



6

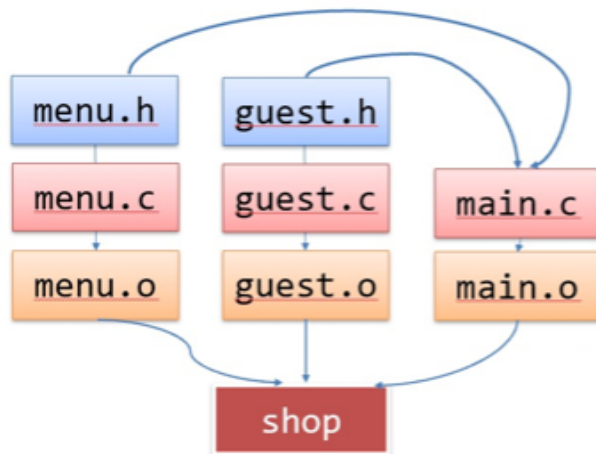
make utility



make utility

- 소프트웨어 개발을 위해 유닉스 계열 운영 체제에서 주로 사용 되는 프로그램 빌드 도구
- 여러 파일들의 의존성과 필요한 명령을 정의할 수 있음
- 컴파일 & 실행파일을 생성할 수 있는 표준문법이 있음
- **Makefile**을 실행함
- 장점
 - 파일에 대한 반복적 명령 자동화로 시간 절약
 - 프로그램의 종속 구조를 빠르게 파악하고 관리가 용이
 - 단순 반복 작업 및 재작성을 최소화

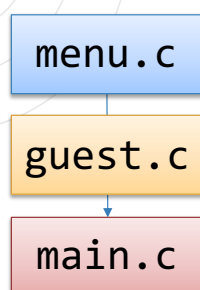
Pizza Ordering System



```
*****
1. Pizza    : 20000
2. Chicken  : 12000
*****
원 하 는 메 뉴 ? 1
Pizza 선택
```

3

Pizza Ordering System



- Compiling libraries

```
walab-HGU:~/20220SS/lab5:> gcc -c menu.c -o menu.o
walab-HGU:~/20220SS/lab5:> gcc -c guest.c -o guest.o
```

- Build(main)

```
walab-HGU:~/20220SS/lab5:> gcc -o pizzashop main.c menu.o guest.o
```

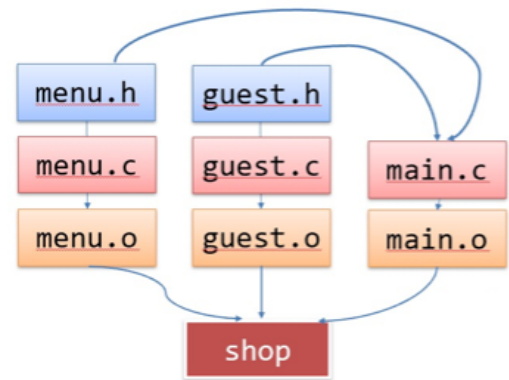
- Run

```
walab-HGU:~/20220SS/lab5:> ./pizzashop
```

4

Pizza Ordering System

- Create header files : menu.h, guest.h
- Create C files : menu.c, guest.c, main.c
- Compile & build
- Run



Pizza Ordering System

- Create header files : menu.h, guest.h

```
1 //menu.h
2 #include <stdio.h>
3
4 void displayMenu();
```

```
1 //guest.h
2 #include <stdio.h>
3
4 int addGuest();
5 void displayGuest(int menu);
```

Pizza Ordering System

- Create C files : menu.c, guest.c, main.c

```
*****
1. Pizza : 20000
2. Spaghetti: 12000
*****
원하는 메뉴는? 1
Pizza 선택
```

```
//menu.c
#include "menu.h"

void displayMenu(){
    printf("*****\n");
    printf("1. Pizza : 20000 \n");
    printf("2. Spaghetti: 12000 \n");
    printf("*****\n");
}
```

```
//guest.c
#include "guest.h"

int addGuest(){
    int menu;
    printf("원하는 메뉴는? ");
    scanf("%d", &menu);
    return menu;
}

void displayGuest(int menu){
    if(menu == 1)
        printf("Pizza 선택");
    else
        printf("Spaghetti 선택");
    printf("\n");
}

//main.c
#include "menu.h"
#include "guest.h"

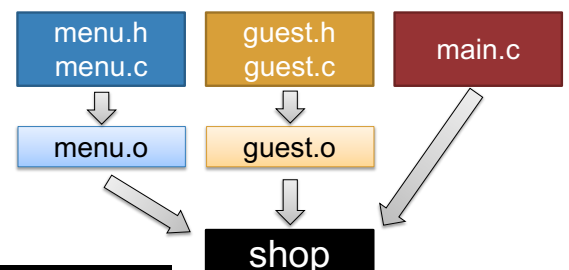
int main(){
    int menu;

    displayMenu();
    menu = addGuest();
    displayGuest(menu);
    return 0;
}
```

7

Pizza Ordering System

- Compile & build
 - menu.c / guest.c
 - main.c



```
walab-HGU:~/lab5:> ls
guest.c guest.h main.c menu.c menu.h
walab-HGU:~/lab5:> gcc -c menu.c
walab-HGU:~/lab5:> gcc -c guest.c -o guest.o
walab-HGU:~/lab5:> gcc main.c guest.o menu.o -o shop
walab-HGU:~/lab5:> ls
guest.c guest.h guest.o main.c menu.c menu.h menu.o shop
walab-HGU:~/lab5:> ./shop
```

