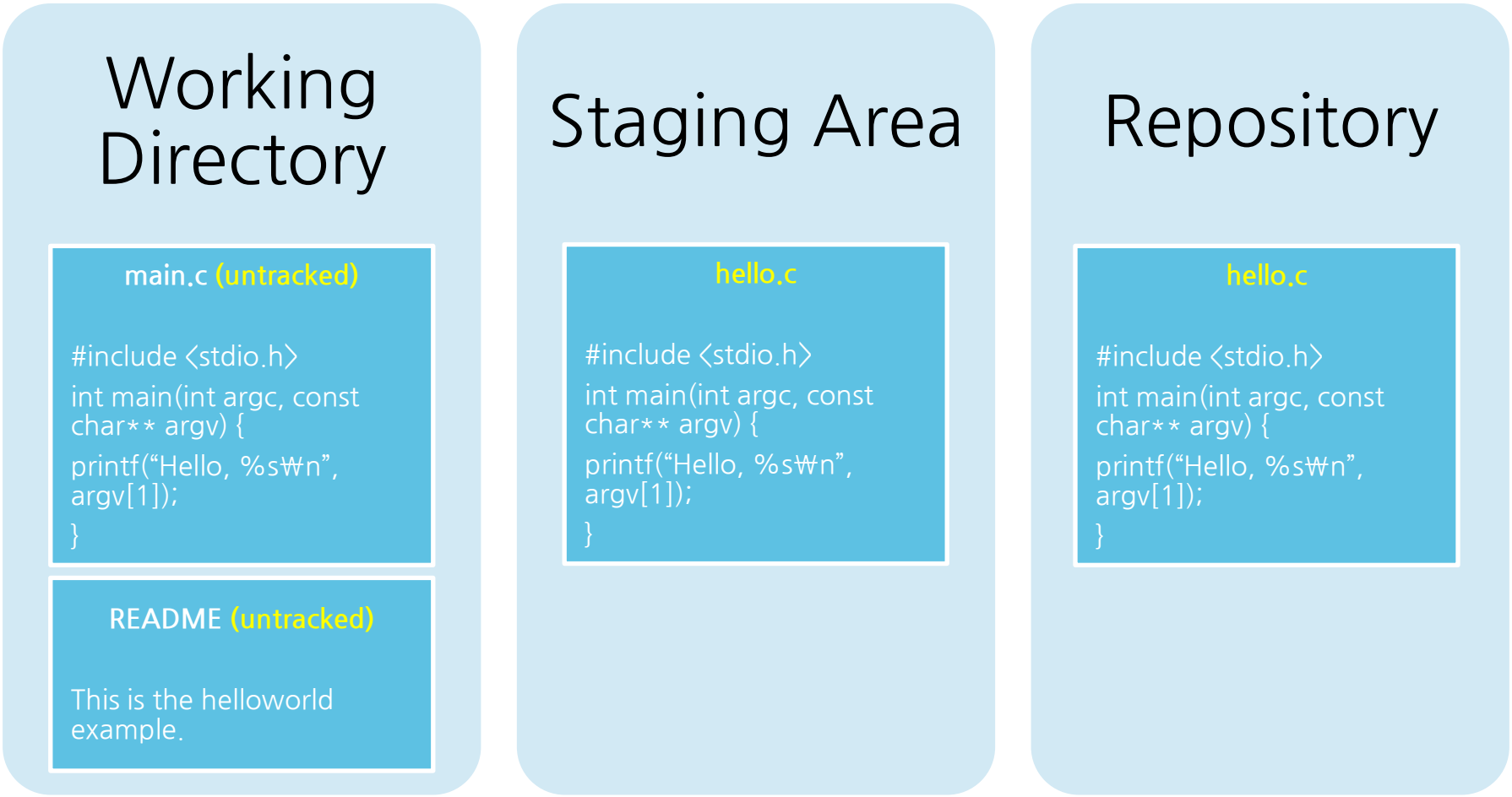
changes to be commited: renamed: hello.c -> main.c

4.3 Git Internal - reset



$ git reset --mixed HEAD^

4.3 Git Internal - reset



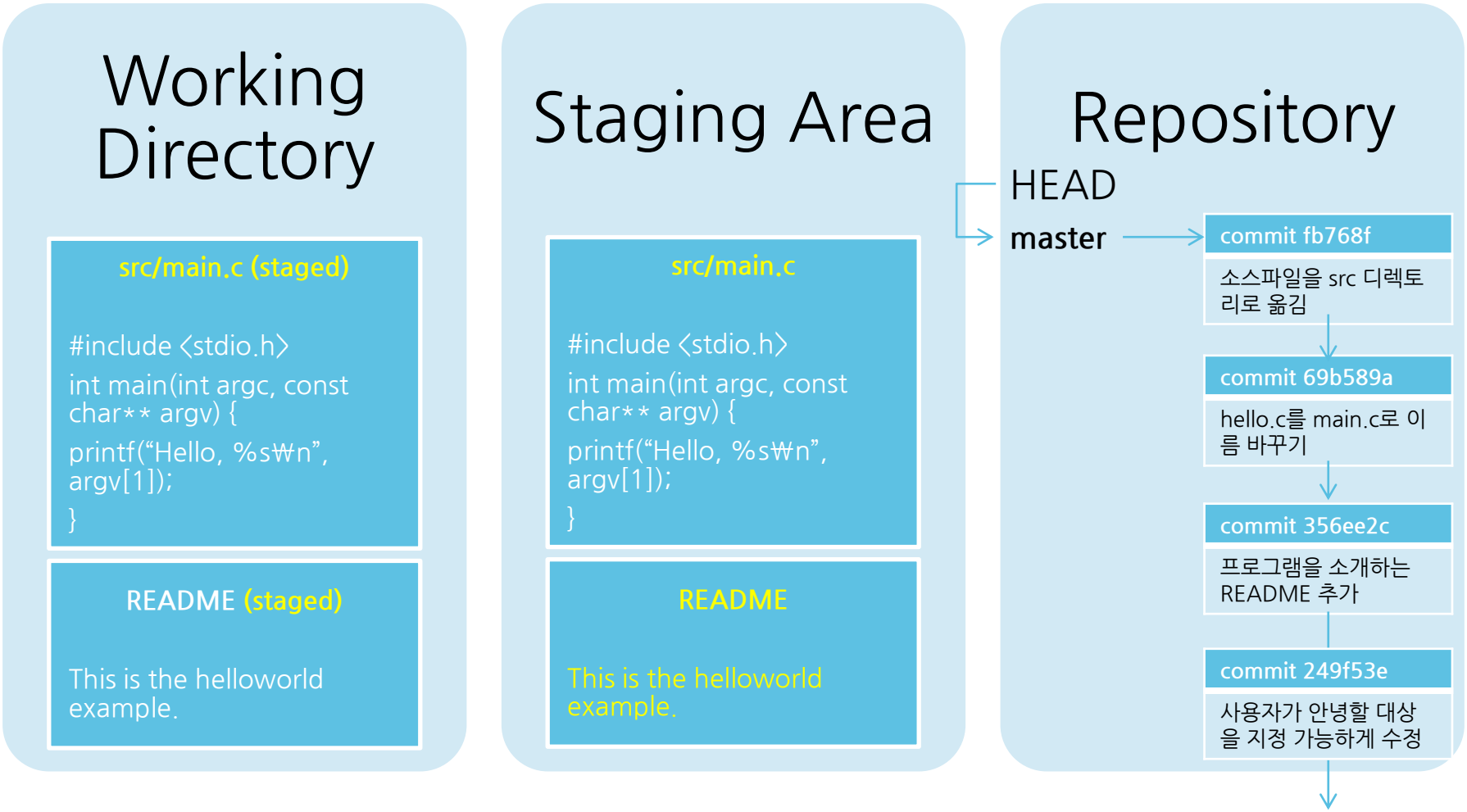
changes not staged for commit: deleted: hello.c

되돌린 것을 되돌리기

```
$ git reset --hard fb768f5
```

되돌릴 커밋 아이디를 확인한 뒤 되돌리기

```
$ git reflog
249f53e HEAD@{0}: reset: moving to HEAD^
69b589a HEAD@{1}: reset: moving to HEAD^
356ee2c HEAD@{2}: reset: moving to HEAD^
fb768f5 HEAD@{3}: commit: 소스파일을 src 디렉토리로
옮김
356ee2c HEAD@{4}: commit: hello.c를 main.c로 이름
바꾸기
…
$ git reset --hard HEAD@{3}
```



