OSS : 오픈소프트웨어
소스코드를 공개해 누구나 특별한 제한 없이 그 코드를 보고 사용할 수 있는 오픈 소스 라이선스를 만족하는 소프트웨어, study, change, distribute

OSS개발도구 : 버전관리시스템(ex)git), 이슈 트래커, 코드리뷰시스템, 지속적 통합 시스템
OSS license : GPL , LGPL, MIT(제약이 없음). 등등
OSS 사용시 라이선스에 따른 제약 반드시 확인할 것

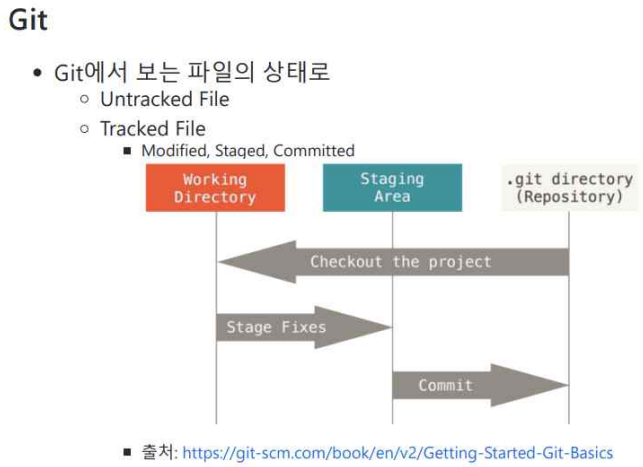
OSS찾기: Black duck Open Hub, Free Open Source Software
오래된 커밋이력을 갖는 OSS는 완벽하다기 보다는 사람들이 이제 관심이 없다는 것

<Git>
작업공간(working directory)
:소스코드가 저장되어 있는 폴더

스테이지(staging area)
:저장소(repository)로 변경 내역을 저장하기 위한 파일들의 목록

저장소(repository)
:로컬 저장소: 보통 작업공간에는 .git 라는 이름의 디렉터리로 존재(숨겨진 파일), 변경 내역이 저장되는 곳

:원격 저장소: 여러 사용자가 공유하는 저장소 : Github



Untracked File: git에서 추적(어디가 어떻게 변했는지)하지 않는 파일로, 한번도 add하지 않는 파일
Tracked File: git에서 추적하는 파일로, 파일의 상태는 3가지. Modified, Staged, Committed
Untracked File로도 체크하는게 싫다. 완전히 무시하고 싶다. : .gitignore파일에 무시하고 싶은 폴더 or 파일명 or 확장자를 입력한다. (뒤에서 확인)

<Git 설정>
명령어 띄어쓰기 반드시 조심해서!

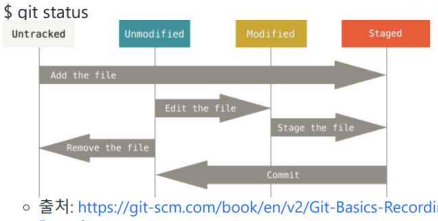
-사용자 설정-
git config --global user.name "SungwooLee"
git config --global user.email "swlee@tukorea.ac.kr"
(--global) : 빼면 저장소마다 별도로 지정 가능함.
git config --global core.editor nano

git help
git help config
(git help 명령어, 헷갈릴 때)

<Git 저장소 만들기>
git init :
-로컬 디렉토리에 저장소 새로 만들기
-하위폴더에 영향을 주는 구조이기 때문에 저장소 폴더를 새로 만들고 현재 디렉토리를 새로 만든 폴더로 옮긴 다음에 git init을 해야 새로 만든 폴더에 .git라는 로컬저장소가 생성됨
-숨겨진 파일이므로 ls가 아니라 ls -a를 해야 .git 폴더가 보임.
-로컬저장소를 만들지 않는다는 것은 commit할 공간이 없다는 것이므로, 반드시 처음에 만들어야함.
-하위폴더에 추가로 저장소를 생성하지 않는다. 복잡해진다. 나중에 git clone할 때 주의하자.