8

Makefile

- make utility를 실행하면 Makefile을 읽어들이м
- 구조
 - **Target** : 명령어가 수행된 결과 파일명
 - target / dummy target(결과 파일 없음)
 - **Dependency** : 대상 파일 생성을 위해 필요한 파일
 - 파일 수정 날짜 체크
 - **Command** : 실행할 명령 문장
 - **Macro** : 코드 단순화

```
target1 : dependency1 dependency2 ...  
<tab>command1  
<tab>command2
```

Makefile 실습

```
target1 : dependency1 dependency2 ...  
<tab>command1  
<tab>command2
```

```
shop : main.c menu.o guest.o  
      gcc -o shop main.c menu.o guest.o  
menu.o: menu.c menu.h  
      gcc -c menu.c -o menu.o  
guest.o: guest.c guest.h  
      gcc -c guest.c  
clean:  
      rm *.o shop
```

Makefile

```
walab-HGU:~/lab5:> make clean  
rm *.o shop  
walab-HGU:~/lab5:> make  
gcc -c menu.c -o menu.o  
gcc -c guest.c  
gcc -o shop main.c menu.o guest.o  
walab-HGU:~/lab5:> make clean  
rm *.o shop  
walab-HGU:~/lab5:> make shop  
gcc -c menu.c -o menu.o  
gcc -c guest.c  
gcc -o shop main.c menu.o guest.o  
walab-HGU:~/lab5:> ./shop
```

Macro

- Makefile을 작성할 때 자주 사용하며 문자열을 간단하게 함
- 변수에 특정 문자열을 정의하고 표현하는 방식
- Pre-defined macro

```
walab-HGU:~:> make -p | grep cc
make: *** No targets specified and no makefile found. Stop.
COMPILE.cpp = $(COMPILE.cc)
CC = cc
LINK.cc = $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
YACC = yacc
LINK.C = $(LINK.cc)
LINK.cpp = $(LINK.cc)
COMPILE.cc = $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
COMPILE.C = $(COMPILE.cc)
```

11

Macro 작성방법

- **NAME = 문자열** (관습적으로 NAME은 대문자 사용)
- **#** 주석문
- **** 여러 행에 걸쳐서 문자열 사용할 때
- **\$(NAME)** 매크로를 참조
- 정의되지 않은 매크로를 사용하는 경우 NULL로 치환됨
- 문자열의 따옴표도 문자열로 인식함

12

Macro 예제

```
shop : main.c menu.o guest.o
    gcc -o shop main.c menu.o guest.o
menu.o: menu.c menu.h
    gcc -c menu.c -o menu.o
guest.o: guest.c guest.h
    gcc -c guest.c
clean:
    rm *.o shop
```

Makefile

macro를 적용한 Makefile

```
CC = gcc
shop : main.c menu.o guest.o
    $(CC) -o shop main.c menu.o guest.o
menu.o: menu.c menu.h
    $(CC) -c menu.c -o menu.o
guest.o: guest.c guest.h
    $(CC) -c guest.c
clean:
    rm *.o shop
```

13

Suffix rule

- 파일 확장자를 보고 그에 따라 적절한 명령을 실행함
 - C 소스 파일(*.c), 목적파일(*.o)을 인식
- 목적파일이 없는 경우 같은 이름의 C 소스 파일을 컴파일하여 생성

확장자 규칙을 적용한 Makefile

```
CC = gcc
shop : main.c menu.o guest.o
    $(CC) -o shop main.c menu.o guest.o
clean:
    rm *.o shop
```

14

Special Macros

- `$@` 현재 Target 이름
- `$*` (확장자가 있는 경우에) 확장자가 없는 현재 Target
- `$<` 첫번째 dependency 이름
- `$$` Dependency 전체
- `$?` Dependency 중 현재 target보다 최근에 갱신된 파일

```
CC = gcc
shop : main.c menu.o guest.o
    $(CC) -o shop main.c menu.o guest.o
clean:
    rm *.o shop
```

Special macro를 적용한 Makefile

```
CC = gcc
shop : main.c menu.o guest.o
    $(CC) -o $@ $^
clean:
    rm *.o shop
```

15

Makefile

```
CC = gcc
shop : main.c menu.o guest.o
    $(CC) -o $@ $^
clean:
    rm *.o shop
```

```
CC = gcc
CFLAGS = -W -Wall
TARGET = shop
OBJECTS = main.o menu.o guest.o
all : $(TARGET)
$(TARGET) : $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^
clean:
    rm *.o shop
```

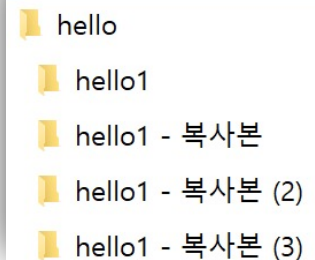
16

7

Git 기초 VCS (Version Control System)

소스 관리는 어떻게?

