

CVS, SVN GIT이 있다.

변경사항을 관리한다는 것이 공통적이다.

CVS의 아성을 SVN이 무너뜨림 ➔ 다시 GIT이 무너뜨림
처음에 사용해 보면 어렵다. 기능이 많다.

일반인이 사용하는 제품이 바로 Dropbox, Google Drive이다.

우리의 현실은 Git보다 더욱 지옥같다.

익숙한 사람들은 간단한 코드라도 관리하는 것이 편하다.

리눅스는 1천5백만라인 정도 된다.

맥에서는 xcode를 미리 설치하면 된다.

Git-scm.com에서 다운로드를 하면 된다.

폴더를 만들고 초기화 한다. .git폴더가 만들어 진다.

git init

이제 파일을 만들어 봅니다.

Vim은 어디에서나 사용할 수 있다.

Vim f1.txt

I를 누르면 입력이 된다.

Esc를 누르고 :wq를 해서 빠져나온다.

Git status를 해서 가장 먼저 익힌다. 현재 엔트래킹되고 있다.

Git add f1.txt를 한다.

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        f1.txt

nothing added to commit but untracked files present (use "git add" to track)
[jongdeokkim@MacBook-Pro gitfth %]
[jongdeokkim@MacBook-Pro gitfth %]
[jongdeokkim@MacBook-Pro gitfth % git add f1.txt]
[jongdeokkim@MacBook-Pro gitfth % git status]
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   f1.txt

jongdeokkim@MacBook-Pro gitfth %
```

버전 관리를 본다. 완결된 상태가 버전이다.
처음에 한번만 시도한다. 이름과 이메일 주소

```
jongdeokkim@MacBook-Pro gitfth %  
jongdeokkim@MacBook-Pro gitfth % git config --g  
jongdeokkim@MacBook-Pro gitfth % git config --g  
jongdeokkim@MacBook-Pro gitfth % git commit  
[master (root-commit) 3d71c90] 이 파일은 연습용  
1 file changed, 2 insertions(+)  
create mode 100644 f1.txt  
jongdeokkim@MacBook-Pro gitfth %
```

git log를 하면 내용을 볼 수 있다.

한번 더 깃에 추가해야 한다.

```
git add f1.txt
```

```
git status
```

add를 한 파일만이 커밋이 된다. 스테이징이 된 파일만 커밋을 할 수 있다.

파일을 추려서 선택적으로 커밋을 할 수 있도록 했다.

Git add f1.txt를 하면 커밋을 할 수 있는 대기상태로 들어간 것이다.

Stage area라고 한다. 대기 상태의 공간

리파지토리 개념을 가지고 있다. 커밋된 결과가 저장된 공간

```
commit e1a7624b0a0c611cb88c7a9db41b0dea9cda9c4a
Author: papasmf <papasmf1@gmail.com>
Date: Thu May 21 12:24:09 2020 +0900
```

4444

```
diff --git a/f1.txt b/f1.txt
index 59ef608..38bcfbd 100644
--- a/f1.txt
+++ b/f1.txt
@@ -1,2 +1,2 @@
-score : 1
+score : 2
```

```
commit 0d205a48635c5af7c43202587d8b44419c50bd2b
Author: papasmf <papasmf1@gmail.com>
Date: Thu May 21 12:18:31 2020 +0900
```

3