<변경 이력 저장하기>
변경 이력 저장하기

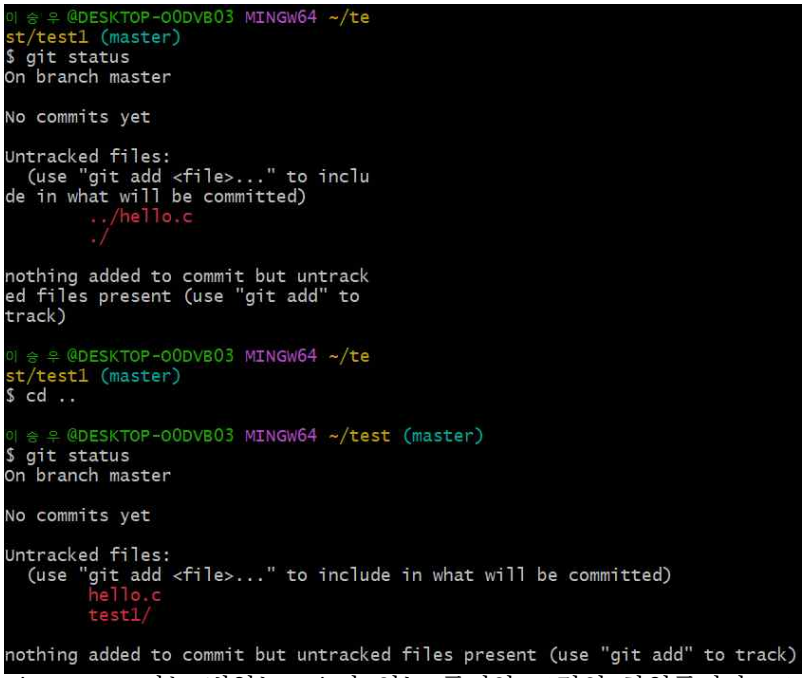
- 작업 디렉터리 내의 파일 Untracked/Tracked
- Tracked 파일: Unmodified(Committed), Modified, Staged
- $ git status



- 출처: https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

Untracked File: git에서 추적(어디가 어떻게 변했는지)하지 않는 파일로, 한번도 add하지 않는 파일
Tracked File: git에서 추적하는 파일로, 파일의 상태는 3가지. Modified, Staged, Committed
Untracked File로도 체크하는게 싫다. 완전히 무시하고 싶다. : .gitignore파일에 무시하고 싶은 폴더 or 파일명 or 확장자를 입력한다. (뒤에서 확인)

git status
현재 디렉토리의 파일 상황(untracked/tracked : modified, staged, committed) 알아보기



git status 가능 범위는 .git가 있는 폴더와 그것의 하위폴더다. 그리고 git status를 어느 폴더에서 하냐에 입력창의 결과가 달라진다. 현재 디렉토리의 변경사항을 보여주는 것임을 명심. 상위폴더와 하위폴더 안의 세부파일은 보여주지 않는다. 상위폴더에서 add, commit하면 하위폴더 test1안에 변경이력이 있는 세부파일도 모두 add, commit 된다.

변경 이력 저장하기(새 파일)

- 새 파일 생성, tracked-staged로 변경

```
$ echo "newfile" > newfile
$ git status
On branch master
No commits yet
Untracked files:
(use "git add <file>..." to include in what will be committed)
        newfile
nothing added to commit but untracked files present (use "git add" to track)
```

```
$ git add newfile
$ git status
On branch master
No commits yet
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
        new file:   newfile
```

- tracked-staged 파일을 committed로 변경

```
$ git commit -m "initial commit"
[master (root-commit) 041682a] initial commit
        1 file changed, 1 insertion(+)
create mode 100644 newfile
$ git status
On branch master
nothing to commit, working tree clean
```

git add 파일명.확장자
-위에 파일은 확장자가 없었지만, 파일명과 확장자를 모두 입력해야한다.
-서로 연관성이 있는 것들을 add로 각각 한 stage에 올린다음 commit을 한다.
-add, commit은 미루지 말고 진행하자.

git commit -> commit 내용입력 -> cntrl^s cntrl^x
-add를 통해 한 스테이지에 올린 변경내용들을 commit을 통해 로컬 저장소로 보낸다.
-comment 내용이 좀 길다 싶으면 git commit을 이용
git commit -m "comment 내용 입력" : 짧은 comment인 경우

변경 이력 저장하기(2개이상 파일)

- stage에 2개 이상의 파일 올리고 커밋하기

```
$ echo "another file" > newfile2
$ vi newfile
$ git status
On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
        modified:   newfile
Untracked files:
(use "git add <file>..." to include in what will be committed)
        newfile2
no changes added to commit (use "git add" and/or "git commit -a")
$ git add newfile
$ git add newfile2
$ git commit -m "Add newfile2"
[master 503f0b7] Add newfile2
2 files changed, 2 insertions(+)
create mode 100644 newfile2
```

git add를 각 파일별로 진행해서 한 개의 stage에 올리고 commit한다.
stage1-commit1, stage2-commit2 이런식으로 진행 안된다.
modified1,modified2,modified3.. - stage - commit 이런식으로 진행된다.