$ git reset --hard fb768f

ref

특정 오브젝트를 가리키는 포인터

head

각 브랜치의 끝을 가리키는 포인터
예: master

HEAD

현재 브랜치의 head를 가리키는 포인터

reflog

head들이 어떻게 변해왔는지에 대한 히스토리

5. 히스토리 살펴보기

로그 보기

```
$ git log [커밋]
```

로그 메시지 검색하기

```
$ git log [커밋] --grep <검색어>
```

각 라인에 대해 누가 언제 고쳤는지 확인하기

```
$ git blame <파일명>
$ git blame <리비전> -- <파일명>
$ git annotate <파일명> [리비전]  # blame을 쓸 것
```

두 커밋 사이의 변경내역 보기

```
$ git diff <커밋>..<커밋>
```

특정 커밋 보기

```
$ git show <커밋>
```

특정 커밋에서의 파일 내용 보기

```
$ git show <커밋>:<파일명>
```

5.2 실습 - log, blame, diff, show

```
$ git log
$ git blame src/main.c
$ git blame HEAD^ -- main.c
$ git diff HEAD^..HEAD
$ git show HEAD
$ git show HEAD^:main.c
$ git show HEAD~3:hello.c
```

6. 브랜치와 머지

브랜치 만들기

```
$ git branch <브랜치 이름>
```

다른 브랜치로 이동하기

```
$ git checkout <브랜치 이름>
```

브랜치 합치기

```
$ git merge <브랜치 이름>
```

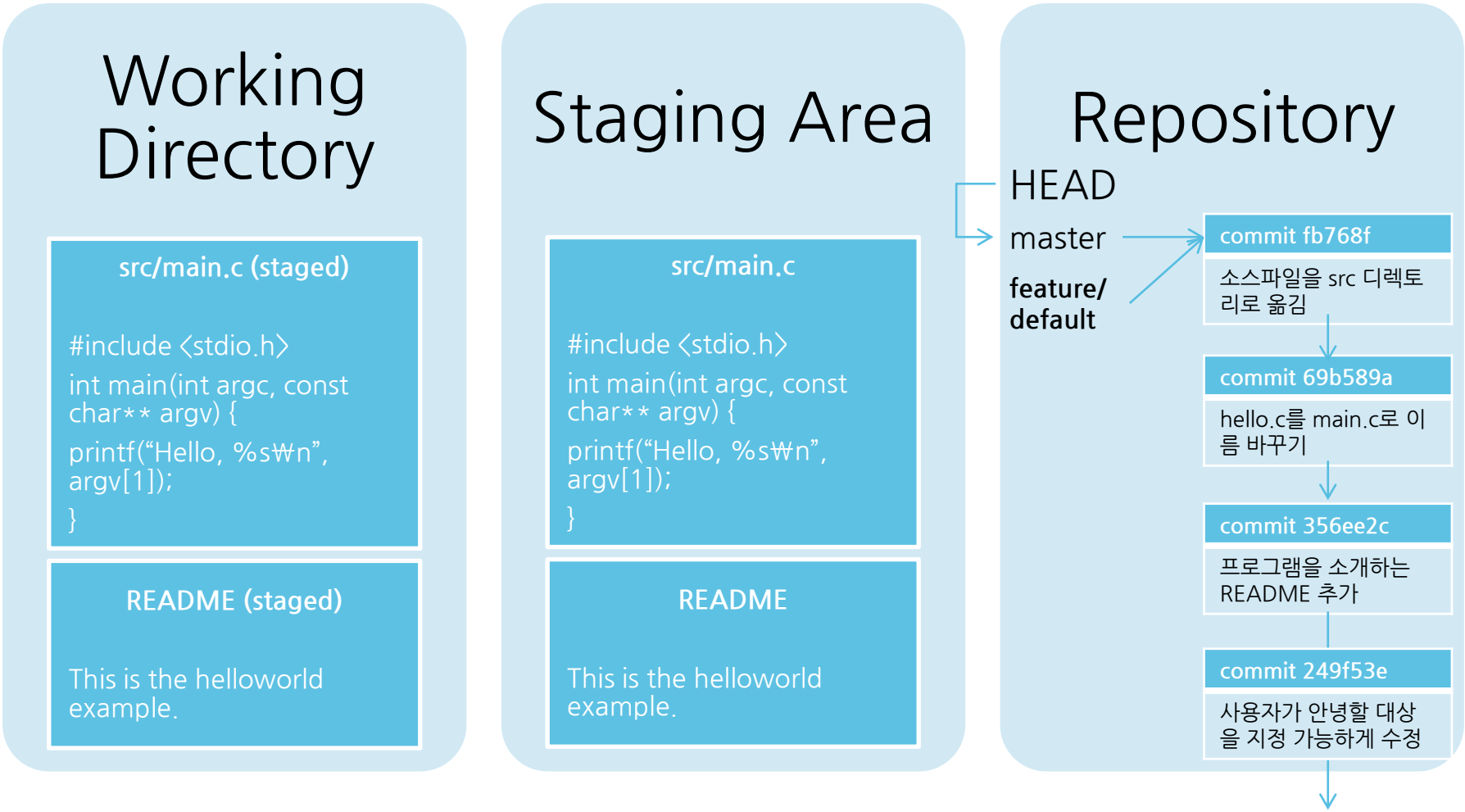
브랜치 삭제하기

```
$ git branch -d <브랜치 이름>
```

6.2 실습 - branch, checkout, merge

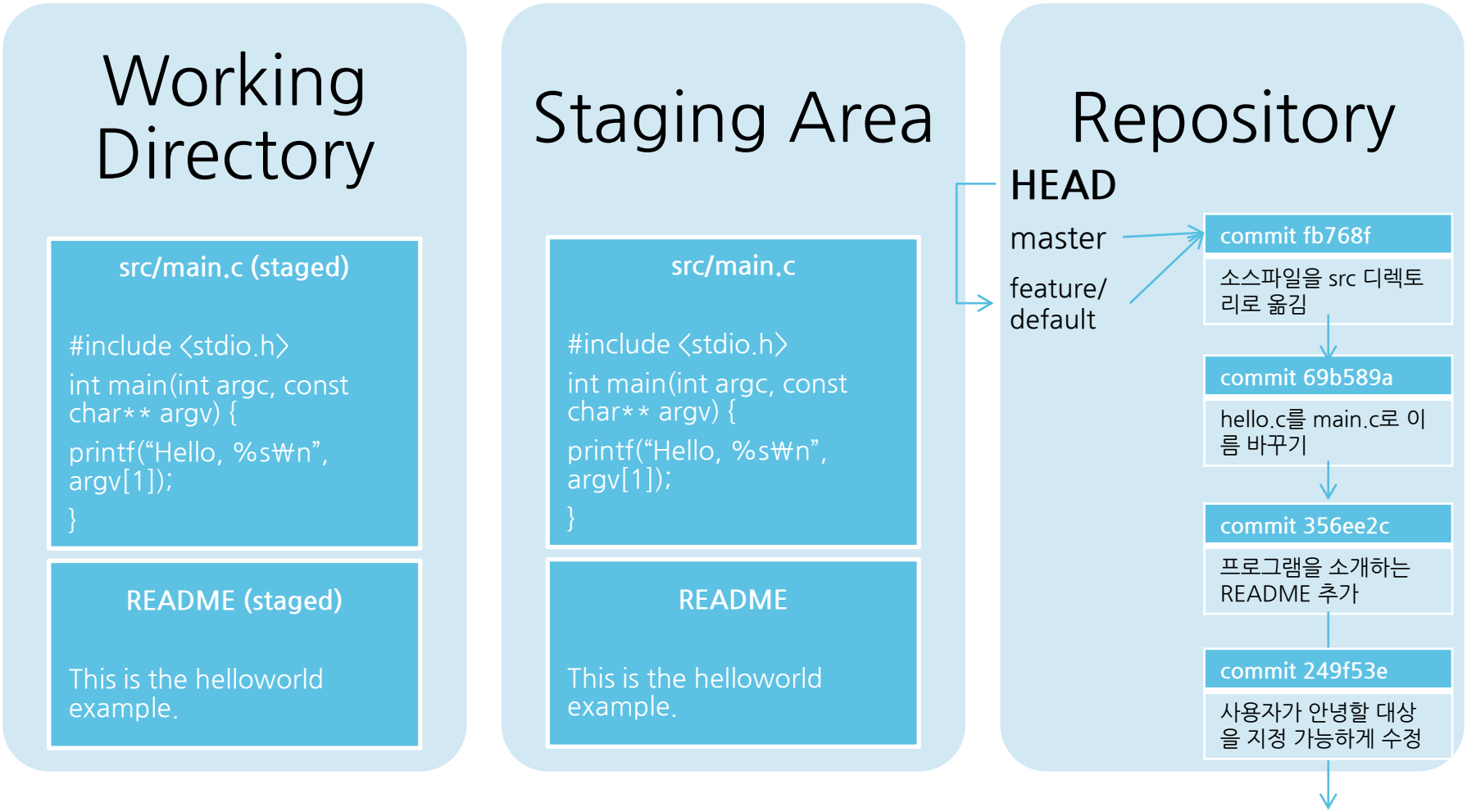
```
$ git branch feature/default        # feature/default 브랜치 만들기
$ git checkout feature/default      # feature/default 브랜치로 이동
$ echo '#include <stdio.h>

int main(int argc, const char** argv) {
    const char* name = argc > 1 ? argv[1]: "World";
    printf("Hello, %s\n", name);
    return 0;
}' > src/main.c
$ git commit -am ' 안녕할 대상에 대한 디폴트값 설정'
$ git checkout master               # master 브랜치로 돌아옴
$ echo 'all:
   gcc hello.c -o hello' > Makefile
$ git add Makefile
$ git commit -m 'Makefile 추가'
$ git merge feature/default         # 현재 브랜치(master)에 feature/default 브랜치를 머지함
$ git branch -d feature/default     # 필요없어진 feature/default 브랜치 삭제
```