```
diff --git a/f2.txt b/f2.txt
new file mode 100644
index 0000000..f9b0988
--- /dev/null
+++ b/f2.txt
@@ -0,0 +1,2 @@
+score : 2
+
```

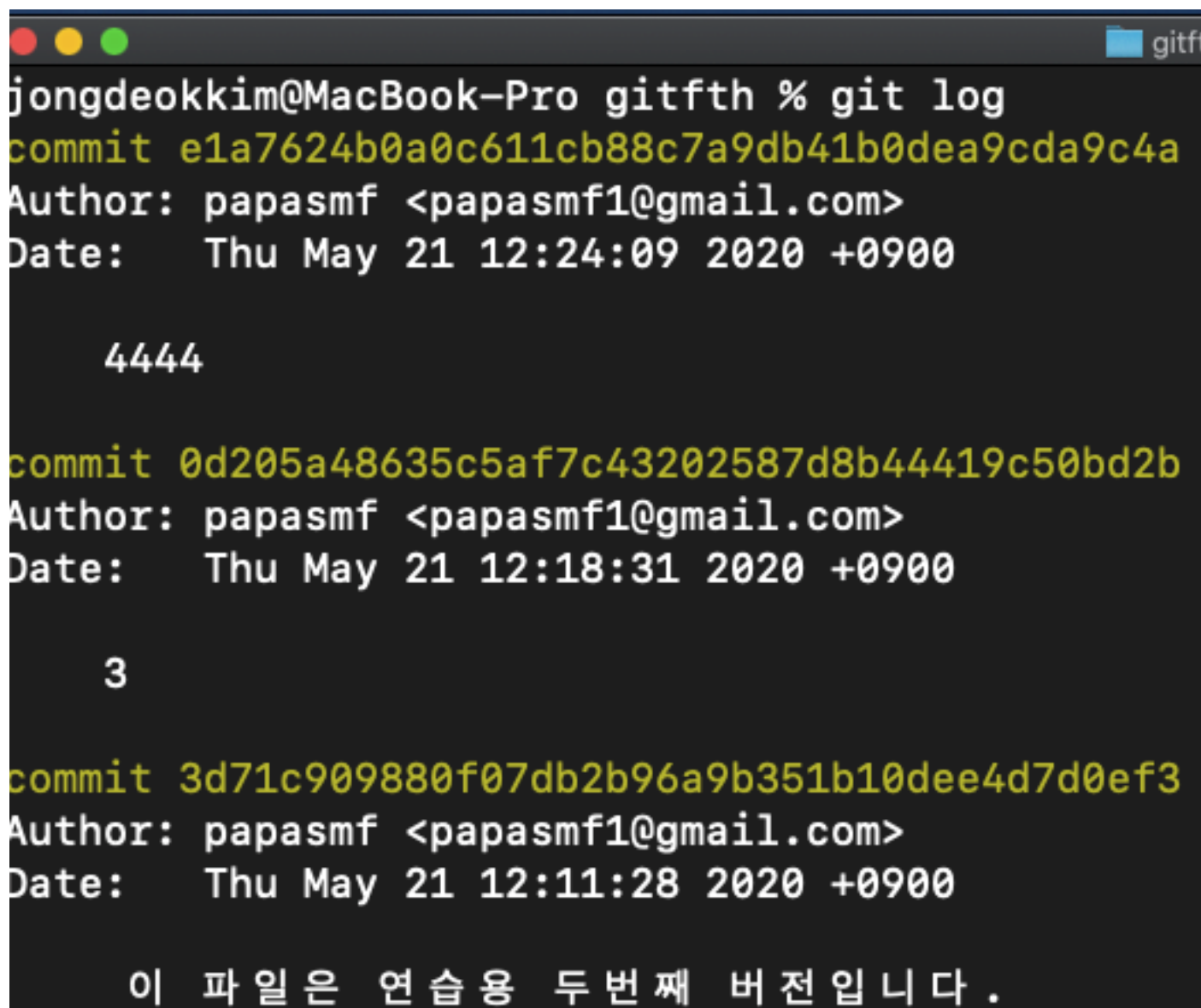
```
commit 3d71c909880f07db2b96a9b351b10dee4d7d0ef3
```

버전2에서 f2.txt파일이 생긴 것을 뜻함.

Git log

Git log -p

각자의 고유한 아이디가 있다.