변경 이력 저장하기(파일 일부만 스태이징)

- stage에 파일 변경 부분 확인하면서 일부분만 올리기
  - $ git add -p

```
$ git add -p
diff --git a/git-basic.md b/git-basic.md
index 5f742ec..839532a 100644
--- a/git-basic.md
+++ b/git-basic.md
@@ -55,8 +55,8 @@ backgroundColor: #fff
-- GUI: git-gui, SourceTree
-![](images/git-gui.png) ![](images/sourcetree.png)
+- GUI: git-gui, SourceTree, Git Extensions
+![](images/git-gui.png) ![](images/sourcetree.png) ![](images/gitextensions.png)

(1/1) Stage this hunk [y,n,q,a,d,e,?]? ?
```

  - y: 이 hunk를 추가, n: 추가하지 않음, q: 종료, ?: 도움말

git add -p

tracked 파일을 대상으로 사용 가능함. untracked 파일은 add commit부터하고 사용가능하다. 한 파일의 내용 수정을 산발적으로 한 경우, 수정 코드 중에 의미 있는 코드 덩어리를 묶어서 add,commit 하고 싶은 경우 사용한다. 어느정도를 덩어리(hunk)로 볼지는 git에서 정한다. y(yes) n(no: no) q(나가기), 이후 과정인 commit은 똑같이 진행된다. 되도록 add와 commit은 그 때 그 때 의미 있는 것끼리 묶어서 진행한다.

git rm [파일이름]

-파일을 삭제하고, 삭제한 파일을 staged 상태로 (add까지함)
-파일 상태는 committed만 가능함. modified or staged는 불가하다
-이어서 git commit을 통해 삭제한 파일을 committed

git mv [파일이름] [새 파일 이름]

-파일 삭제와 마찬가지로 staged로 만들고, 파일상태는 committed만 가능함
-이어서 git commit

git diff

-tracked/modified 파일의 변경된 부분을 보여줌
-한번 add,commit한 파일을 수정한 경우, 그 수정된 부분을 보여줌

git diff -staged

-tracked/staged 파일의 변경된 부분을 보여줌
-add로 stage에 올라와 있는 파일의 변경된 부분을 보여줌

.gitignore

-git이 무시할 파일 or 폴더 or 확장자 지정
-보통 git init을 하는 디렉토리에 하는 것이 일반적
-nano .gitignore :
파일명.확장자 : 파일명이 없으면 숨겨진 파일을 의미함. .gitignore은 숨겨진파일이고 확장자는 gitignore임.
.gitignore파일을 생성한 다음 필요에 따라 파일에 아래 내용을 선택하여 입력한다.

.a : 하위폴더에 확장자 a를 갖는 파일은 모두 무시한다.
!lib.a : 확장자가 a 인 파일중 파일명이 lib인 파일은 무시하지 않는다.
/TODO : .gitignore파일이 있는 디렉토리에 TODO라는 파일명을 가진 파일만 무시한다.
하위디렉토리의 TODO는 무시하지 않는다.
build/ : .gitignore파일이 있는 디렉토리의 바로 하위폴더 build에 있는 모든 것을 무시한다.

```
# no .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# only ignore the TODO file in the current directory, not subdir/TODO
/TODO
# ignore all files in the build/ directory
build/
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .pdf files in the doc/ directory
doc/**/*.pdf
```

git reset HEAD^
-HEAD의 마지막 commit을 단순히 제거하는 방법으로 취소
-혼자 작업할 때는 상관없음. 공동작업에서는 비추
-기존에 commit 되었던 파일을 수정하고 add,commit을 한 상황을 가정
git reset HEAD^하면 add와 commit만 단순히 제거함. nano 파일을 하면 파일의 수정한 내용은 그대로 있음을 확인할 수 있음. 파일 상태는 modified 된 상태임. 단순히 add와 commit을 취소한다고 생각하자.