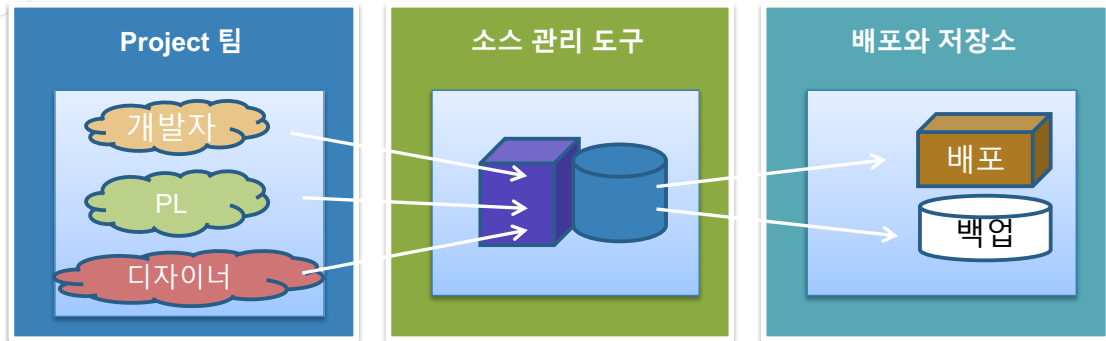
- 개발 완료된 소스는 어떻게 관리하는가?
 - 각 폴더는 무엇을 위한 버전인가?
 - 왜, 언제 수정했지?
 - 1년 후에 보면 무슨 소스인지 알까?
 - 새 기능 개발하기 전 버전을 다시 가져오려면...
- 동료와 협업할 때 소스 관리는 어떻게 할까?
 - 압축해서 Gmail로?
 - Drive를 이용해서 share?
 - 서버에 올려놓고 같이 접속해서 고치면?
 - 최종본은 누가?



```
hello
├── hello1
│   ├── hello1 - 복사본
│   ├── hello1 - 복사본 (2)
│   └── hello1 - 복사본 (3)
```

VCS (Version Control System)

- SCM (Source Control System, Source Configuration Mangement)
- 프로젝트 개발(협업)할 때 소스코드, 문서 등 버전을 관리하는 시스템



19

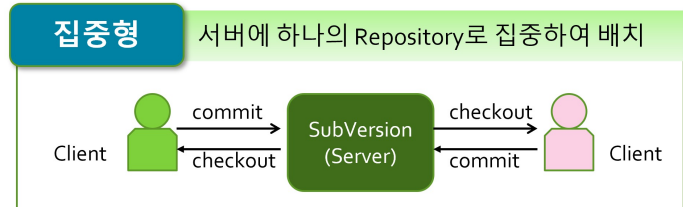
VCS 종류

- CVCS (Centralized VCS)
 - 중앙집중식 / Client-Server 방식
 - 서버에 소스파일과 버전 히스토리 저장
 - 단점 : 서버가 고장 나면 협업이 어려움
 - CVS (Concurrent Version System), SVN (SubVersion) 등
- DVCS (Distributed VCS)
 - 분산 관리 시스템
 - 소스와 히스토리를 여러 PC에 분산 저장
 - 서버 장애시 로컬 저장소를 이용하여 중앙 저장소 복원 가능
 - Git, Mercurial, Bazaar 등

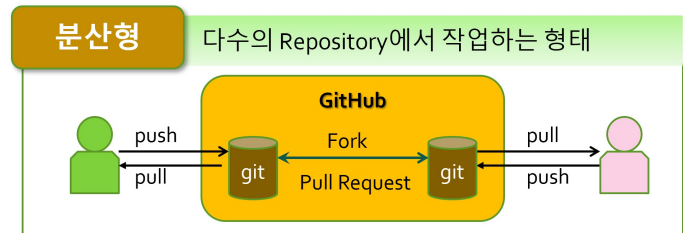
20

VCS 종류

- CVCS (Centralized VCS)



- DVCS (Distributed VCS)



Git

- 2005년, 리눅스 커널 개발자, 리누스 토르발스에 의해 개발
- DVCS (분산 버전 관리 시스템) : 빠른 수행속도
- 빠른 협업 환경 조성
- 지속적인 버전 관리가 필요한 경우 사용(SW, Design, ...)
- 수천 개의 브랜치(branch) 작업 동시 수행
- 누가, 무엇을, 언제, 왜, 어떻게 수정했는지 코드 리뷰가 가능
- 대형 프로젝트의 버전 관리가 가능함.
- 중앙 서버와 독립적으로 Local에서 완벽하게 소스와 이력 관리
- Local에서 소스 버전 관리할 때 사용