```
jongdeokkim@MacBook-Pro gitfth % git log
commit e1a7624b0a0c611cb88c7a9db41b0dea9cda9c4a
Author: papasmf <papasmf1@gmail.com>
Date: Thu May 21 12:24:09 2020 +0900

    4444

commit 0d205a48635c5af7c43202587d8b44419c50bd2b
Author: papasmf <papasmf1@gmail.com>
Date: Thu May 21 12:18:31 2020 +0900

    3

commit 3d71c909880f07db2b96a9b351b10dee4d7d0ef3
Author: papasmf <papasmf1@gmail.com>
Date: Thu May 21 12:11:28 2020 +0900

    이 파일은 연습용 두번째 버전입니다 .
```

Git diff 하나의아이디..두번째아이디

```
[jongdeokkim@MacBook-Pro gitfth %  
[jongdeokkim@MacBook-Pro gitfth % git diff e1a76  
3202587d8b44419c50bd2b  
diff --git a/f1.txt b/f1.txt  
index 38bcfbd..59ef608 100644  
--- a/f1.txt  
+++ b/f1.txt  
@@ -1,2 +1,2 @@  
-f1.txt : 2  
+score : 1
```

이렇게 소스 코드 사이의 차이점을 확인할 수 있다.

내가 한 작업을 커밋하기 전에 마지막 리뷰를 할 수 있는 기회를 준다.

git diff

과거 버전으로 돌아가기

reset vs revert가 있다.

경계를 헛갈려한다. 깃 리셋을 하면 5, 4를 해당하고

원격 저장소를 사용하면 협업을 할 수 있다.

이런 경우는 리셋을 하면 안된다. 공유하기 전에 내 컴에 있는 것만 리셋 작업을 해야 한다.

```

[jongdeokkim@MacBook-Pro gitfth % git reset 0d205a48635c5a
HEAD is now at 0d205a4 3
[jongdeokkim@MacBook-Pro gitfth % git log
commit 0d205a48635c5af7c43202587d8b44419c50bd2b (HEAD ->
Author: papasmf <papasmf1@gmail.com>
Date: Thu May 21 12:18:31 2020 +0900

    3

commit 3d71c909880f07db2b96a9b351b10dee4d7d0ef3
Author: papasmf <papasmf1@gmail.com>
Date: Thu May 21 12:11:28 2020 +0900

