jinyPHP 3권 - 06. 파일제어

**06. 파일제어**

====================

컴퓨터에서 수많은 데이터를 파일 형태로 하드디스크 저장 장치에 저장을 합니다. 파일 시스템은 수많은 파일들을 관리하고 입출력을 처리합니다. C언어와 같이 전형적인 프로그램 언어들은 파일 입출력을 통하여 데이터를 저장하고 관리를 합니다.

PHP는 서버사이트 웹 인터프리터 언어 입니다. 하지만 PHP는 파일시스템의 다양한 파일들을 입출력 제어함으로써 데이터를 처리할 수 있습니다. 이와 관련하여 다양한 파일을 처리할 수 있는 내장함수들을 제공하고 있습니다. 파일관련 함수들은 파일을 읽고, 쓰고, 디렉토리, 권한등을 제어할 수 있습니다.

PHP에서 파일시스템의 파일들을 제어한다는 것은 보다 향상된 PHP 응용 프로그램을 개발할 수 있다는 것입니다. 템플릿 파일들을 제어하고, 환경 설정을 파일로 저장할 수 있습니다. 또한 각종 로그기록등을 처리하는데 파일처리 함수들은 매우 유용합니다.

**06.1 파일시스템**

====================

파일시스템은 파일들을 관리하는 운영체제의 한 일부분 입니다. 파일시스템은 운영체제별로 차이가 있으며, 다양한 형태의 시스템이 존재합니다. FAT, FAT32, NTFS, Ext3 등 운영체제와 밀접한 다양한 파일 시스템 구조들이 있습니다.

운영체제 및 파일시스템의 특성에 따라서 처리하는 PHP 개발 환경은 약간씩 차이가 발생합니다. 예를들어 유닉스 플랫폼은 디렉토리 구분을 슬래쉬(/)를 통하여 구분을 합니다. 하지만 윈도우 플랫폼은 디렉토리 구분을 백슬래시(\)를 통하여 구분을 합니다.

또한 윈도우 시스템의 파일시스템과 리눅스/Mac 등의 파일시스템의 차이로 인하여 몇몇 기능들은 동작을 하지 않을 수 있습니다. 파일처리 함수를 사용할때는 이런점들을 주의하면서 개발하여야 합니다.

PHP 파일처리 함수 및 기능은 PHP 코어에 기본적으로 탑재되어 있습니다. 별도의 외부 모듈을을 추가하지 않아도 바로 사용이 가능합니다. PHP의 파일처리 함수들은 php.ini 환경설정의 영향을 받습니다.

**06.2 디렉토리**

====================

파일에 대해서 먼저 학습을 하기 전에 디렉토리 시스템에 대해서 살펴보도록 하겠습니다. 디렉토리는 윈도우와 같은 시스템에서는 “폴더”라고도 부르기도 합니다. 서로 같은 의미라고 보시면 됩니다.

디렉토리는 수많은 파일들을 구분하고 정리하기 위해서 고안된 개념입니다. 초창기 하드디스크의 용량이 매우 적고 파일들이 몇 개 안될 때에는 큰 문제가 없었습니다. 하지만, 시스템이 커지고 파일들도 많아짐으로서 유사한 파일들을 묽어 관리해야하는 필요성이 생겼습니다.

디렉토리는 파일과 다르게 하나의 묽음공간이며, 여러 개의 파일들을 관리합니다. 또한 한 개의 디렉토리는 또 다른 여러 개의 디렉토리를 관리할 수 있습니다.

**06.2.1 디렉토리 확인**

====================

파일시스템은 파일과 디렉토리를 한곳에 같이 출력 관리를 합니다. 대부분의 파일들은 확장자를 포함하고, 디렉토리의 경우에는 확장자를 가지지 않습니다. 하지만, 파일명이 꼭 확장자를 가져야 하는 것은 아니기 때문에 입력되는 값이 파일인지, 디렉토리인지를 구분하기 어려울 수 있습니다.