Git vs Github

Git

- 분산 버전 관리 시스템 소프트웨어
- 로컬저장소(Local Repository) 사용
- Local 내에서 git을 이용하여 버전 관리

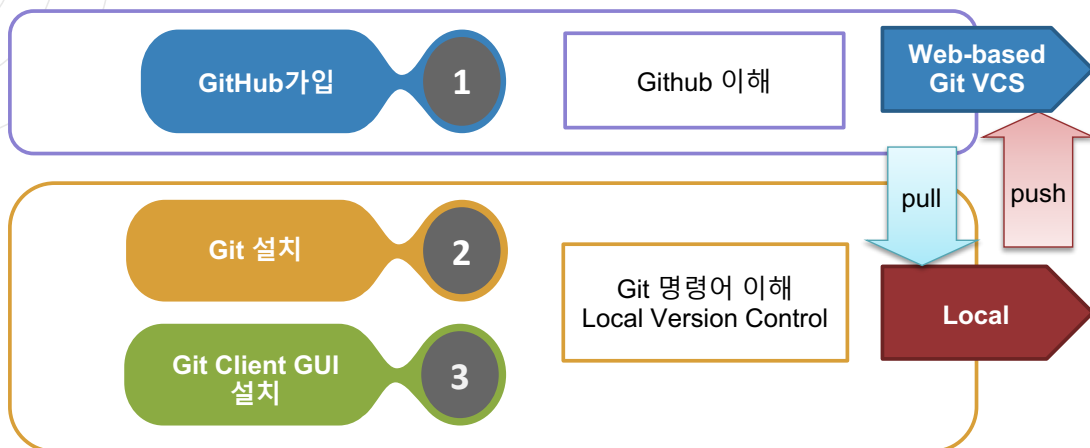


GitHub

- 원격저장소(Remote Repository) 제공.
- Local에서 관리하는 소스코드를 업로드후 공유
- 다른 개발자와 협업할 때 사용
- <https://github.com/>

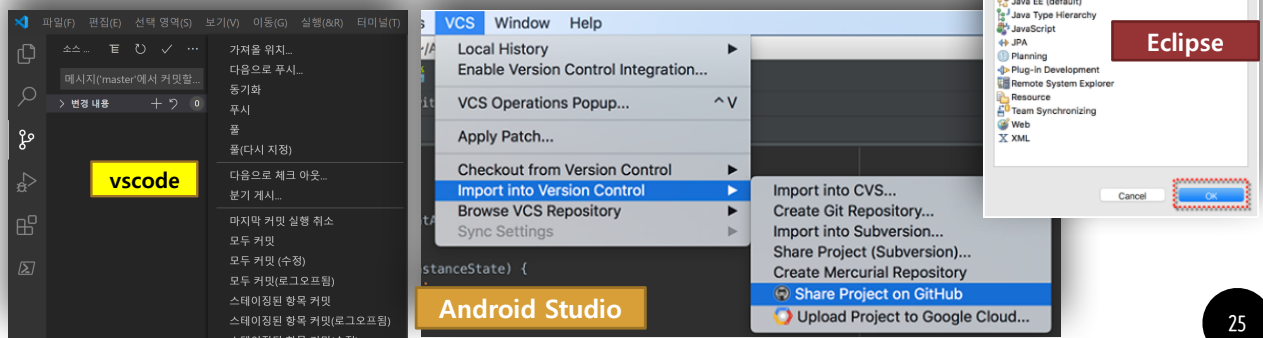


Git & Github



Git 연동

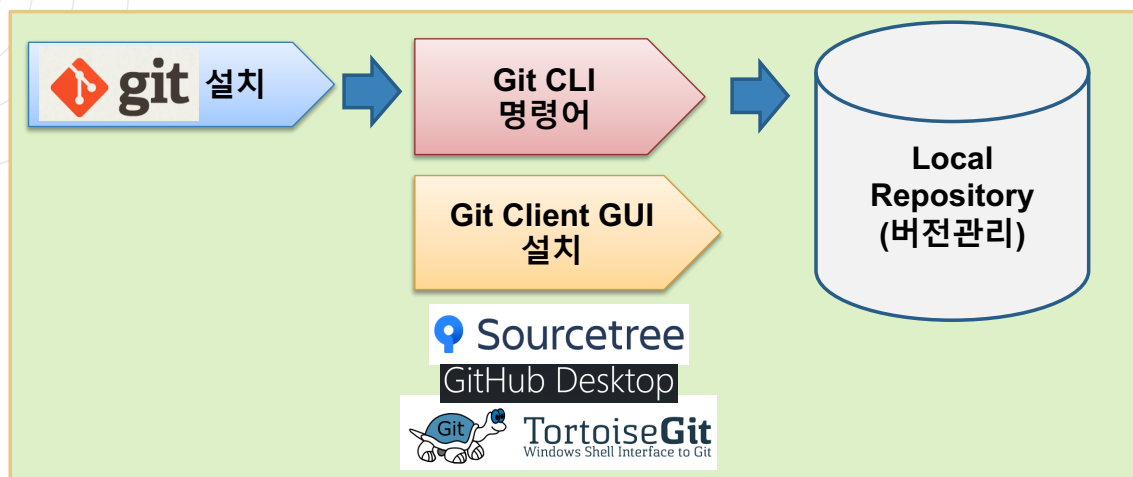
- Windows, Mac OS, 리눅스, 유닉스 등 지원
- 여러 프로그램에서 git 연동 및 github 소스 공유기능 제공
 - Vscode, VS, android studio, eclipse, IntelliJ IDEA 등