    이 파일은 연습용 두 번째 버전입니다 .
[jongdeokkim@MacBook-Pro gitfth % ls -al

```

--hard는 좀 위험하다.

리버트는 현재 커밋을 날리고 다시 커밋을 하는 것이다.

중국어의 단어들중에 빈도수가 높은 단어를 검색한다.
Commit, add, log, diff, init이 가장 빈도수가 높다.

Git commit -help 를 보면 잔뜩 나온다.

파이썬을 설치하고
Sudo pip3 install gistory 를 설치한다.

브랜치 나무의 가지
고객에게 제공할 파일을 따로 만든다.

수업소개

이 수업은 Git 의 초심자에게는 기본적인 사용법을 중급자는 Git 이 동작하는 원리를 소개해드리기 위한 수업입니다.

이 수업에서는 명령어를 통해서 Git 을 다루는 방법을 소개합니다. 명령어 기반 Git 은 많은 시스템에 기본적으로 설치 되어 있기 때문에 명령어로 Git 을 다룰 수 있게 되면 많은 곳에서 특별한 설치 없이 Git 을 사용할 수 있습니다.

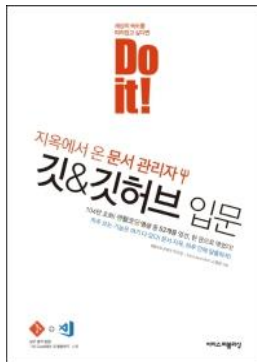
명령어로 Git 을 다루는 것이 어렵게 느껴지신다면 GUI 로 Git 을 다루는 수업을 추천드립니다.

<https://opentutorials.org/course/1492>

본 수업은 기술자료가 부족한 언어로 번역 할 계획이기 때문에 부득이하게 본문에 영어를 사용했습니다. 양해 부탁드립니다. 본 영상의 영어 작업은 개발자 영어의 나솔님께서 도움을 주셨습니다.

GIT 강의가 책으로도 나왔습니다.

저자가 만든 동영상 강의를 책으로도 나왔습니다. 책을 구입하시면 인세의 5%가 비영리 단체 오픈튜토리얼스에 후원금으로 사용됩니다. 책으로 공부하고 싶은 분들은 살펴보세요.



Do it! 지옥에서 온 문서관리자 깃&깃 허브 입문 ([교보](#), [알라딘](#), [yes24](#))

깃은 6개의 커밋으로 만들어져 있다.

이미 작업했던 것을 원격 저장소로 올려본다.

기본이 master, origin

Push : 깃은 로컬 저장소를 기준으로 말을 한다. 로컬에서 원격저장소로 보낼때 푸쉬한다라고 말한다.

git push -u origin master -u는 한번만 사용하면 된다.

주로 동기화되는 원격 저장소를 origin을 많이 사용한다. 기본 브랜치는 master를 사용한다.

Git_home

Git_Office를 만들어서 집에서 작업한 것을 회사에서 그대로 받아서 작업하는 것을 보여준다.

(계정이 2개인 것을 테스트하는 것과 비슷하다)

작업을 끝내면서 push를 한다.

작업을 하기 전에 pull로 땀겨온다.

Git commit -am '5'

Git push

다시 작업을 하는 경우

Git pull

수정하고

Git commit -am '6'

Pit push를 한다.

https가 아닌 Use SSH를 사용할 수 있다.

지옥에서 온 깃 개발자 강의 정리



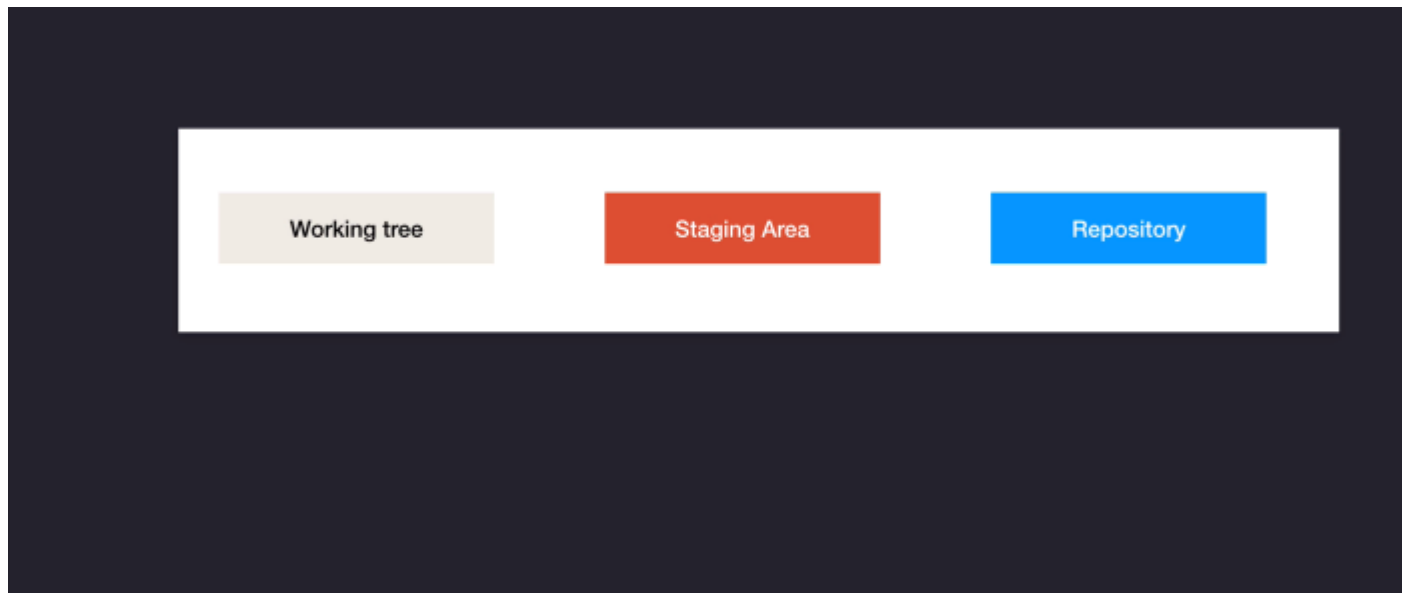
Git-scm.com에서 받은 툴(셸)이 기본이 된다.

<https://opentutorials.org/course/3839>

SourceTree
TortoiseGit
Github Desktop

다양한 도구들이 있다.

파일의 이름을 바꾸지 않고도 로그(히스토리)를 관리할 수 있다.



버전이 저장되어 있는 곳이 라파지토리이다.

.git이라는 폴더가 하위에 만들어 진다.

Working tree는 아직 버전으로 만들어지기 이전이다.

2개의 버전을 Staging Area에 올린다.

깃에게 버전을 만들라고 하면 2개의 파일을 묶어서 하나의 버전을 만들다.

`git init .` 을 실행한다.

hello1.txt를 만든다.

`git status` 를 실행해 본다.

아직은 언트래킹되고 있다.

`git add hello1.txt` 를 실행한다.

`git commit` 버전을 만들어서 제출한다는 의미이다.

메시지를 한번에 추가하는 경우라면