디렉토리 관련 작업을 할때에는 먼저 입력한 값이 디렉토리인지를 먼저 확인하는 것이 좋습니다. 또한 존재하는 디렉토리명 인지도 같이 검사를 해야 합니다.

우리는 보통 경로를 지정할때는 상대경로, 절대경로등 다양한 방법을 사용합니다. 이러한 다양한 경로의 표현은 자칫 실수로 인하여 오류를 발생할 수 있습니다. 존재하지 않은 디렉토리 접근, 정확하지 않은 디렉토리를 접근은 프로그렘에 심각한 오류를 발생할 수 있습니다.

사전에 디렉토리를 확인하는 코드는, 파일처리를 하는데 있어서 발생할 수 있는 오류들을 줄여주고, 코드의 안정적인 동작을 도와 줍니다.

PHP에서는 디렉토리의 존재 여부를 확인해 줄 수 있는 다음과 같은 내장 함수를 지원합니다.

**|내부함수|**

|  |
| --- |
| bool **is\_dir** ( string $filename ) |

디렉토리를 확인하는 is\_dir() 함수는 매개변수로 입력된 값이 디렉토리인지 파일인지를 확인하여 논리값으로 반환을 합니다.

**예제파일) file\_dir-01.php**

|  |
| --- |
| <?php  $pathDir = "testing";  if (!is\_dir($pathDir)) {  echo "Err] $pathDir 디렉토리가 존재하지 않습니다.<br>";  } else {  echo "OK] $pathDir 디렉토리 작업이 가능합니다.<br>";  }  $pathDir = "abcd";  if (!is\_dir($pathDir)) {  echo "Err] $pathDir 디렉토리가 존재하지 않습니다.<br>";  } else {  echo "OK] $pathDir 디렉토리 작업이 가능합니다.<br>";  }  $pathDir = "abcd/123";  if (!is\_dir($pathDir)) {  echo "Err] $pathDir 디렉토리가 존재하지 않습니다.<br>";  } else {  echo "OK] $pathDir 디렉토리 작업이 가능합니다.<br>";  }  ?> |

**화면출력)**

Err] testing 디렉토리가 존재하지 않습니다.

OK] abcd 디렉토리 작업이 가능합니다.

OK] abcd/123 디렉토리 작업이 가능합니다.

위의 실습은 실행하고 있는 스크립트 안에 있는 디렉토리 여부를 확인합니다. 입력한 경로값의 디렉토리가 존재하는지, 또는 파일이 아닌 유효한 디렉토리 인지를 논리값으로 반환을 합니다. 디렉토리 검사는 서브 디렉토리를 포함하여 한번에 검사를 할 수도 있습니다.

**06.2.2 디렉토리 생성**

====================

PHP코드를 통하여 새로운 디렉토리를 생성할 수 있습니다. 디렉토리를 신규로 생성을 하기 위해서는 PHP 또는 사용자가 해당 디렉토리에 쓰기 권한이 있어야 합니다. 디렉토리는 수많은 파일들을 관리하는데 매우 유용합니다.

PHP에서는 새로운 디렉토리 생성을 위한 다음과 같은 내장 함수를 지원합니다.

**|내부함수|**

|  |
| --- |
| bool **mkdir** ( string $pathname [, int $mode = 0777 [, bool $recursive = false [, resource $context ]]] ) |

PHP 내부함수 mkdir()는 새로운 디렉토리를 생성합니다. 함수명이 리눅스계열의 운영체제에서 디렉토리 생성 명령인 mkdir 과 매우 유사합니다.

mkdir() 함수는 몇개의 매개변수를 동시에 입력받습니다. 하지만 기본적으로 신규 디렉토리만 생성을 할때에는 경로명만 입력하면 됩니다. 만일, 폴더 권한등을 동시에 설정이 필요할때는 두번째 권한 설정을 같이 입력하면 됩니다.