25

Git 사용

<https://git-scm.com/>



26

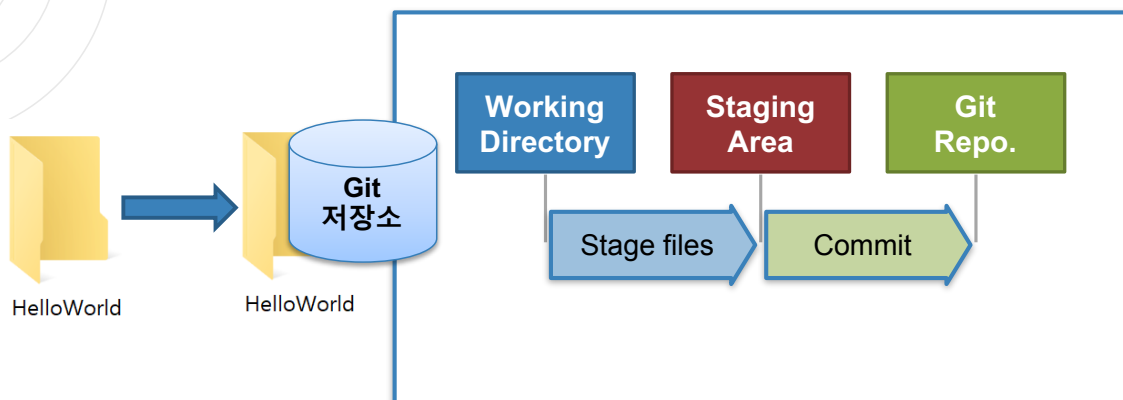
Git : 사용자, 이메일 설정

- Local PC 버전관리
 - Windows : Git Bash 프로그램으로 설정
 - Mac : terminal 사용하여 설정
- Server 에서 버전관리
 - 서버에 접속하여 설정

```
walab-HGU:~:> git config --global user.name "user1"
walab-HGU:~:> git config --global user.email "user1@gmail.com"
walab-HGU:~:> git config --global color.ui auto
walab-HGU:~:> git config --global --list
user.name=user1
user.email=user1@gmail.com
color.ui=auto
```

27

Git 버전 관리 과정



28

Repository 생성

- 로컬 저장소 생성

```
walab-HGU:~/lab6:> mkdir hello
walab-HGU:~/lab6:> cd hello
walab-HGU:~/lab6/hello:> git init
Initialized empty Git repository in /home/User1/20210SS/lab6/hello/.git/
```

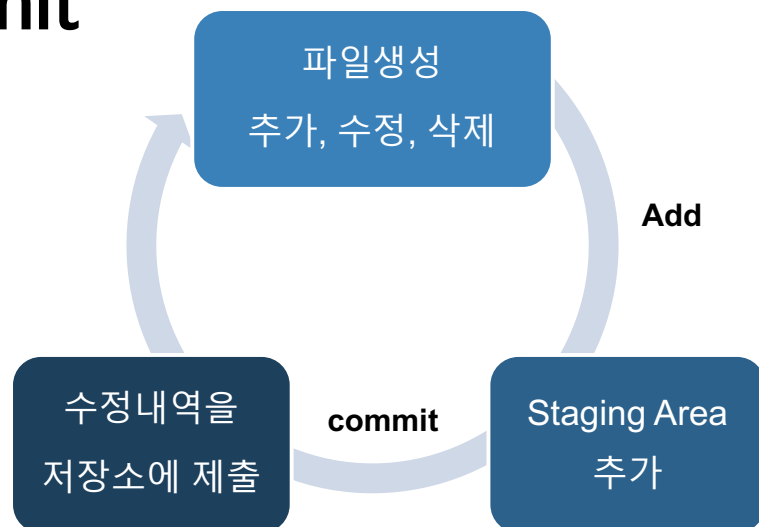
- 원격 저장소 복제

- 원격 저장소 Repo URL 복사

```
walab-HGU:~/lab6:> git clone <GitHub의 Repository주소>
Cloning into 'simple-chat-client-server'...
remote: Enumerating objects: 7, done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 7
Unpacking objects: 100% (7/7), done.
walab-HGU:~/lab6:> ls
hello simple-chat-client-server
```

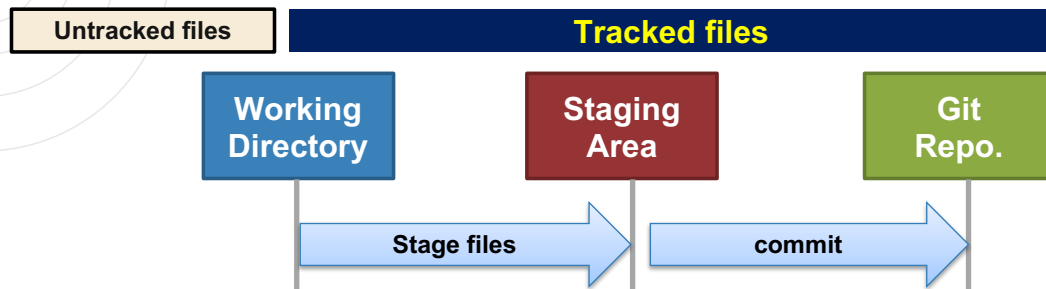
29

add/commit



30

add/commit



```
git add index.html index2.html
git add *.*
git add *.html
```

```
git commit
git commit -m "commit msg"
git commit -am "commit msg"
```

git rm

- Untracked file 삭제
 - rm 명령어를 사용하여 삭제
- Tracked file 삭제
 - 로컬 디렉터리와 git 저장소에서 모두 삭제
 - `git rm <filename>`
- git에서만 삭제, 로컬 디렉터리에는 삭제하지 않음
 - `git rm --cached <filename>`

git status

- 현재 파일 상태를 확인하기 위해 사용하는 명령
 - Untracked file 상태
 - Staged 전 상태
 - commit 전 상태

```
walab-HGU:~/lab6/hello:> touch hello.c
walab-HGU:~/lab6/hello:> git status
walab-HGU:~/lab6/hello:> git add *.c
walab-HGU:~/lab6/hello:> git status
walab-HGU:~/lab6/hello:> git commit -m "create hello.c"
[master (root-commit) 0d01bae] create hello.c
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hello.c
walab-HGU:~/lab6/hello:>
```

