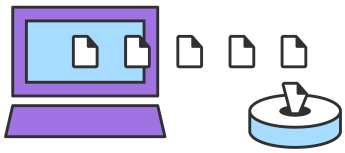
.gitignore



Git은 작업 카피의 모든 파일을 세 가지 중 하나로 본다.

1. 추적 - 이전에 수행되었거나 확약 된 파일.
2. untracked - 준비되거나 커밋되지 않은 파일. 또는
3. 무시 됨 - Git이 명시 적으로 무시하도록 지시 한 파일.

무시 된 파일은 일반적으로 저장소 소스에서 파생되거나 커밋되지 않아야하는 빌드 아티팩트 및 컴퓨터 생성 파일입니다. 몇 가지 일반적인 예는 다음과 같습니다.

- / node_modules 또는 / 패키지의 내용과 같은 종속성 캐시
- .o, .pyc 및 .class 파일과 같은 컴파일 된 코드
- / bin, / out 또는 / target과 같은 출력 디렉토리 빌드
- .log, .lock 또는 .tmp와 같이 런타임에 생성되는 파일
- .DS_Store 또는 Thumbs.db와 같은 숨겨진 시스템 파일
- .idea / workspace.xml과 같은 개인 IDE 구성 파일

무시 된 파일은 저장소의 루트에서 체크인 된 .gitignore라는 특수 파일에서 추적됩니다. 명시 적으로 git ignore 명령은 없습니다. 대신에 .gitignore 파일을 편집하고 커밋해야합니다. 무시할 새 파일이 있으면 수동으로 커밋해야합니다. .gitignore 파일은 저장소의 파일 이름과 일치하는 패턴을 포함하여 무시해야하는지 여부를 결정합니다.

Git ignore 패턴

.gitignore는 globbing 패턴을 사용하여 파일 이름과 일치시킵니다. 다양한 기호를 사용하여 패턴을 구성 할 수 있습니다.

Pattern	Example matches	Explanation*

`**/logs`	`logs`
	`/debug.log`
	`logs`
	`/monday`
	`/foo.bar`
	`build`
	`/logs`
	`/debug.log`
		You can prepend a pattern with a double asterisk to match directories anywhere in the repository.

`**/logs/debug.log`	`logs/debug.log`
	`build/logs/debug.log`
	but not
	`logs/build/debug.log`
		You can also use a double asterisk to match files based on their name and the name of their parent directory.

`*.log`	`debug.log`
	`foo.log`
	`.log`
	`logs/debug.log`
		An asterisk is a wildcard that matches zero or more characters.

`*.log`
`!important.log`
	`debug.log`
	`trace.log`
	but not
	`important.log`
	`logs/important.log`
		Prepending an exclamation mark to a pattern negates it. If a file matches a pattern, but *also* matches a negating pattern defined later in the file, it will not be ignored.

`*.log`
`!important/*.log`
`trace.*`
	`debug.log`
	`important/trace.log`
	but not
	`important/debug.log`
		Patterns defined after a negating pattern will re-ignore any previously negated files.

`/debug.log`	`debug.log`
	but not
	`logs/debug.log`
		Prepending a slash matches files only in the repository root.

`debug.log`	`debug.log` `logs/debug.log`	By default, patterns match files in any directory

`debug?.log`	`debug0.log`
	`debugg.log`
	but not
	`debug10.log`
		A question mark matches exactly one character.

`debug[0-9].log`	`debug0.log`
	`debug1.log`
	but not
	`debug10.log`
		Square brackets can also be used to match a single character from a specified range.

`debug[01].log`	`debug0.log`
	`debug1.log`
	but not
	`debug2.log`
	`debug01.log`
		Square brackets match a single character form the specified set.

`debug[!01].log`	`debug2.log`
	but not
	`debug0.log`
	`debug1.log`
	`debug01.log`
		An exclamation mark can be used to match any character except one from the specified set.

`debug[a-z].log`	`debuga.log`
	`debugb.log`
	but not
	`debug1.log`
		Ranges can be numeric or alphabetic.

`logs`	`logs`
	`logs/debug.log`
	`logs/latest/foo.bar`
	`build/logs`
	`build/logs/debug.log`
		If you don't append a slash, the pattern will match both files and the contents of directories with that name. In the example matches on the left, both directories and files named *logs* are ignored

logs/	`logs/debug.log`
	`logs/latest/foo.bar`
	`build/logs/foo.bar`
	`build/logs/latest/debug.log`
		Appending a slash indicates the pattern is a directory. The entire contents of any directory in the repository matching that name - including all of its files and subdirectories - will be ignored

`logs/!logs/important.log`	`logs/debug.log` `logs/important.log`	Wait a minute! Shouldn't `logs/important.log`be negated in the example on the left Nope! Due to a performance-related quirk in Git, you *can not* negate a file that is ignored due to a pattern matching a directory

`logs/**/debug.log`	`logs/debug.log`
	`logs/monday/debug.log`
	`logs/monday/pm/debug.log`
		A double asterisk matches zero or more directories.

`logs/*day/debug.log`	`logs/monday/debug.log`
	`logs/tuesday/debug.log`
	but not
	`logs/latest/debug.log`
		Wildcards can be used in directory names as well.

`logs/debug.log`	`logs/debug.log`
	but not
	`debug.log`
	`build/logs/debug.log`
		Patterns specifying a file in a particular directory are relative to the repository root. (You can prepend a slash if you like, but it doesn't do anything special.)