**예제파일) file\_dir-02.php**

|  |
| --- |
| <?php  $pathDir = "aaa";  if (!is\_dir($pathDir)) {  echo "Err] $pathDir 디렉토리가 존재하지 않습니다.<br>";  echo "새로운 디렉토리를 생성합니다.<br>";  if (mkdir($pathDir)) {  echo "$pathDir 디렉토리를 생성하였습니다.<br>";  } else {  echo "Err] 디렉토리 생성 실패! <br>";  }  } else {  echo "OK] $pathDir 디렉토리 작업이 가능합니다.<br>";  }  ?> |

**화면출력)**

Err] aaa 디렉토리가 존재하지 않습니다.

새로운 디렉토리를 생성합니다.

aaa 디렉토리를 생성하였습니다.

**06.2.3 서브 디렉토리**

====================

디렉토리는 다수의 또 다른 디렉토리를 포함하고 관리할 수 있습니다. 따라서, 디렉토리 안에 새로운 서브 디렉토리를 추가하여 생성을 할 수 있습니다.

PHP 내장함수인 mkdir()은 새로운 디렉토리를 생성을 합니다. 만일 서브 디렉토리를 생성할때는 상위 디렉토리가 존재하여야 합니다. 상위 디렉토리가 없이 하위 디렉토리를 가지는 멀티레벨의 디렉토를 한번에 생성을 할 수 없습니다.

다수의 단계를 가지는 디렉토리를 생성할때는 항상 상위 단계의 디렉토리가 미리 있는지 확인을 해야 합니다. 이러한 사전 디렉토리 존재 여부를 확인하는 것은 코드상 복잡한 구조를 가지게 됩니다. 이런경우에는 mkdir()함수의 세번째 recursive 옵션을 같이 사용하면 편리합니다. 이 옵션값을 true 로 설정을 하면 다단계의 서브패스 디렉토리를 한번에 생성을 할 수 있습니다.

다음은 recursive 옵션을 이용하여 다단계 디렉토리를 한번 생성을 하는 예제코드 입니다.

**예제파일) file\_dir-03.php**

|  |
| --- |
| <?php  $pathDir = "aa/bb/cc";  if (!is\_dir($pathDir)) {  echo "Err] $pathDir 디렉토리가 존재하지 않습니다.<br>";  echo "새로운 디렉토리를 생성합니다.<br>";  if (mkdir($pathDir, 0777, true)) {  echo "$pathDir 디렉토리를 생성하였습니다.<br>";  } else {  echo "Err] 디렉토리 생성 실패! <br>";  }  } else {  echo "OK] $pathDir 디렉토리 작업이 가능합니다.<br>";  }  ?> |

**출력화면)**

Err] aa/bb/cc 디렉토리가 존재하지 않습니다.

새로운 디렉토리를 생성합니다.

aa/bb/cc 디렉토리를 생성하였습니다.

**06.2.4 파일목록**

====================

디렉토리는 다수의 파일들을 포함하고 있습니다. 또한 여러 개의 디렉토리도 포함을 하고 있습니다. PHP는 디렉토리 안에 있는 파일들의 목록을 쉽게 처리할 수 있는 내장함수를 지원합니다.

**|내부함수|**

|  |
| --- |
| array **scandir** ( string $directory [, int $sorting\_order = SCANDIR\_SORT\_ASCENDING [, resource $context ]] ) |

내장함수 scandir()는 입력된 디렉토리의 경로에 존재하는 모든 파일명과 디렉토리명을 배열 형태로 반환을 합니다. 또한 두번째 매개변수 인자를 통하여 파일의 정렬 순서를 지정할 수 있습니다.

0 : 오름차순(기본값)

1: 내림차순

**예제파일) file\_dir-04.php**

|  |
| --- |
| <?php  $pathDir = "./";  if (is\_dir($pathDir)) {  echo "OK] $pathDir 디렉토리 작업이 가능합니다.<br>";  $dirARR = scandir($pathDir);  for ($i=0;$i<count($dirARR);$i++) {  echo $dirARR[$i]."<br>";  }  } else {  echo "Err] $pathDir 디렉토리가 존재하지 않습니다.<br>";  }  ?> |

**화면 출력)**

OK] ./ 디렉토리 작업이 가능합니다.