6.3 Git Internal - branch, checkout, merge



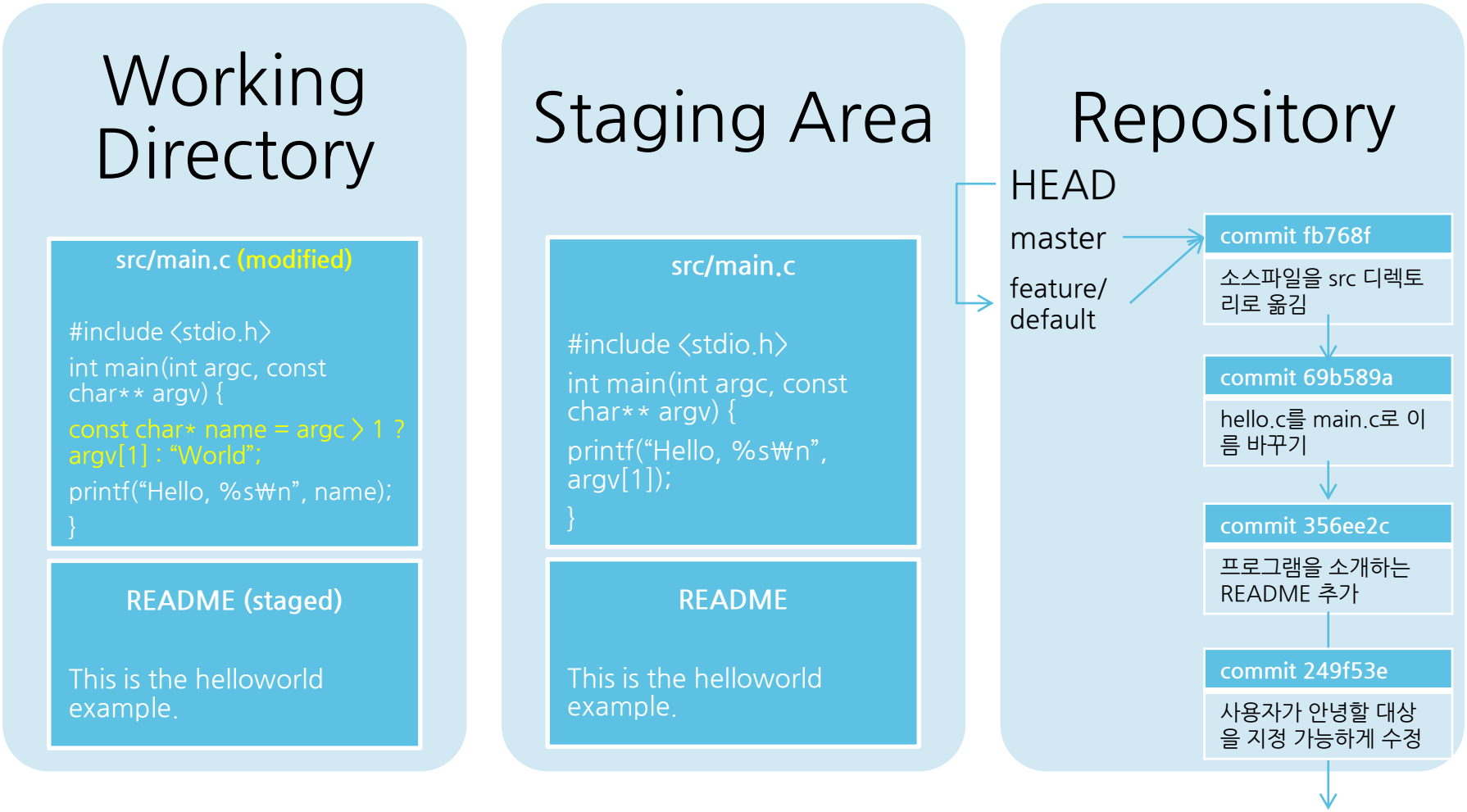
$ git branch feature/default

6.3 Git Internal - branch, checkout, merge



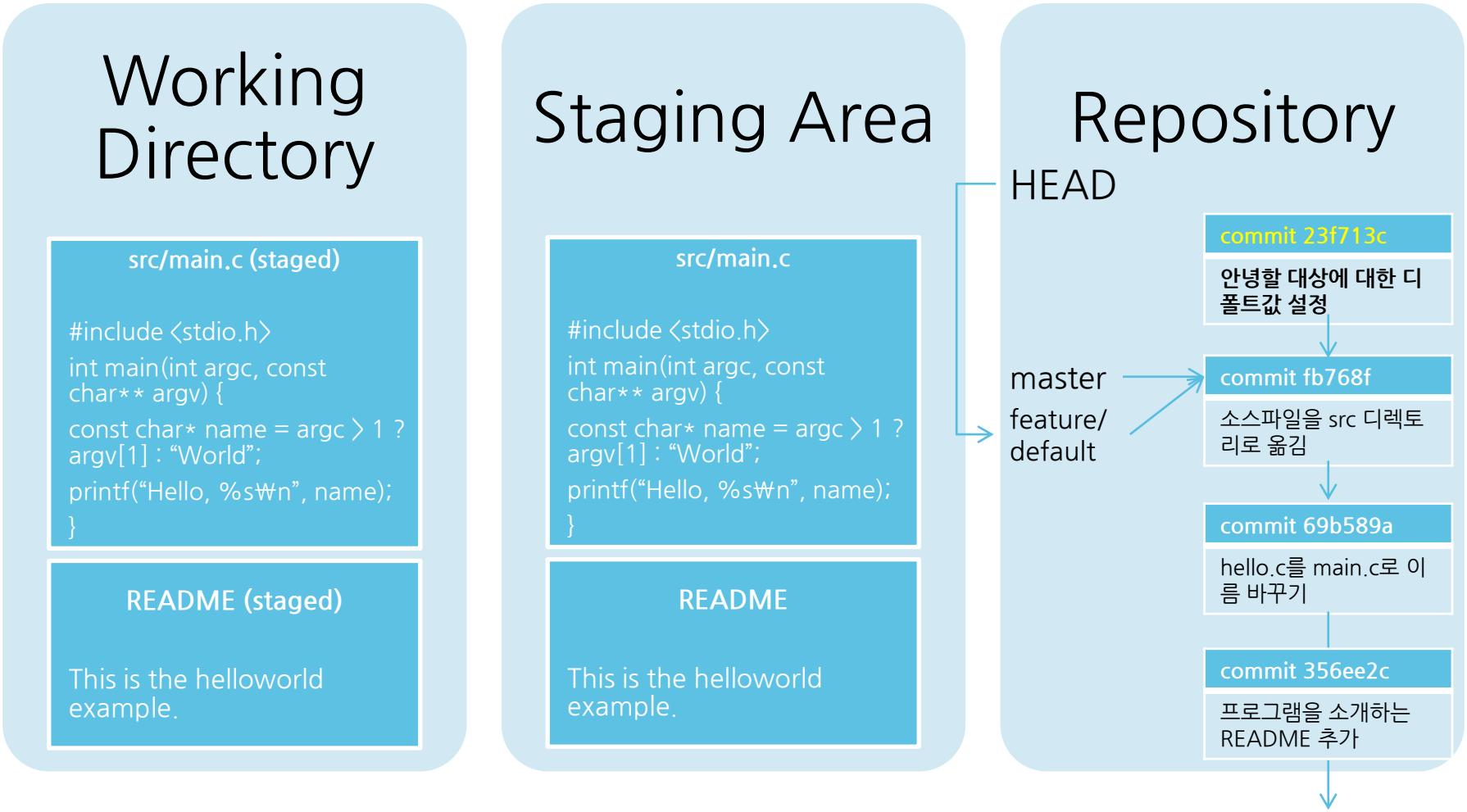
$ git checkout feature/default