33

git log

- Commit history를 볼 수 있음

```
git log
git log -3
git log -p -1
git log -2 --oneline
git log --author=Brandon
git log --pretty=oneline --graph
```

34

git diff

- 커밋 내역을 비교하거나 commit과 working tree 변경 내역 확인
- 워킹 디렉터리와 Staging area 비교

```
walab-HGU:~/lab6/hello:> git diff
```

- Commit내용과 Staging area 비교

```
walab-HGU:~/lab6/hello:> git diff --cached  
walab-HGU:~/lab6/hello:> git diff --staged
```

- Commit과 다른 Commit 비교

```
walab-HGU:~/lab6/hello:> git diff commit1_checksum commit2_checksum
```

35

gitignore 파일

- Git 저장소에서 관리할 필요가 없는 파일이나 폴더 작성
 - 실행파일, 목적파일
 - Hidden file
 - Id/password 등 보안 정보
 - 개인 파일
 - 실행파일 저장된 폴더(/bin, /out)
- Git 저장소 폴더에 .gitignore 파일 생성
- .gitignore 파일 생성 사이트 : 개발환경, 언어별 작성가능

<https://www.gitignore.io/>

36

8

Git (2) VCS (Version Control System)

이력 수정

- 최종 완료한 commit 내역 수정
 - Commit 후 Staging Area에 추가한 파일을 다시 포함하여 commit
 - Commit 메시지 수정

```
walab-HGU:~/hello:> git commit -m 'commit message'
walab-HGU:~/hello:> git add new_file
walab-HGU:~/hello:> git commit --amend
```

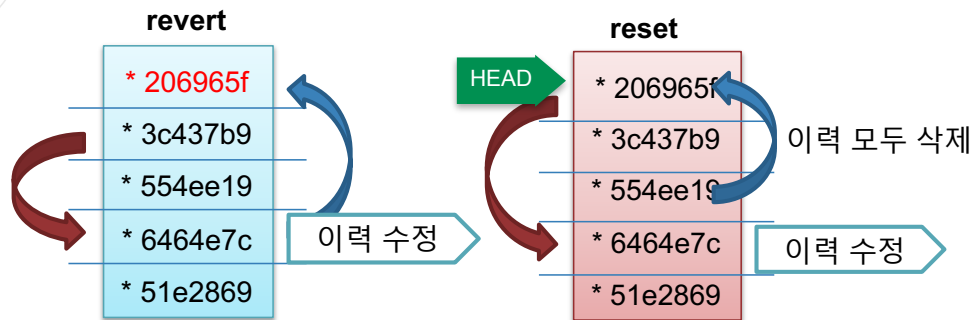
- WD에서 수정한 파일(modified)을 수정되기 전 상태(unmodified)로 변경

```
walab-HGU:~/hello:> git checkout <file1>
```

- git reset, git revert

revert vs reset

- 특정 커밋으로 되돌아 갈 수 있음. 취소 효과



39

reset

- Commit 취소
- 특정 커밋으로 돌아감으로 취소 작업
- 최근 커밋부터 특정 커밋 이후의 버전들을 히스토리에서 삭제

```
$ git reset [option] commit_checksum
```

- git add 취소 (Staging Area => WD 로 이동)

```
$ git reset HEAD <file명>
```

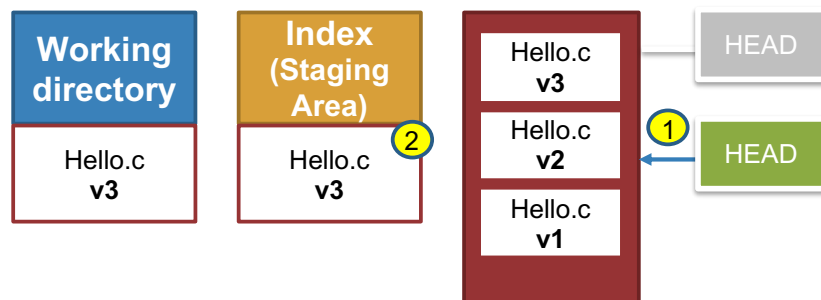
- Option
 - --soft
 - --mixed
 - --hard

40

git reset --soft

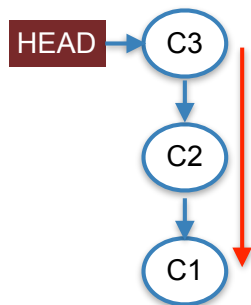
- HEAD를 특정 커밋으로 이동
- WD파일 보존, 해당 파일은 staged로 이동
- Commit 하면 원래 상태로 복원 가능

```
$ git reset --soft HEAD~
```