.

..

aa

abcd

file\_dir-01.php

file\_dir-02.php

file\_dir-03.php

file\_dir-04.php

file\_dir-05.php

hello.txt

입력한 디렉토리 경로의 파일과 디렉토리 목록을 출력합니다. 출력시 상위 디렉토리를 의미하는 . 와 .. 도 같이 출력이 됩니다. 이는 시스템에서 출력하는 디렉토리 목록 그대로 반환하기 때문입니다.

내부함수 dir()은 Directory 클래스의 인스턴스을 통하여 디렉토리의 목록을 출력할 수 있습니다. 주어진 디렉토리가 열립니다.

**|내부함수|**

|  |
| --- |
| Directory **dir** ( string $directory [, resource $context ] ) |

**예제파일) dir.php**

|  |
| --- |
| <?php  $d = dir("/home/php7");  echo "Handle: " . $d->handle . "\n";  echo "Path: " . $d->path . "\n";  while (false !== ($entry = $d->read())) {  echo $entry."<br>";  }  $d->close();  ?> |

**화면출력)**

Handle: Resource id #3 Path: /home/php7 .

..

readme.txt

내장함수 glob ()는 쉘에서 사용되는 규칙과 비슷한 libc glob () 함수에 따라서 패턴과 일치하는 모든 경로 이름을 검색합니다.

**|내장함수|**

|  |
| --- |
| array glob ( string $pattern [, int $flags = 0 ] ) |

**예제) glob.php**

|  |
| --- |
| <?php  foreach (glob("\*.exe") as $filename) {  echo "$filename size " . filesize($filename) . "<br>";  }  ?> |

**화면출력)**

deplister.exe size 96768

php-cgi.exe size 57856

php-win.exe size 33280

php.exe size 101376

phpdbg.exe size 274432

**06.2.5 경로**

====================

파일시스템은 다단계의 여러 디렉토리로 이루어저 있습니다. 현재의 디렉토리 또는 다른곳을 지정하는 상태 디렉토리를 지정할 수 있습니다.

PHP에서 실행되는 파일의 기본 경로는 현재 실행되는 스크립트의 위치 입니다. 따라서 다른 경로를 지정하기 위해서는 ./\*\*\*/\*\*\* 처럼 서브경로 또는 상위경로 .././ 처럼 상대경로를 지정하면 됩니다.

PHP는 실행파일 및 경로를 확인할 수 있는 몇가지 내장함수들을 제공하고 있습니다.

**|내부함수| 현재 경로 확인**

|  |
| --- |
| string **getcwd** ( void ) |

내장함수 getcwd()는 현재의 스크립트가 동작하고 있는 현재의 경로를 반환합니다.

**|내부함수| 디렉토리 패스경로**

|  |
| --- |
| string **dirname** ( string $path [, int $levels = 1 ] ) |

내부함수 dirname()는 입력한 경로명에서 현재의 자신의 디렉토리를 제외한 상위의 경로를 출력 합니다.

**예제파일) file\_dir-05.php**

|  |
| --- |
| <?php    // 현재의 디렉토리를 확인합니다.  $path = getcwd();  echo $path."<br>";  echo "== 상위 디렉토리 패스 == ".dirname($path)."<br>";  echo "<br>";  $pathDir = "aa";  echo "절대경로 ==> ".realpath("./".$pathDir)."<br><br>";  if (!is\_dir($pathDir)) {  echo "Err] $pathDir 디렉토리가 존재하지 않습니다.<br>";  } else {  echo "OK] $pathDir 디렉토리 작업이 가능합니다.<br>";    if (chdir($pathDir)) {  echo "작업 디렉토리를 $pathDir 로 변경합니다.<br>";  } else {  echo "Err] 작업 디렉토리 변경 실패<br>";  }  }  // 현재의 디렉토리를 확인합니다.  $path = getcwd();  echo $path."<br>";  echo "== 상위 디렉토리 패스 == ".dirname($path)."<br>";  echo "<br>";  ?> |

**화면출력)**

C:\php-7.1.4-Win32-VC14-x86\03

== 상위 디렉토리 패스 == C:\php-7.1.4-Win32-VC14-x86

절대경로 ==> C:\php-7.1.4-Win32-VC14-x86\03\aa

OK] aa 디렉토리 작업이 가능합니다.