6.3 Git Internal - branch, checkout, merge



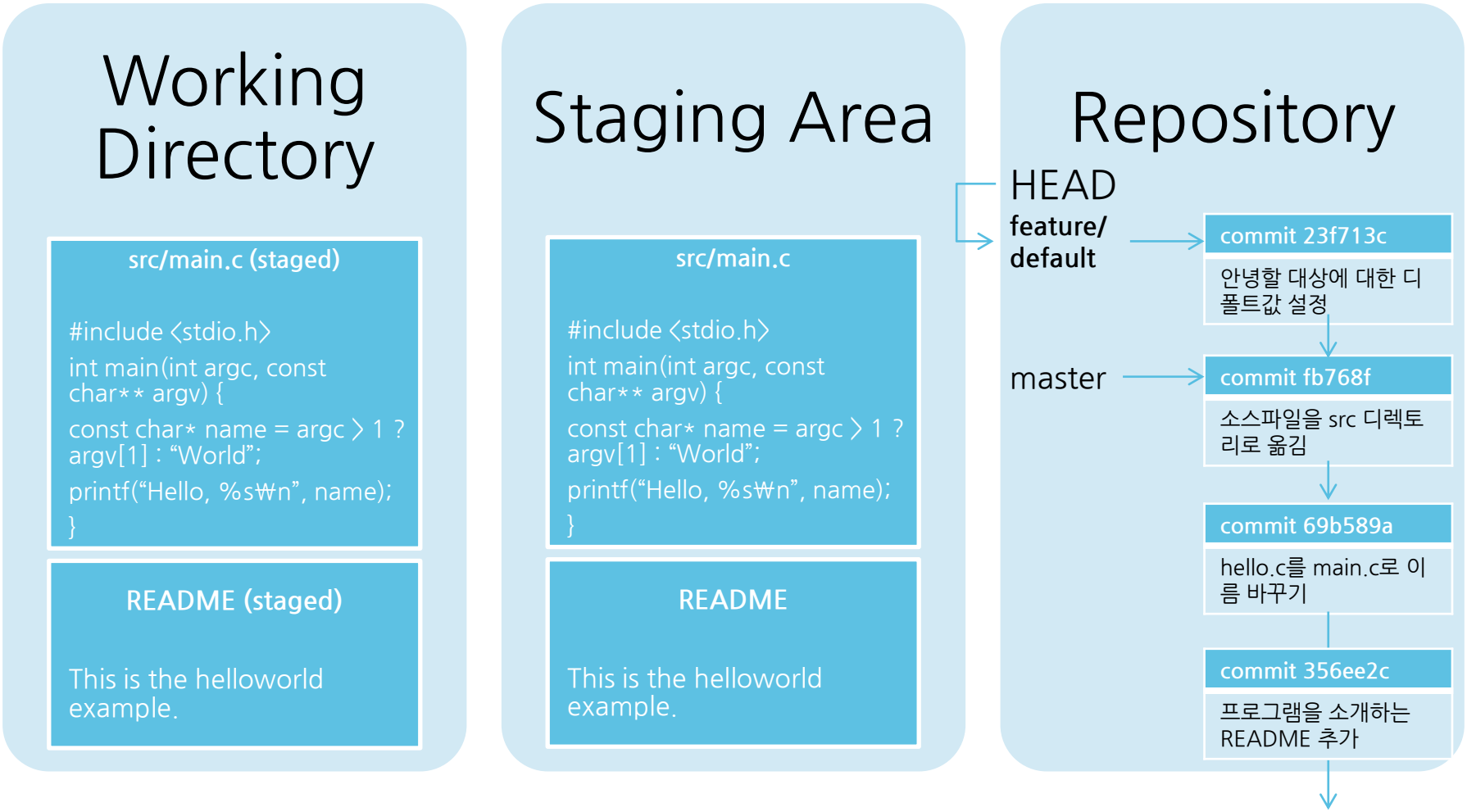
src/main.c 편집

6.3 Git Internal - branch, checkout, merge



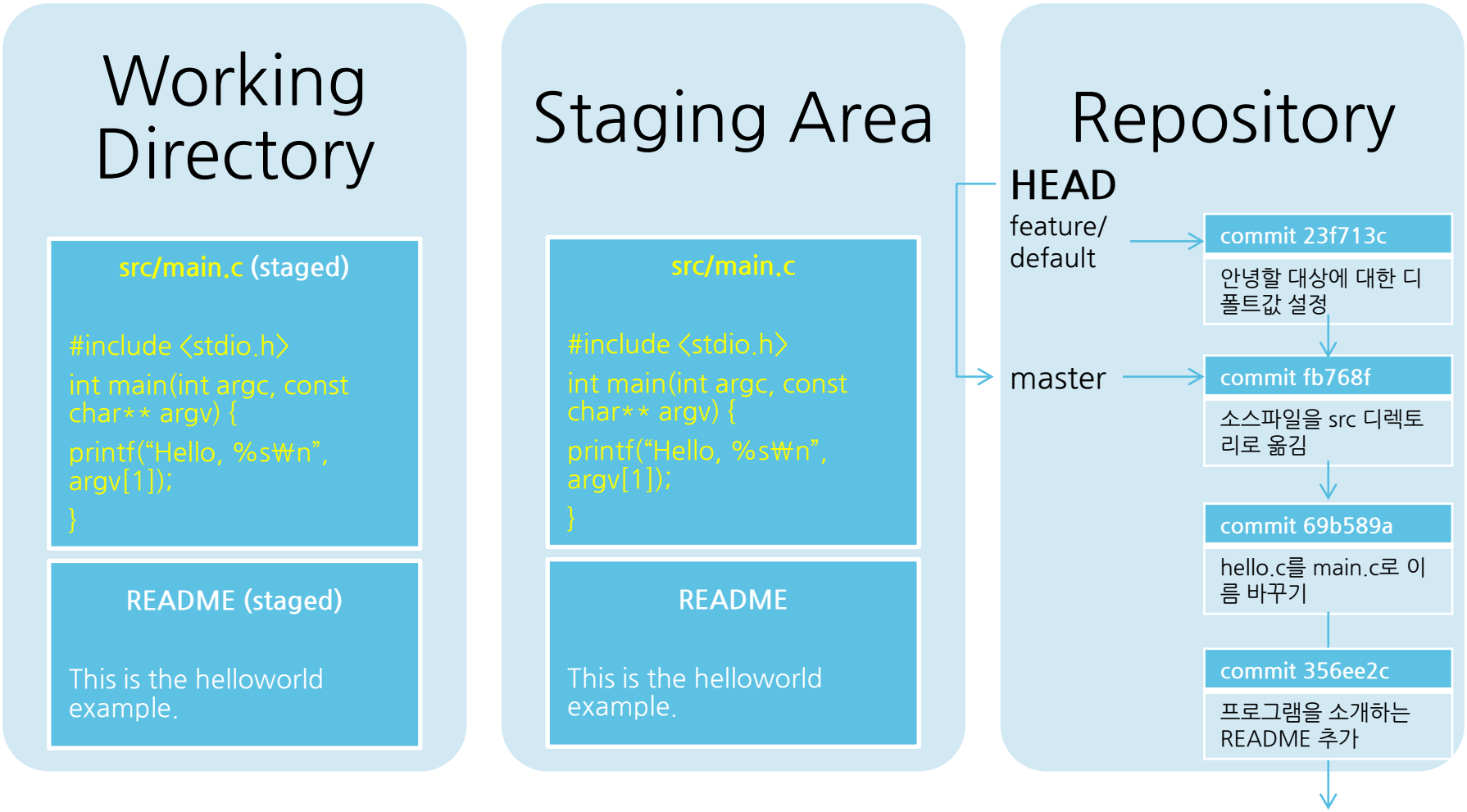
$ git commit -am '안녕할 대상에 대한 디폴트값 설정'