```
git commit -m "message 1"
```

`git log` 역사를 보고 싶다고 생각하면 된다.

파일이 다시 한줄을 더 추가한다.

`git add hello1.txt` 워킹트리에 있는 것을 스테이징으로 올린다.

`git commit -m "message 2"` 를 하면 스테이징에 올려둔 파일을 버전으로 만들어 준다.

hello1.txt 파일에 for를 추가하고

`git status`

`git diff` 를 하면 현재 상태가 아직 스테이징에 있지 않고 다른 부분을 보여준다.

```
C:\WDocumentWhello-git-cli>
C:\WDocumentWhello-git-cli>
C:\WDocumentWhello-git-cli>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:   hell1.txt

no changes added to commit (use "git add" and/or "git commit -a")

C:\WDocumentWhello-git-cli>git diff
diff --git a/hell1.txt b/hell1.txt
index bfbb64f..101f5d2 100644
--- a/hell1.txt
+++ b/hell1.txt
@@ -1,5 +1,3 @@
 test
 test2
-test3
-four
-sss
+four2
```

과거 버전을 돌아간다면

`git reset --hard`

`git log -p` 패치의 약자를 사용한다.

```
commit 3fdb0c8a2e33e74c7a8200bf07d0c3881a085513 (HEAD -> master)
Author: papasmf1 <papasmf1@gmail.com>
Date: Thu Feb 18 14:33:29 2021 +0900
```

message 55

```
diff --git a/hell1.txt b/hell1.txt
index 3d46185..bfbb64f 100644
--- a/hell1.txt
+++ b/hell1.txt
@@ -1,4 +1,5 @@
 test
 test2
 test3
-four
W No newline at end of file
+four
+sss
```

```
commit 4aa304e1ff47de6676a2b38901def f734d066ddf
Author: papasmf1 <papasmf1@gmail.com>
Date: Thu Feb 18 14:32:17 2021 +0900
```

message 4

```
diff --git a/hell1.txt b/hell1.txt
index b02def2..3d46185 100644
--- a/hell1.txt
+++ b/hell1.txt
```

헤더가 마스터를 가리키고 있다. 현재는 최신 버전이 마스터이다.

git checkout 커밋아이드를 복사하면 된다. 그러면 그 이전버전으로 돌아간다.