작업 디렉토리를 aa 로 변경합니다.

C:\php-7.1.4-Win32-VC14-x86\03\aa

== 상위 디렉토리 패스 == C:\php-7.1.4-Win32-VC14-x86\03

**5.2.6 리얼패스**

====================

내장함수 realpath() 함수는 입력한 경로명에 대해서 시스템의 절대 경로를 반환합니다. 입력되는 경로명은 상대경로명을 사용할 수 있습니다. 만일, 상태경로명을 입력하면, 입력한 상태 경로의 절대 경로값을 출력합니다.

**|내부함수| 절대경로**

|  |
| --- |
| string **realpath** ( string $path ) |

내부함수 realpath\_cache\_get()는 realpath 캐시 엔트리 항목을 배열로 반환합니다. 확인 된 경로, 만료 날짜 및 캐시에 보관 여러 옵션들을 반환합니다.

**|내부함수| 캐시정보**

|  |
| --- |
| array **realpath\_cache\_get** ( void ) |

**예제파일) realpath\_cache\_get.php**

|  |
| --- |
| <?php  var\_dump(realpath\_cache\_get());  ?> |

**화면출력)**

array(3) { ["C:\php-7.1.4-Win32-VC14-x86\date\_get\_last\_errors.php"]=> array(8) { ["key"]=> float(2823479507) ["is\_dir"]=> bool(false) ["realpath"]=> string(52) "C:\php-7.1.4-Win32-VC14-x86\date\_get\_last\_errors.php" ["expires"]=> int(1502349080) ["is\_rvalid"]=> bool(false) ["is\_wvalid"]=> bool(false) ["is\_readable"]=> bool(false) ["is\_writable"]=> bool(false) } ["C:\php-7.1.4-Win32-VC14-x86\realpath\_cache\_get.php"]=> array(8) { ["key"]=> float(4089114914) ["is\_dir"]=> bool(false) ["realpath"]=> string(50) "C:\php-7.1.4-Win32-VC14-x86\realpath\_cache\_get.php" ["expires"]=> int(1502349085) ["is\_rvalid"]=> bool(false) ["is\_wvalid"]=> bool(false) ["is\_readable"]=> bool(false) ["is\_writable"]=> bool(false) } ["C:\php-7.1.4-Win32-VC14-x86"]=> array(8) { ["key"]=> float(3046037941) ["is\_dir"]=> bool(true) ["realpath"]=> string(27) "C:\php-7.1.4-Win32-VC14-x86" ["expires"]=> int(1502349080) ["is\_rvalid"]=> bool(false) ["is\_wvalid"]=> bool(false) ["is\_readable"]=> bool(false) ["is\_writable"]=> bool(false) } }

내부함수 realpath\_cache\_size()는 realpath 캐시가 사용하는 메모리 양을 리턴합니다.

**|내부함수| 캐시 사이즈**

|  |
| --- |
| int **realpath\_cache\_size** ( void ) |

**예제파일) realpath\_cache\_size.php**

|  |
| --- |
| <?php  var\_dump(realpath\_cache\_size());  ?> |

화면출력)

int(328)

**05.2.7 디렉토리 삭제**

====================

PHP코드를 통하여 디렉토리를 삭제할 수 있습니다. 디렉토리를 삭제하기 위해서는 시스템의 디렉토리 쓰기 권한이 설정되어 있어야 합니다. 또한 생성한 디렉토리를 삭제하기 위해서는 안에 존재하고 있는 파일이 없어야 합니다. 만일 파일을 포함하고 있는 폴더를 삭제하고자 할 경우에는 E\_WARNING 오류가 발생합니다.