6.3 Git Internal - branch, checkout, merge



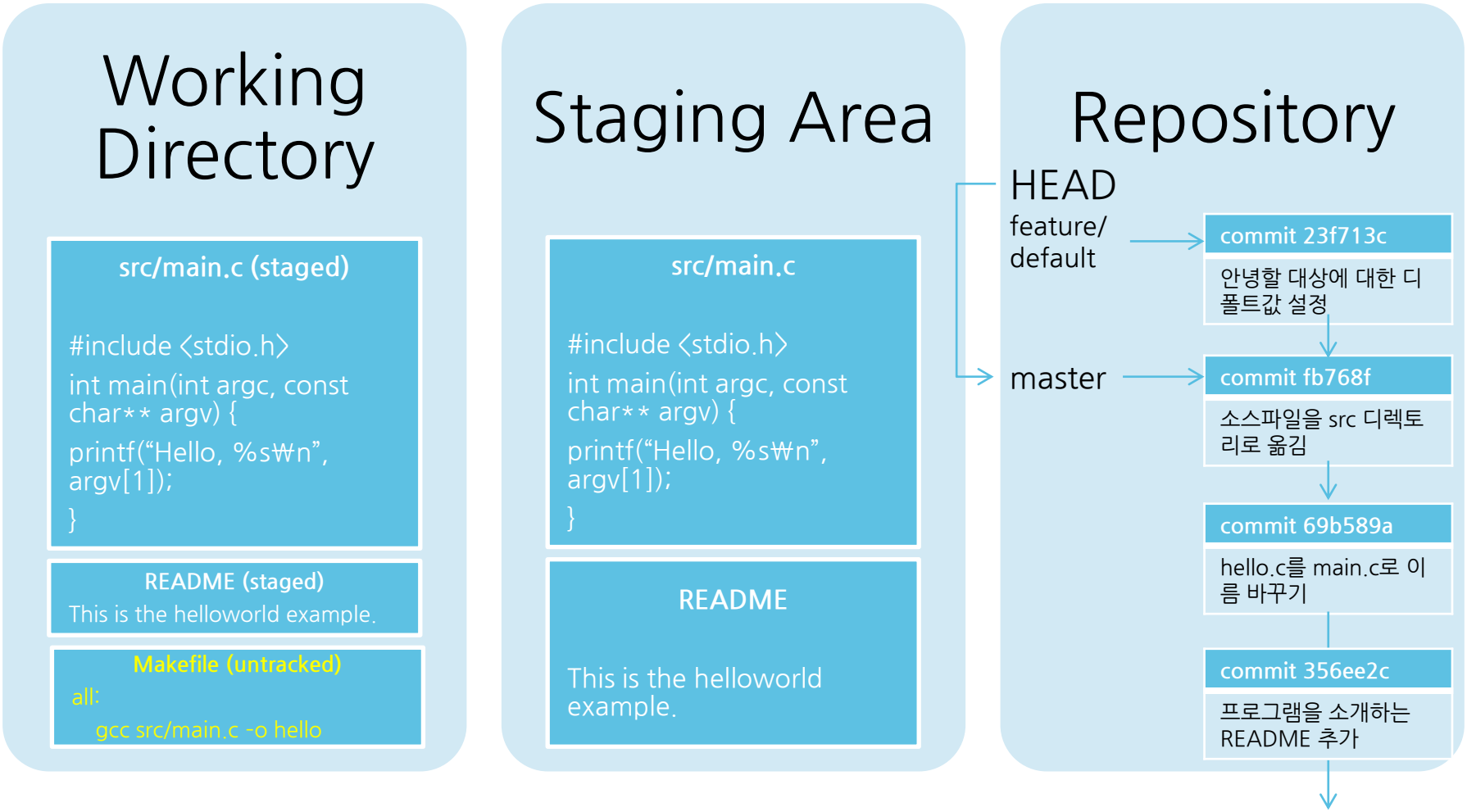
$ git commit -am '안녕할 대상에 대한 디폴트값 설정'