git revert HEAD ->comment입력
-HEAD의 마지막 commit을 취소하는 commit을 추가로 만드는 것
-기존에 commit되었던 파일을 수정하고 add,commit을 한 상황을 가정
최근에 수정한 내용을 지워야 함, 하지만 수정한 내용을 언제든지 복원할 수 있어야함.
git revert HEAD을 입력하면 comment입력창이 뜸(revert "최근 comment 내용") cntrl^s로 나오면 취소한 commit을 추가로 만든다(=파일을 수정하기 전으로 돌려놓는 commit)
이후 HEAD는 취소한 commit(=파일을 수정하기 전)을 가르키고 있다. 가지의 flow를 보면
기존 commit - 수정한 내용 commit - revert한 commit이다.
따라서 HEAD를 수정한 내용으로 옮겨 놓으면 복구가 가능하다.

git commit --amend
-같이 commit 해야 할 파일을 실수로 빼고 commit 했을 때
-
git add newfile
git commit -m "add newfile"
git add forgottenfile
git commit --amend
->같은 stage로 묶여서 commit 된다.

git restore --staged [파일이름]
-staged를 되돌리기(=modified 상태로 되돌린다)

git restore [파일이름]
-modified를 마지막 commit 버전으로 되돌리기
-수정한 내용은 모두 사라지니 주의하자.

git tag -a [태그이름]
-annotated tag: 태그이름, 이메일, 날짜, 커맨트 등을 함께 저장
-ex)git tag -a tagname ->메모창 뜸, 추가정보 입력->cntrl^s로 나옴

git tag [태그이름]
-lightweight tag : 단순 태깅

git log
-git 히스토리 : commit 내용, 번호, 태그, comment, 시간
-q로 엔터로 나올 수 있다.

#좋은 comment 작성 요령
제목과 본문을 한 줄 띄워 분리하기
제목은 영문 기준 50자 이내로
제목 첫 글자를 대문자로
제목 끝에. 금지
제목은 명령조로
본문은 영문기준 72자마다 줄 바꾸기
본문은 어떻게보다 무엇을, 왜에 맞춰 작성하기

Exercise

- Exercise
  - 로컬 저장소를 생성한다.(git init)
  - .gitignore 파일을 만들고 .o와 a.out을 넣는다.
  - .gitignore 파일을 staged/committed 한다.
  - main.c 파일을 만들고 staged/committed 한다. (이후 main.c를 수정할 때마다 gcc로 컴파일 한다.)
  - main.c 파일에 main() 함수를 추가하고 staged/committed 한다.
  - 태깅을 해본다. 태그 이름은 v1.0 으로 하자.
  - main.c 파일에 multiplication_table() 함수를 추가하고 staged/committed 한다. (구구단 함수)
  - main.c 파일 main() 함수에서 multiplication_table() 함수를 호출하도록 수정하고 staged/committed 한다.
  - main.c 파일을 임의로 수정한다.
  - git diff 를 해본다.
  - main.c 파일을 마지막 committed 상태로 되돌린다.
  - 히스토리를 살펴본다.

-git 명령어 모음-

git config (--global) user.name “SungwooLee”
git config (--global) user.email “swlee@tukorea.ac.kr”
git config --global core.editor nano
git help
git help config
git init
git status
git add 파일명.확장자
git commit
git commit -m “comment 내용 입력”
git add -p
git rm [파일이름]
git mv [파일이름] [새 파일 이름]
git diff
git diff -staged
.gitignore
git reset HEAD^
git revert HEAD
git commit --amend
git restore --staged [파일이름]
git restore [파일이름]
git tag -a [태그이름]
git tag [태그이름]
git log

-bash명령어 모음-

echo “파일내용” > 파일명 : 확장자 없는 간단한 파일, 실험용

nano 파일명.확장자 -> 파일내용입력 ->cntrl^s cntrl^x

pwd : 현재 절대 경로

ls : 현재 위치해 있는 디렉토리안에 있는 파일,폴더 목록 나열

ls -a : 숨긴파일도 포함 출력

ls -l : 소유자, 파일크기 등 자세한 정보

ls -al : 숨김파일 포함, 자세한 정보

ls [폴더이름] : 현재 위치한 디렉토리에 folder라는 디렉토리안에 있는 파일,폴더

cd ~/경로

cd [폴더이름]

cd ..

cd ../..

cd - : 방금 있던 위치로 이동 (cd ../..로 점프했으면 , 점프하기 전으로!)

mkdir [폴더명]

rmdir [폴더명]