PHP에서는 기존 디렉토리 삭제를 위한 다음과 같은 내장 함수를 지원합니다.

**|내장함수|**

|  |
| --- |
| bool **rmdir** ( string $dirname ) |

내장함수 rmdir()는 디렉토리를 삭제합니다. 보통 리눅스계열의 운영체제서 디렉토리 삭제시 rmdir 명령을 사용하는 것과 유사한 함수 이름을 사용하고 있습니다.

**예제파일) file\_dir-06.php**

|  |
| --- |
| <?php  $pathDir = "aaa";  if (is\_dir($pathDir)) {  echo "OK] $pathDir 디렉토리 작업이 가능합니다.<br>";  echo "새로운 디렉토리를 삭제합니다.<br>";  if (rmdir($pathDir)) {  echo "$pathDir 디렉토리를 삭제하였습니다.<br>";  } else {  echo "Err] 디렉토리 삭제 실패! <br>";  }  } else {  echo "Err] $pathDir 디렉토리가 존재하지 않습니다.<br>";  }  ?> |

**화면출력)**

OK] aaa 디렉토리 작업이 가능합니다.

새로운 디렉토리를 삭제합니다.

aaa 디렉토리를 삭제하였습니다.

**5.2.8 디렉토리 이름변경**

====================

PHP코드를 통하여 디렉토리의 이름을 변경할 수 있습니다. 디렉토리의 이름을 변경하기 위해서는 쓰기 권한이 필요로 합니다.

**|관련함수|**

|  |
| --- |
| bool **rename** ( string $oldname , string $newname [, resource $context ] ) |

내장함수 rename()는 디렉토리 이름을 변경합니다. 입력 매개변수로는 두개의 값을 전달 받습니다. 바꾸기 전의 oldname 이름과 새로이 변경되는 newname 이름입니다.

내장함수 rename()는 새로운 이름으로 디렉토리의 명을 바꾸고, 필요할 경우 디렉토리간 이동을 합니다.

**예제파일) rename.php**

|  |
| --- |
| <?php  rename("aaa", "bbb");  ?> |

**5.2.9 경로변경**

====================

내부함수 chdir() 는 현재의 실행되고 있는 스크립트 경로에서 다른 상태 디렉토리를 변경합니다. 이는 현재의 실행되는 스크립트의 경로를 내부적으로 변경합니다.

**|내부함수| 경로변경**

|  |
| --- |
| bool **chdir** ( string $directory ) |

내장함수 chroot()는 현재의 디렉토리에서 입력한 루트 디렉토리로 변경합니다.

**|내장함수|**

|  |
| --- |
| bool **chroot** ( string $directory ) |

윈도우 환경에서는 이 함수를 사용할 수 없습니다. 리눅스 계열의 운영체제에서만 사용이 가능합니다. 이 함수를 사용하기 위해서는 루트 권한이 필요로 합니다.

**예제파일) chroot.php**

|  |
| --- |
| <?php  chroot("/path/to/your/chroot/");  echo getcwd();  ?> |

**5.2.10 디렉토리 접근**

====================

디렉토리 접근 및 처리를 위해서 좀더 세분화된 기능의 PHP 내장 함수들을 지원합니다.

내부함수 opendir()는 directory handle 오픈 생성합니다.

**|내부함수|**

|  |
| --- |
| resource **opendir** ( string $path [, resource $context ] ) |

디렉토리를 오픈 후에는 closir (), readdir () 및 rewinddir () 와 같은 함수로 접근을 할 수 있습니다.

dir\_handle가 가리키는 디렉토리 스트림을 닫습니다. 스트림은 opendir ()에 의해 열렸저 있는 스트림 이어야 합니다.