6.3 Git Internal - branch, checkout, merge



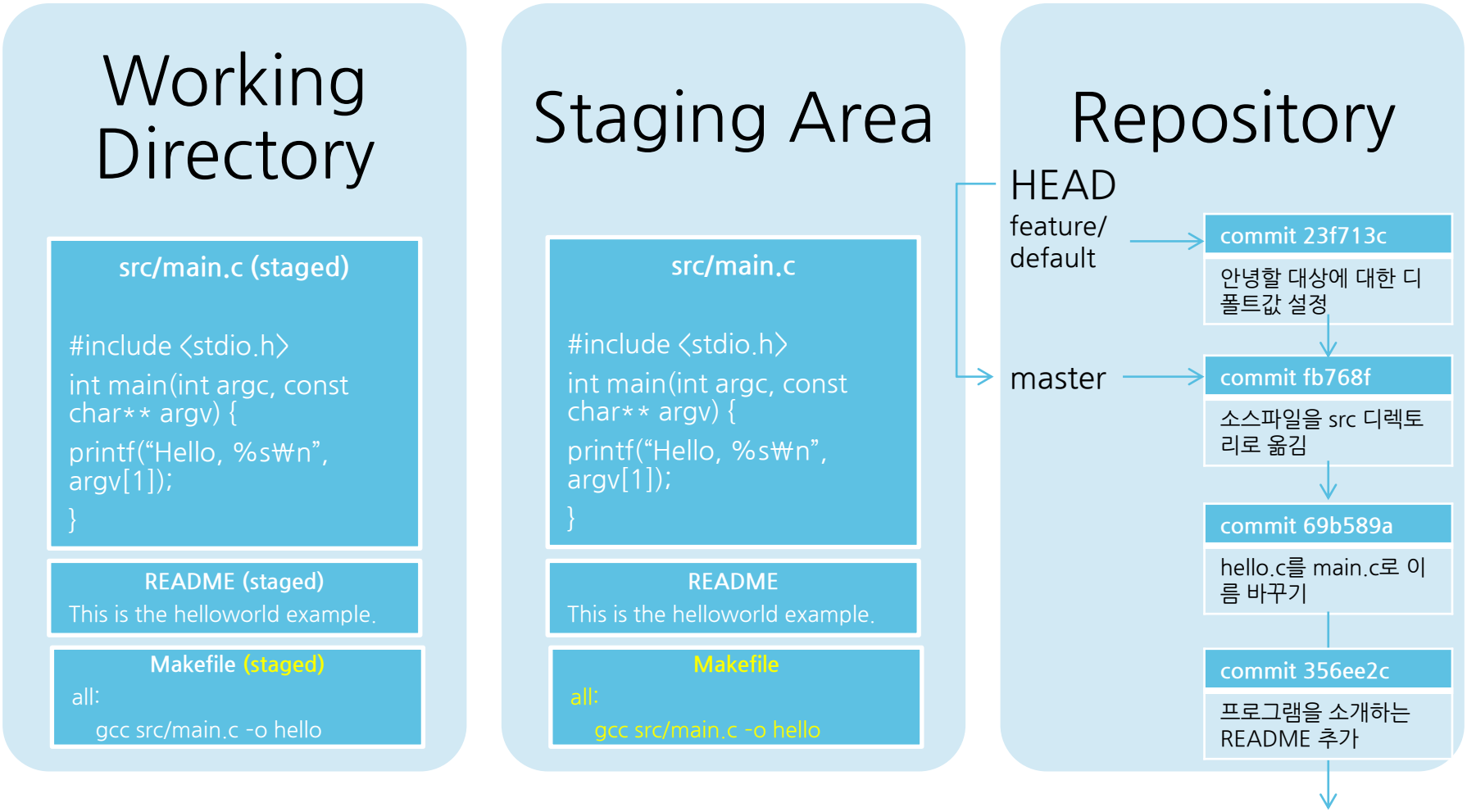
$ git checkout master

6.3 Git Internal - branch, checkout, merge



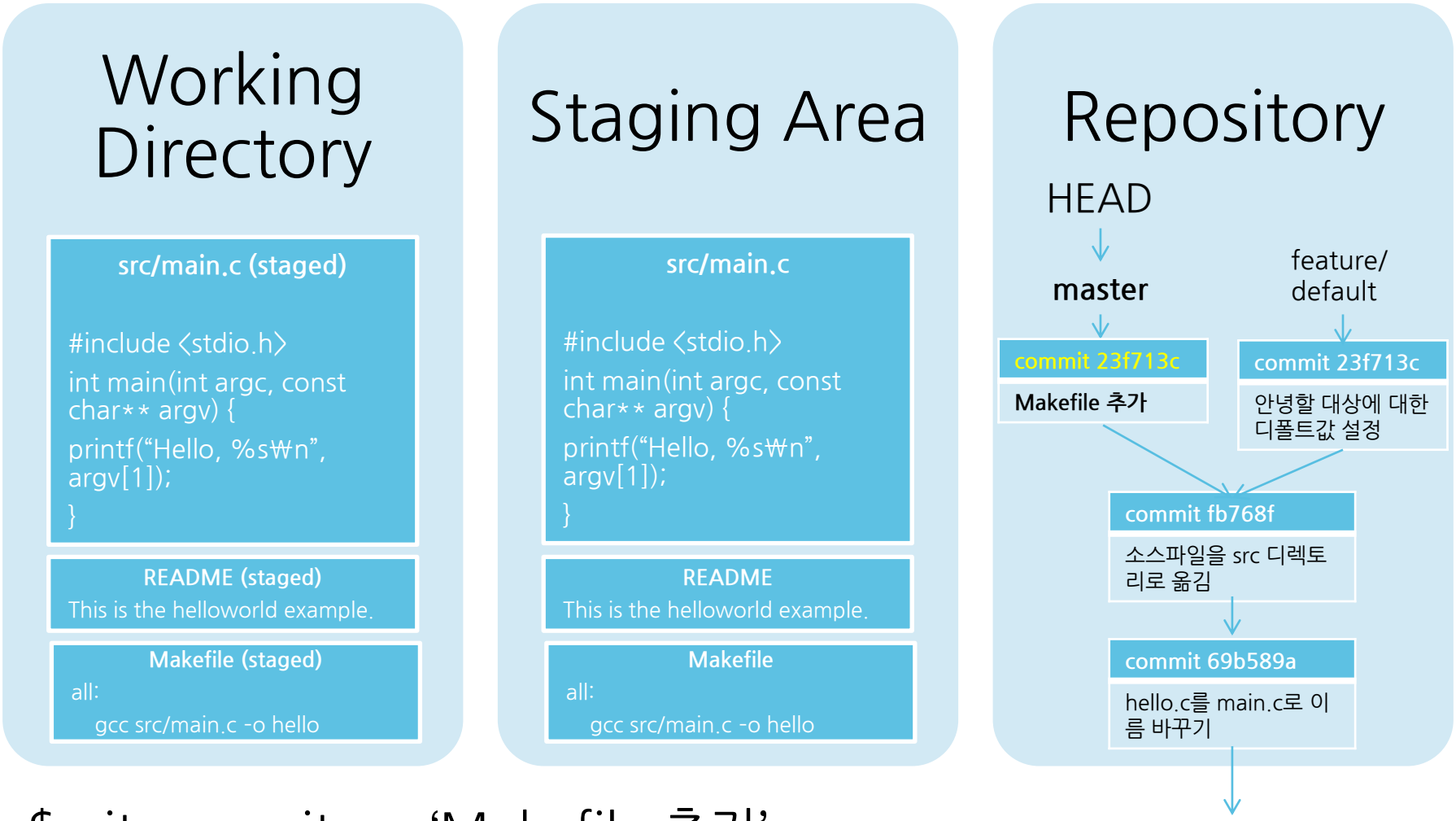
Makefile 작성

6.3 Git Internal - branch, checkout, merge



$ git add Makefile

6.3 Git Internal - branch, checkout, merge



$ git commit -m 'Makefile 추가'

6.3 Git Internal - branch, checkout, merge

Working Directory

src/main.c (staged)

```
#include <stdio.h>
int main(int argc, const char** argv) {
const char* name = argc > 1 ? argv[1] : "World";
printf("Hello, %s₩n", name);
}
```

README (staged)

This is the helloworld example.

Makefile (staged)

```
all:
    gcc src/main.c -o hello
```

Staging Area

src/main.c

```
#include <stdio.h>
int main(int argc, const char** argv) {
const char* name = argc > 1 ? argv[1] : "World";
printf("Hello, %s₩n", name);
}
```

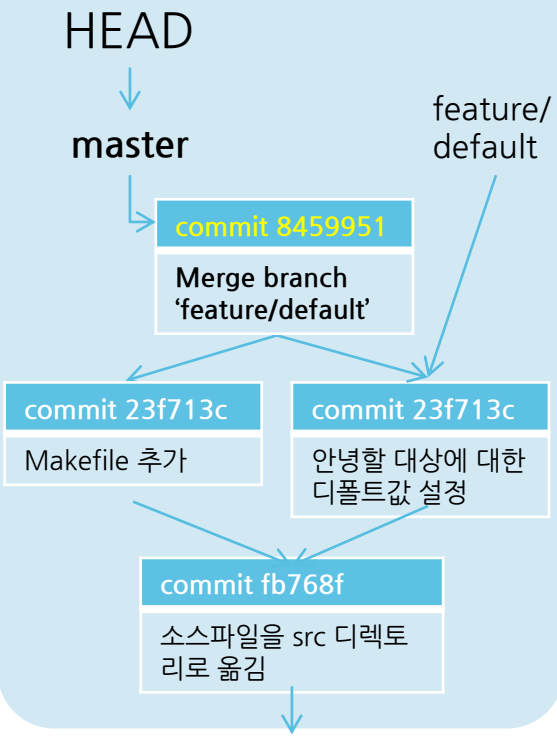
README

This is the helloworld example.

Makefile

```
all:
    gcc src/main.c -o hello
```

Repository



$ git merge feature/default

6.3 Git Internal - branch, checkout, merge

Working Directory

src/main.c (staged)

```
#include <stdio.h>
int main(int argc, const char** argv) {
const char* name = argc > 1 ? argv[1] : "World";
printf("Hello, %s₩n", name);
}
```

README (staged)

This is the helloworld example.

Makefile (staged)

```
all:
    gcc src/main.c -o hello
```

Staging Area

src/main.c

```
#include <stdio.h>
int main(int argc, const char** argv) {
const char* name = argc > 1 ? argv[1] : "World";
printf("Hello, %s₩n", name);
}
```

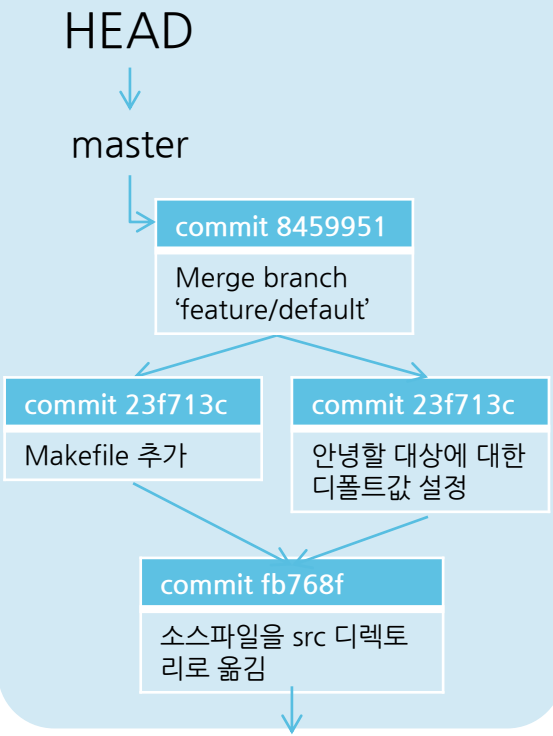
README

This is the helloworld example.

Makefile

```
all:
    gcc src/main.c -o hello
```

Repository




```
$ git branch -d feature/default
```

머지 과정에서 충돌이 발생했을 때 해결하는 법

1. 편집기로 충돌한 파일을 바르게 고침
2. 고친 파일을 add
3. 커밋

mergetool 로 충돌 해결하기 (사용자가 설정한 툴이 실행됨)

1. `$ git mergetool`
2. 실행된 툴로 충돌한 파일을 바르게 고침
3. 커밋

6.5 실습 - 직접 파일을 편집하여 충돌 해결

```
$ git branch feature/stdin              # feature/stdin 브랜치를 만듦
$ git checkout feature/stdin             # feature/stdin 브랜치로 이동
$ echo '#include <stdio.h>
#include <string.h>

int main(int argc, const char** argv) {
    char name[1024];

    if (argc >= 2) {
        strcpy(name, argv[1]);
    } else {
        scanf("%s", name);
    }

    printf("Hello, %s\n", name);

    return 0;
}' > main.c
$ git commit -am '표준입력에서 name 읽기'
$ git checkout master                    # master 브랜치로 돌아감
```

6.5 실습 - 직접 파일을 편집하여 충돌 해결

```
$ echo '#include <stdio.h>

int main(int argc, const char** argv) {
        const char* name = argc > 1 ? argv[1] : "World";
        int result = printf("Hello, %s\n", name);
        return result >= 0 ? 0 : result;
}' > main.c
$ git commit -am 'Exit Status 반환하기'
$ git log --graph               # 두 브랜치의 상태를 확인
$ git merge feature/stdin       # feature/stdin 브랜치를 현재 브랜치(master)에 머지
# 충돌발생. src/main.c 를 편집해서 충돌을 해결함
$ git add src/main.c                # 충돌이 해결된 파일을 Staging Area 에 추가
$ git commit                    # 커밋로그를 저장하여 커밋하면 머지가 완료됨
```