```
C:\WDocumentWhello-git-cli>git checkout 60900a5e8c247798dc778918e4516290
Warning: you are leaving 1 commit behind, not connected to
any of your branches:

    a8ae72b message 7

If you want to keep it by creating a new branch, this may be a good time
to do so with:

    git branch <new-branch-name> a8ae72b

HEAD is now at 60900a5 message 2

C:\WDocumentWhello-git-cli>type hell1.txt
test
```

`git checkout master` 를 실행하면 원상태로 돌아간다. 와우~~

`git add .` 을 하면 모든 파일을 추가한다.

또는 `git add src` 폴더를 지정해도 된다.

`git commit -am "message12"`

한번에 `add`를 하고 `commit`을 할 때 사용한다.

조심할 부분은 최초에 한번은 `add`가 되어야 가능하다.

엔트랙트상태에서는 `-am`을 사용할 수 없다. 조심~~

`git commit` 하고 엔터를 치면

기본 에디터에서 메시지를 입력할 수 있는 창이 뜬다.

나노 에디터인 경우 `ctrl-x,y`를 입력하면 저장하고 빠져나온다.

기본 에디터를 변경한다면 아래와 같이 한다.


```
create mode 100644 hello2.exe  
→ hello-git-cli git:(master)  
git log  
→ hello-git-cli git:(master)  
git config --global core.editor "nano"
```

삭제를 하는 경우는 아래와 같이 한다.

git reset -hard 아이디를 적어준다.

깃백업 수업

호스팅은 깃허브에 할 수 있고 깃랩도 좋다. 프라이빗 서비스도 깃랩은 무제한으로 무료

원격 저장소를 연결할 때 아래와 같이 한다. ssh가 아닌 http를 사용해서 한다. 조심!!

Quick setup — if you've done this kind of thing before



Set up in Desktop

or

HTTPS

SSH

<https://github.com/papasmf1/pythonBasic.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#).

...or create a new repository on the command line

```
C:\WDocument\hello-git-cli>git remote add origin  
https://github.com/papasmf1/pythonBasic.git
```

원격 저장소를 확인할 수 있다.

```
C:\Document\hello-git-cli>git remote  
origin
```

원격 저장소의 주소를 보고 싶은 경우라면

```
C:\Document\hello-git-cli>git remote -v  
origin https://github.com/papasmf1/pythonBasic.git (fetch)  
origin https://github.com/papasmf1/pythonBasic.git (push)
```

파일을 업로드하는 것은 다음과 같이 한다.

```
C:\Document\hello-git-cli>git push
```

fatal: The current branch master has no upstream branch.

To push the current branch and set the remote as upstream, use

```
git push --set-upstream origin master
```

(그대로 복사해서 사용하면 된다.)

```
C:\Document\hello-git-cli>git push --set-upstream origin master
```

info: please complete authentication in your browser...

Enumerating objects: 15, done.

Counting objects: 100% (15/15), done.

Delta compression using up to 2 threads

Compressing objects: 100% (5/5), done.

Writing objects: 100% (15/15), 1.07 KiB | 274.00 KiB/s, done.

Total 15 (delta 0), reused 0 (delta 0), pack-reused 0

To https://github.com/papasmf1/pythonBasic.git

* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

새로운 파일을 만들어서 추가하는 것은 다음과 같다.

hello2.txt를 생성한다.

```
git add hello2.txt
```

```
git commit -am "message 2"
```

```
git push
```

이렇게 하면 웹사이트로 업로드된 것을 볼 수 있다.

클론을 만드는 것은 다음과 같이 한다. 폴더를 지정하지 않으면 아래와 같이 pythonBasic폴더를 만들고 그대로 복제해 준다.

```
C:\Document\workGit>git clone
```

```
https://github.com/papasmf1/pythonBasic.git
```

```
Cloning into 'pythonBasic'...
```

```
remote: Enumerating objects: 18, done.
```

```
remote: Counting objects: 100% (18/18), done.
```

```
remote: Compressing objects: 100% (7/7), done.
```

```
remote: Total 18 (delta 0), reused 18 (delta 0), pack-reused 0
```

```
Receiving objects: 100% (18/18), done.
```

```
C:\Document\workGit>cd pythonBasic
```