41

git reset --soft



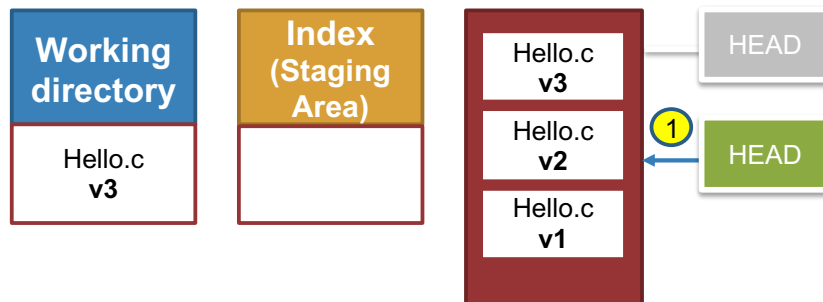
```
walab-HGU:~/lab7> touch test.txt
walab-HGU:~/lab7> git add test.txt
walab-HGU:~/lab7> git commit -m "C1"
walab-HGU:~/lab7> vim test.txt
walab-HGU:~/lab7> git commit -am "C2"
walab-HGU:~/lab7> vim test.txt
walab-HGU:~/lab7> git commit -am "C3"
walab-HGU:~/lab7> git log --oneline
dcf2782 (HEAD -> master) C3
edc8683 C2
a977e1c C1
walab-HGU:~/lab7> git reset --soft a977e1c
walab-HGU:~/lab7> git log --oneline
a977e1c (HEAD -> master) C1
walab-HGU:~/lab7> git status
```

42

git reset --mixed

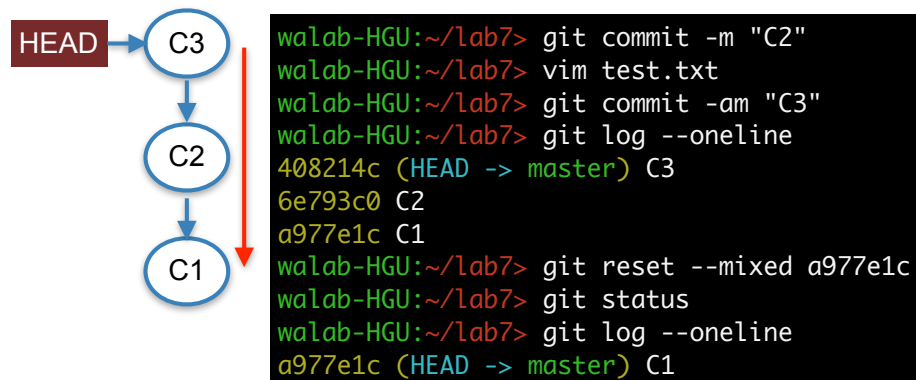
- Default 옵션
- WD파일 보존, 해당 파일 unstaged, HEAD 이동

```
$ git reset --mixed HEAD~  
$ git reset HEAD~
```



43

git reset --mixed



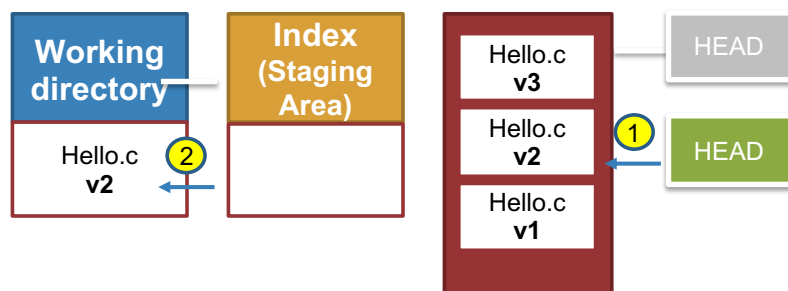
```
walab-HGU:~/lab7> git commit -m "C2"  
walab-HGU:~/lab7> vim test.txt  
walab-HGU:~/lab7> git commit -am "C3"  
walab-HGU:~/lab7> git log --oneline  
408214c (HEAD -> master) C3  
6e793c0 C2  
a977e1c C1  
walab-HGU:~/lab7> git reset --mixed a977e1c  
walab-HGU:~/lab7> git status  
walab-HGU:~/lab7> git log --oneline  
a977e1c (HEAD -> master) C1
```

44

git reset --hard

- WD파일 변경사항 삭제, 해당 파일 unstaged, HEAD 이동

```
$ git reset --hard HEAD~
```

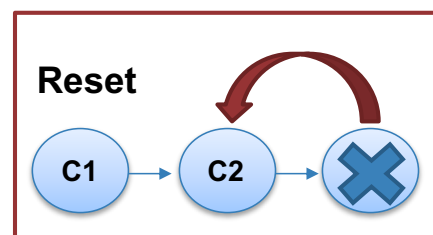
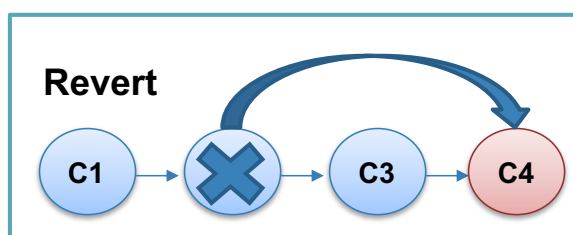


45

git revert

- commit된 스냅샷을 취소하는 명령
- 커밋 이력에서 취소하기 원하는 커밋에 의해 변경된 내용을 취소하기 위한 방법을 찾고 그 결과를 새로운 커밋으로 추가함
- 현재 커밋 이력을 삭제하지 않음

```
$ git revert HEAD~
```



46

git rm

- Untracked files

```
$ rm sample.txt
```

- Tracked files

- git 저장소 + 로컬디렉터리(WD) 모두 삭제

```
$ git rm sample.txt
```

- git 저장소 삭제, 로컬디렉터리(WD) 삭제하지 않음

```
$ git rm --cached sample.txt
```

47

git clone

- 원격저장소에서 로컬 저장소로 복제

Local Repo



Remote Repo

```
$ git clone <remote repository URL>
$ git clone <remote repository URL> <new folder>
$ git clone -b <branchname> <remote repository URL>
```