6.6 실습 - mergetool로 충돌해결

```
$ git reset HEAD^ --hard        # mergetool 을 써서 다시 merge를 해보기위해 reset
$ git merge feature/stdin       # 머지 시도
$ git mergetool                 # mergetool 로 충돌 해결
$ git commit                    # 커밋로그를 저장하여 커밋하면 머지가 완료됨
```

p4merge 다운로드 받아 설치한 뒤,

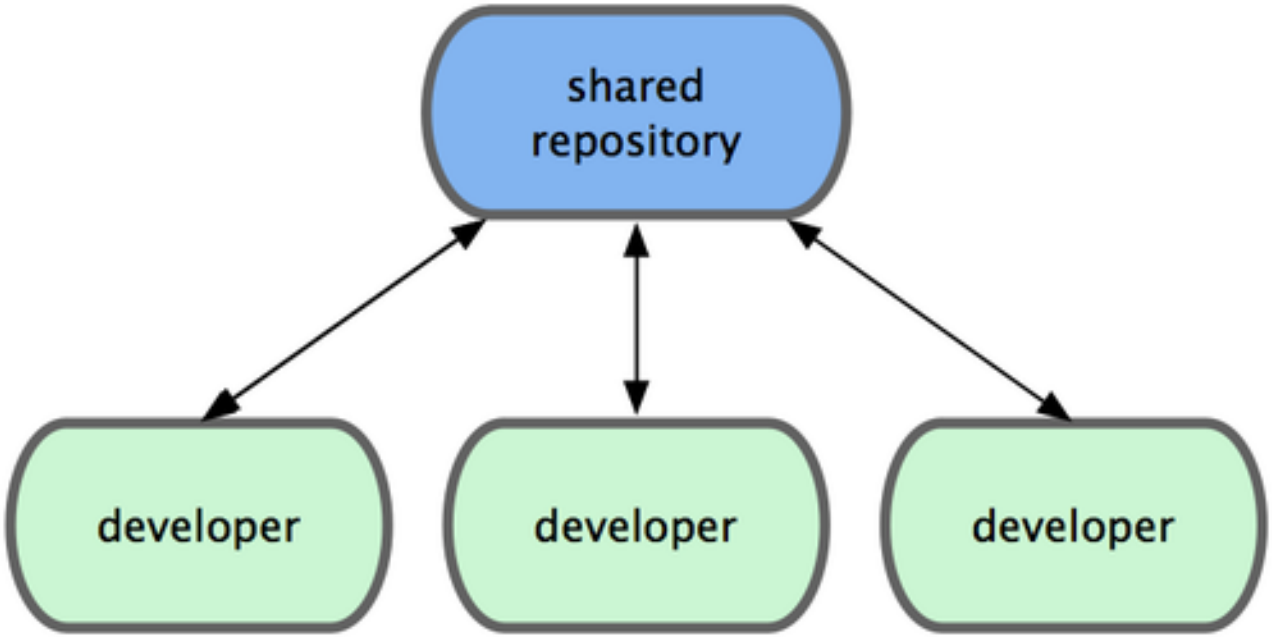
http://www.perforce.com/downloads/complete_list

p4merge를 기본 mergetool로 설정 (.gitconfig 편집)

```
[merge]
        keepBackup = false
        tool = custom
[mergetool "custom"]
        cmd =
/Applications/p4merge.app/Contents/Resources/launchp4merge
"$PWD/$BASE" "$PWD/$REMOTE" "$PWD/$LOCAL" "$PWD/$MERGED"
        keepTemporaries = false
        trustExitCode = false
        keepBackup = false
```

7. 공유하기

7.1 중앙집중식 Workflow



Scott Chacon(2009). Pro Git

내 저장소의 브랜치를 다른 저장소의 브랜치에 덮어쓰기

```
$ git push <저장소 url> <내 브랜치>:<상대방 브랜치>
```

보통은 다른 저장소를 '원격 저장소'로 추가하여 사용

```
$ git remote add <저장소 이름> <저장소 url>
$ git push <저장소 이름> <내 브랜치>:<상대방 브랜치>
```

브랜치/저장소 생략 가능

```
$ git push repo master # master:master 와 같음
$ git push repo # 원격 저장소에 내 저장소와 같은
                # 이름의 브랜치가 있으면 모두 push
$ git push      # git push origin 과 같음
```

다른 저장소의 브랜치를 내 저장소의 브랜치에 머지

```
$ git pull <저장소 url> <상대방 브랜치>:<내 브랜치>
```

보통은 다른 저장소를 '원격 저장소'로 추가하여 사용

```
$ git remote add <저장소 이름> <저장소 url>
$ git pull <저장소 이름> <상대방 브랜치>:<내 브랜치>
```

브랜치/저장소 생략 가능

```
$ git pull repo master  # 원격 저장소의 master를
                        # 현재 작업중인 branch에 머지
$ git pull              # .git/config의 설정대로 동작
```

다른 저장소를 내 로컬에 복제

```
$ git clone <저장소 url>
```

복제된 저장소에는 원격 저장소 및 설정이 추가됨

```
$ git remote  -v
origin <저장소 url> (fetch)
origin <저장소 url> (push)
$ cat .git/config
…
[remote "origin"]
        fetch = +refs/heads/*:refs/remotes/origin/*
        url = <저장소 url>
[branch "master"]
        remote = origin
        merge = refs/heads/master
```

일반적인 작업 순서

```
$ git clone http://devcode.nhncorp.com/git/test.git
$ cd test

…코드 편집…

$ git push
To origin
 ! [rejected]        master -> master (non-fast-forward)
$ git pull

…충돌이 발생했다면 수정 후 커밋…

$ git push
```

Git 호스팅 서비스

이름	Private	URL
GitHub	유료	http://github.com
Bitbucket	무료	http://bitbucket.com
네이버 개발자센터	무료	http://dev.naver.com
DevCode (사내)	무료	http://devcode.nhncorp.com

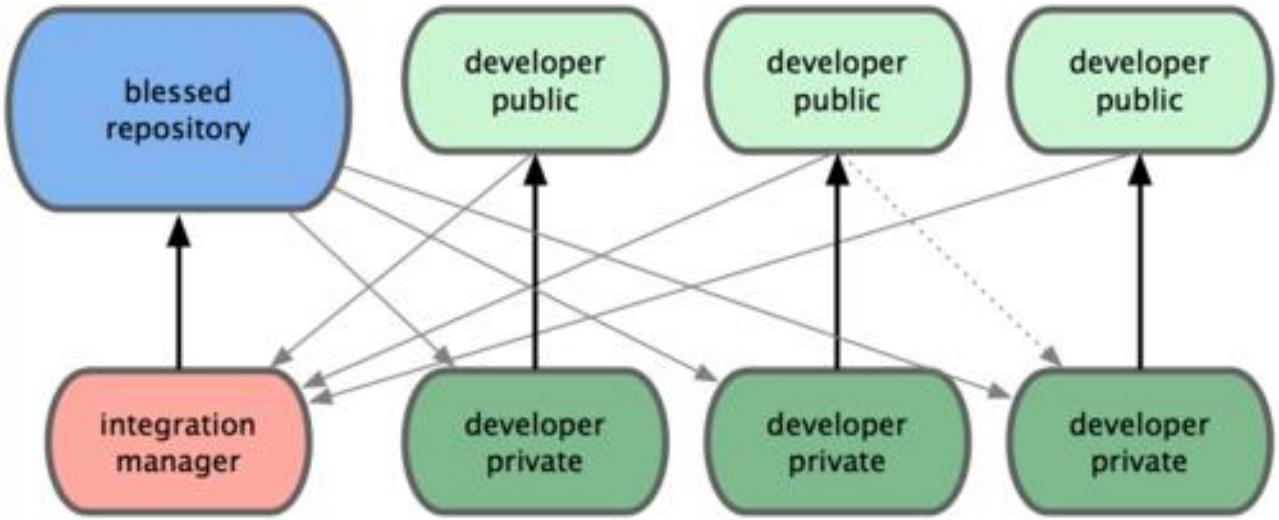
DevCode 에서 코드저장소를 사용하려면

http://devcode.nhncorp.com 접속 >
나의 프로젝트 등록 배너 클릭 >
프로젝트 등록 폼을 채운 후 확인 버튼 클릭 >
생성된 프로젝트의 ‘코드’ 메뉴에서 clone url 확인