**이 설명은 .gitignore 파일이 협약과 마찬가지로 저장소의 최상위 디렉토리에 있다고 가정합니다. 저장소에 .gitignore 파일이 여러 개있는 경우 "저장소 루트"를 ".gitignore 파일이있는 디렉토리"로 정신적으로 대체하십시오 (팀의 온전한 고려를 위해 통합을 고려하십시오). *

이러한 문자 외에도 .gitignore 파일에 #을 사용하여 주석을 포함 할 수 있습니다.

```
# ignore all logs
*.log
```

파일이나 디렉토리가있는 경우 ₩를 사용하여 .gitignore 패턴 문자를 이스케이프 처리 할 수 있습니다.

```
# ignore the file literally named foo[01].txt
foo\[01\].txt
```

저장소에 .gitignore 공유하기

Git 무시 규칙은 일반적으로 저장소의 루트에있는 .gitignore 파일에 정의되어 있습니다. 그러나 저장소의 여러 디렉토리에 여러 .gitignore 파일을 정의하도록 선택할 수 있습니다. 특정 .gitignore 파일의 각 패턴은 해당 파일이 들어있는 디렉토리를 기준으로 테스트됩니다. 그러나 규칙과 가장 단순한 접근법은 루트에 단일 .gitignore 파일을 정의하는 것입니다. .gitignore 파일이 체크인되면 리포지토리의 다른 파일과 마찬가지로 버전이 지정되고 푸시 할 때 팀원과 공유됩니다. 일반적으로 .gitignore에는 저장소의 다른 사용자에게 도움이되는 패턴 만 포함시켜야합니다.

Personal Git ignore rules

특정 저장소에 대한 개인 무시 패턴을 .git / info / exclude의 특수 파일에 정의 할 수도 있습니다. 이러한 버전은 버전이 지정되지 않으며 저장소와 함께 배포되지 않으므로 이점 만 얻을 수있는 패턴을 포함하는 것이 좋습니다. 예를 들어 사용자 정의 로깅 설정이나 저장소의 작업 디렉토리에 파일을 생성하는 특수 개발 도구가있는 경우 .git / info / exclude에 추가하여 실수로 저장소에 커밋되지 않도록 할 수 있습니다.

Global Git ignore rules

또한 Git core.excludesFile 속성을 설정하여 로컬 시스템의 모든 리포지토리에 대한 전역 Git 무시 패턴을 정의할 수 있습니다. 이 파일을 직접 만들어야합니다. 전역 .gitignore 파일을 어디에 둘지 확실치 않으면 홈 디렉토리가 나쁜 선택이 아니며 나중에 쉽게 찾을 수 있습니다. 파일을 만들었 으면 git config로 위치를 구성해야합니다.

```
$ touch ~/.gitignore
$ git config --global core.excludesFile ~/.gitignore
```

서로 다른 파일 유형이 서로 다른 프로젝트와 관련되므로 어떤 패턴을 전역 적으로 무시할지 선택해야합니다. 일부 개발자 도구로 만든 특별 운영 체제 파일 (예 : .DS_Store 및 thumbs.db) 또는 임시 파일은 전 세계적으로 무시할 수있는 대표적인 후보입니다.

Ignoring a previously committed file

과거에 커밋 한 파일을 무시하려면 저장소에서 파일을 삭제 한 다음 .gitignore 규칙을 추가해야합니다. git rm과 함께 --cached 옵션을 사용하면 파일이 저장소에서 삭제되지만 작업 디렉토리에는 무시 된 파일로 남아있게됩니다.

```
$ echo debug.log >> .gitignore

$ git rm --cached debug.log
rm 'debug.log'

$ git commit -m "Start ignoring debug.log"
```

저장소 및 로컬 파일 시스템에서 파일을 삭제하려면 --cached 옵션을 생략 할 수 있습니다.

Committing an ignored file

git add와 함께 -f (또는 --force) 옵션을 사용하여 무시 된 파일을 저장소에 강제로 적용 할 수 있습니다.

```
$ cat .gitignore
*.log

$ git add -f debug.log

$ git commit -m "Force adding debug.log"
```

일반적인 패턴 (예 : * .log)이 정의되어 있지만 특정 파일을 커미트하려는 경우이 작업을 고려할 수 있습니다. 그러나 더 좋은 해결책은 일반적인 규칙에 대한 예외를 정의하는 것입니다.

```
$ echo !debug.log >> .gitignore

$ cat .gitignore
*.log
!debug.log

$ git add debug.log

$ git commit -m "Adding debug.log"
```

이 접근법은 팀원에게보다 분명하고 혼란스럽지 않습니다.

Stashing an ignored file

git stash는 로컬 변경 사항을 일시적으로 저장 및 되돌리기위한 강력한 Git 기능으로, 나중에 다시 적용 할 수 있습니다. 예상대로, 기본적으로 git stash는 무시 된 파일을 무시하고 Git에 의해 추적되는 파일에 대한 변경 사항만 숨 깁니다. 그러나, --all 옵션을 사용하여 git stash를 호출하면 무시되거나 변경되지 않은 파일에 변경 사항을 숨길 수 있습니다.

Debugging .gitignore files

복잡한 .gitignore 패턴이 있거나 여러 .gitignore 파일에 패턴이 퍼져있는 경우 특정 파일을 왜 무시하는지 추적하기가 어려울 수 있습니다. git check-ignore 명령을 -v (또는 --verbose) 옵션과 함께 사용하여 어떤 패턴이 특정 파일을 무시하게하는지 결정할 수 있습니다.

```
$ git check-ignore -v debug.log
.gitignore:3:*.log debug.log
```

결과:

```
<file containing the pattern> : <line number of the pattern> : <pattern>
<file name>
```

원하는 경우 여러 파일 이름을 git check-ignore에 전달할 수 있으며 이름 자체는 저장소에있는 파일에 해당 할 필요조차 없습니다.