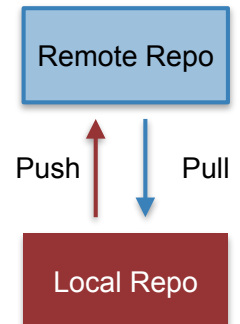
```
walab-HGU:~:> git clone -b gh-pages https://github.com/ahfarmer/calculator.git
walab-HGU:~/calculator:> git branch -a
walab-HGU:~:> git clone https://github.com/ahfarmer/calculator.git
walab-HGU:~/calculator:> git branch -a
```

48

Remote repo

- 로컬저장소에 원격 저장소를 연결하여 소스를 push 하거나 pull 할 수 있음
- 로컬저장소에 원격저장소 연결/삭제/ 정보보기

```
$ git remote
$ git remote -v
$ git remote -h
$ git remote add origin <remote repository URL>
$ git remote remove origin
$ git remote add calculator <remote repository URL>
```



49

Remote repo

- Push



```
$ git push origin master
```

- Pull



```
$ git pull origin master
```

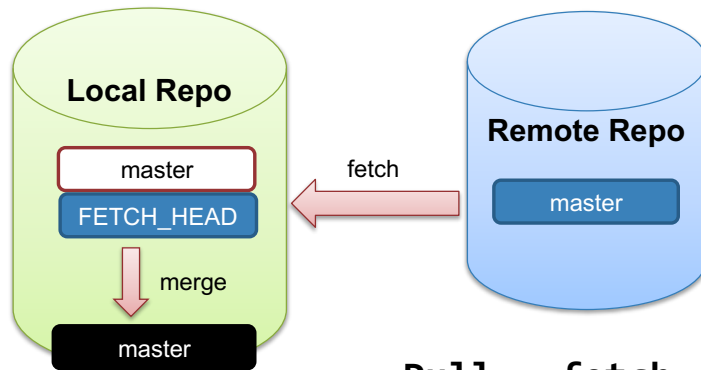
- fetch



```
$ git fetch
```

50

Pull vs fetch



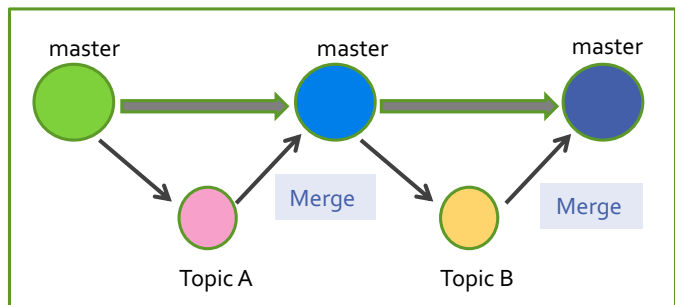
Pull = fetch + merge

51

Branch

- 기본 브랜치 : master branch
- 새로운 작업이 발생할 때 브랜치를 생성하여 작업
- 브랜치에서 작업이 완료되면 변경내용 및 이력을 master branch로 병합(merge)

```
$ git branch <new branch>  
$ git branch -h  
$ git branch -a  
$ git branch  
$ git branch -d <branchname>  
$ git branch -D <branchname>
```



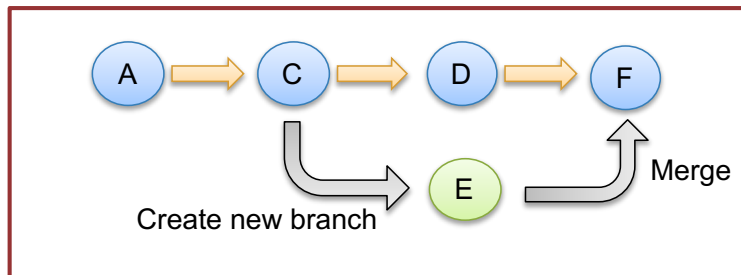
52

Checkout

- 다른 브랜치 전환할 때 사용

```
$ git checkout <branchname>
$ git checkout -b <new branch>
```

```
walab-HGU:~:> git branch
master
test
* test2
```

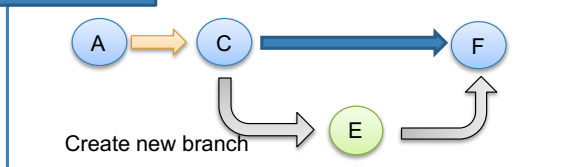


53

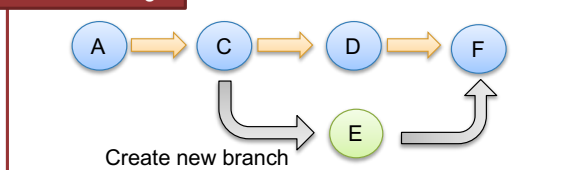
git merge

- 브랜치와 브랜치(master)에 병합하는 작업

Fast-forward merge



No fast-forward merge



```
$ mkdir git1
$ cd git1
$ git init
$ touch hello.c
$ git add hello.c
$ git commit -m "A"
$ vim hello.c
$ git commit -am "C"
$ git checkout -b new
$ vim hello.c
$ git commit -am "E"
$ git checkout master
$ git merge new
```

54