7.7 실습

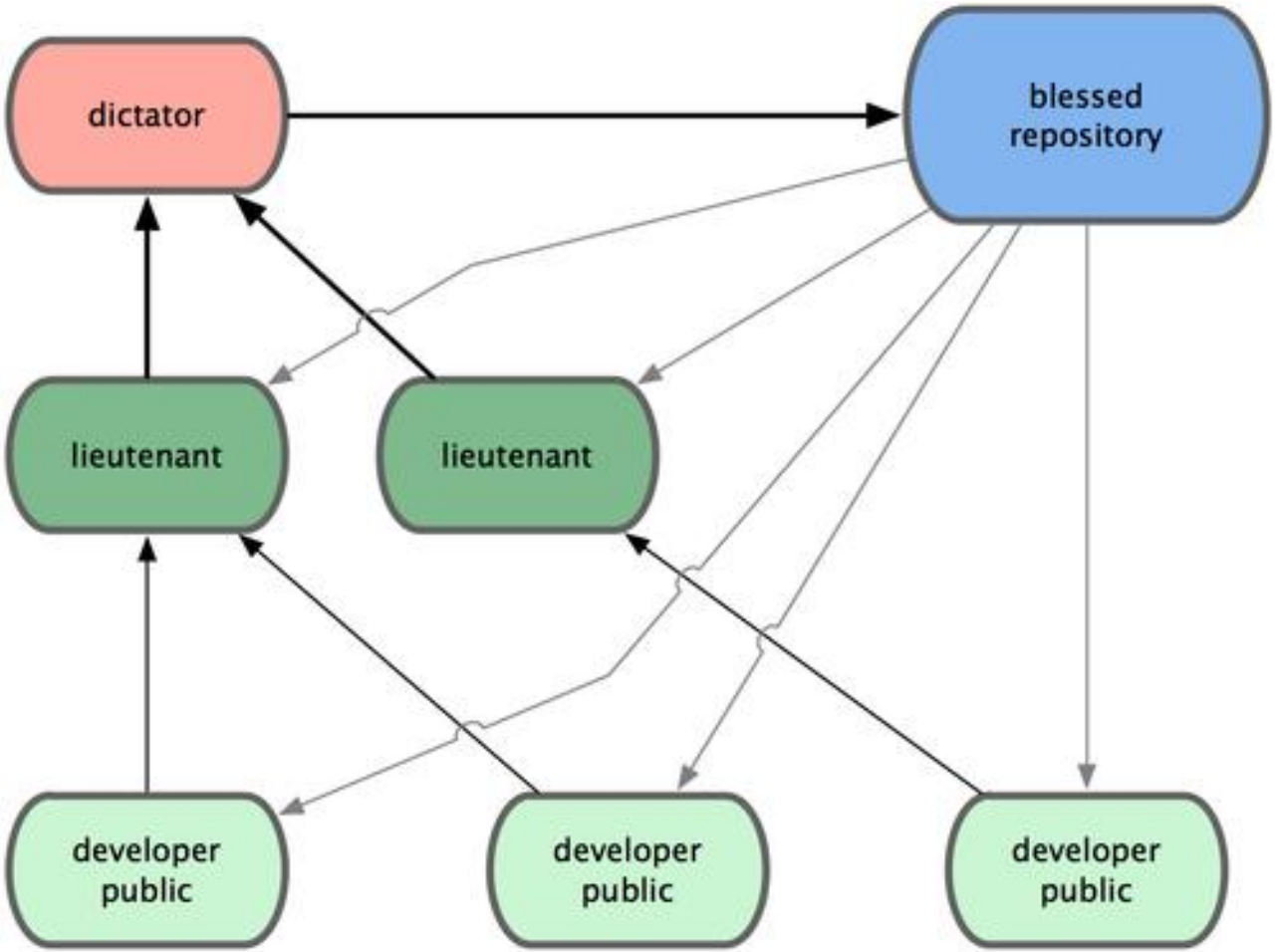
```
# 원격 저장소의 url이 http://devcode.nhncorp.com/git/git-tutorial.git 인 경우

$ git remote add myrepo http://devcode.nhncorp.com/git/git-tutorial.git  # myrepo 라는 이름으로 원격저장소 추가
$ git push myrepo master:yourname                                        # myrepo 저장소의 yourname 브랜치에 로컬의 master 브랜치를 머지
$ cd ..
$ git clone http://devcode.nhncorp.com/git/git-tutorial.git              # sebok-git 저장소를 내 로컬에 복제
$ cd sebok-git                                                           # sebok-git 저장소로 가서
$ git checkout yourname                                                  # yourname 브랜치로 이동
$ git log                                                                # push한 코드가 잘 들어갔는지 확인
$ cd ../hello                                                            # 다시 원래 저장소로 돌아옴
$ echo ‘myname <myname@mail.com>' > AUTHORS                              # AUTHORS 파일을 작성하고
$ git commit -m ‘저자 목록이 담긴 AUTHORS 파일 작성‘                      # 커밋
$ git push myrepo master:yourname                                        # 커밋한 것을 원격저장소에 push
$ cd ../sebok-git                                                        # clone 한 저장소에 가서
$ git pull origin yourname:yourname                                      # pull 명령으로 원격 저장소의 코드를 가져옴
$ git log                                                                # pull 이 잘 되었는지 확인
```

7.8 Integration-Manager Workflow



Scott Chacon(2009). Pro Git

7.9 Dictator and Lieutenants Workflow



Scott Chacon(2009). Pro Git

8. 태깅

태그란?

어떤 Git Object(주로 커밋)에 대한 참조.
사용자가 임의로 만들거나 삭제할 수 있다.

태그 붙이기

```
$ git tag <태그이름> <커밋>
```

태그 지우기

```
$ git tag -d <태그이름>
```

태그 목록 보기

```
$ git tag
```

태그는 커밋에 대한 참조이므로, 커밋 아이디처럼 활용가능

```
$ git checkout <태그이름>
$ git reset <태그이름>
$ git log <태그이름>
$ git show <태그이름>
```

8.2 실습 - tag

```
$ git tag tutorial-8                # 현재 HEAD 에 tutorial-6.5 라는 태그를 붙임
$ git tag tutorial-6.5 HEAD^^       # HEAD^^ 에 yourname-b 라는 태그를 붙임
$ git tag                           # 태그들이 생겨났는지 확인
$ git log --decorate=full           # 태그들이 제대로 붙어있는지 확인
$ git push --tags myrepo            # 태그들을 원격저장소 ‘myrepo' 로 push
$ git tag -d tutorial-8             # tutorial-8 태그를 삭제
$ git push myrepo :tutorial-8       # 원격 저장소에서도 tutorial-8 태그 삭제
```

9. 편리한 기능들

버그 찾기 - git bisect

```
$ git bisect start      # 시작
$ git bisect bad        # 버그가 있다면 bad 표시
$ git checkout HEAD~10  # 버그가 없었던 지점으로
$ git bisect good       # 버그가 없다면 good 표시
Bisecting: 0 revisions left to test after this
(roughly 4 step)
[78f875a749fac361ea697ca5b49f182d95f05ec1] test2
$ git bisect bad      # 버그가 있다면 bad 표시
…
b976c23a22900e7fd97b9015974209de153ade2e is the
first bad commit
$ git bisect reset    # 문제점을 찾았으면 종료
$ git bisect good
```

버그 찾기 - git bisect run

```
$ git bisect start      # 시작
$ git bisect bad        # 버그가 있다면 bad 표시
$ git checkout HEAD~10  # 버그가 없었던 지점으로
$ git bisect good       # 버그가 없다면 good 표시
$ git bisect run maven test
b976c23a22900e7fd97b9015974209de153ade2e is the
first bad commit
$ git bisect reset    # 문제점을 찾았으면 종료
$ git bisect good
```

현재 커밋에서 코드검색

```
$ git grep <keyword>
```

커밋로그 검색

```
$ git log --grep <keyword>
```

전 커밋의 변경내역에서 코드 검색

```
$ git log -S <keyword>
```

일부분만 add 하기

```
$ git add --edit
```

편집기가 실행되면, 패치파일에서 적용할 부분만 남기고 저장

커밋 다시하기

```
$ git commit --amend
```

편집기가 실행되면, 커밋로그를 편집하고 저장
(현재 Staging Area에 들어있는 변경내용도 커밋됨)

작업중인 내용 저장해두기

```
$ git stash
```

(-u 옵션을 주지 않으면 Git이 추적하는 파일만 저장)

저장한 내용 복원하기

```
$ git stash pop
```

저장된 내역 확인

```
$ git stash list
```

유용한 설정들

```
$ git config --global core.editor vim
$ git config --global alias.co checkout
```

특정 파일 무시하기

.gitignore 에 무시할 파일의 glob 패턴을 적음

```
$ cat .gitignore
*.a
build/
```

10. 더 공부하기

책

Pro Git, written by Scott Chacon
역서: http://dogfeet.github.com/progit/progit.ko.pdf
(시간이 없다면 2장만이라도)

메일링 리스트

http://vger.kernel.org/vger-lists.html#git
(단점: 영어)

https://groups.google.com/forum/?fromgroups#!forum/git-ko
(한국어)

Thank you.

A. 라이선스