**|내부함수|**

|  |
| --- |
| void **closedir** ([ resource $dir\_handle ] ) |

**예제파일) opendir.php**

|  |
| --- |
| <?php  $dir = "/home/php7/";  // O디렉토리를 열고 내용을 읽습니다.  if (is\_dir($dir)) {  if ($dh = opendir($dir)) {  while (($file = readdir($dh)) !== false) {  echo "파일명: $file : 파일타입: " . filetype($dir . $file) . "<br>";  }  closedir($dh);  }  } else {  echo $dir . " 디렉토리가 존재하지 않습니다.";  }  ?> |

**화면출력)**

파일명: . : 파일타입: dir

파일명: .. : 파일타입: dir

파일명: readme.txt : 파일타입: file

**예제파일) closedir.php**

|  |
| --- |
| <?php  $dir = "/home/php7/";  // 디렉토리를 열고 디렉토리를 변수로 읽어 들인후에 닫습니다.  if (is\_dir($dir)) {  if ($dh = opendir($dir)) {  $directory = readdir($dh);  closedir($dh);  }  }  ?> |

내부함수 readdir()는디렉토리 핸들에서 항목 읽어 옵니다. 파일 시스템이 저장 한 순서대로 항목의 이름을 리턴됩니다.

**|내부함수|**

|  |
| --- |
| string **readdir** ([ resource $dir\_handle ] ) |

**예제파일) readdir.php**

|  |
| --- |
| <?php  if ($handle = opendir('/home/php7')) {  echo "Directory handle: $handle <br>";  echo "Entries:<br>";  echo "디렉토리를 반복하는 올바른 방법입니다. <br>";  while (false !== ($entry = readdir($handle))) {  echo "$entry <br>";  }  echo "디렉토리를 반복하는 잘못된 방법입니다. <br>";  while ($entry = readdir($handle)) {  echo "$entry <br>";  }  closedir($handle);  }  ?> |

**화면출력)**

Directory handle: Resource id #3

Entries:

디렉토리를 반복하는 올바른 방법입니다.

.

..

readme.txt

디렉토리를 반복하는 잘못된 방법입니다.

내부함수 rewinddir()은 디렉토리 핸들을 처음으로 다시 초기화 합니다.

**|내부함수|**

|  |
| --- |
| void **rewinddir** ([ resource $dir\_handle ] ) |

**예제파일) rewinddir.php**

|  |
| --- |
| <?php  if ($handle = opendir('/home/php7')) {  echo "Directory handle: $handle <br>";  echo "Entries:<br>";  echo "디렉토리를 반복하는 올바른 방법입니다. <br>";  while (false !== ($entry = readdir($handle))) {  echo "$entry <br>";  }  **// 처음으로 다시 초기화**  **rewinddir($handle);**  echo "디렉토리를 반복하는 잘못된 방법입니다. <br>";  while ($entry = readdir($handle)) {  echo "$entry <br>";  }  closedir($handle);  }  ?> |

**화면출력)**

Directory handle: Resource id #3

Entries:

디렉토리를 반복하는 올바른 방법입니다.

.

..

readme.txt

디렉토리를 반복하는 잘못된 방법입니다.

**.**

**..**

**readme.txt**

**06.2.11 디렉토리 용량 표시**

====================

파일작업을 하기 위해서 디스크의 저장 용량은 매우 중요합니다. 서버의 용량이 매우 커서 신경을 쓸 필요가 없이 사용을 하는 경우는 없을 것입니다. 만일 디스크의 용량이 부족하다면 정상적인 파일 처리 및 작업을 할 수 없을 것입니다.

PHP에서는 디렉토리의 용량을 확인할 수 있는 다음과 같은 내장 함수를 지원합니다.

**|내장함수| 전체공간**

|  |
| --- |
| float **disk\_total\_space** ( string $directory ) |

내장함수 disk\_total\_space()는 현재 디스크의 전체 용량을 확인 할 수 있습니다.

**|내장함수| 여유공간**

|  |
| --- |
| float **disk\_free\_space** ( string $directory ) |

내장함수 disk\_free\_space() 는 현재 작업하는 디렉토리의 남아 있는 여분의 용량을 확인 할 수 있습니다.

**예제파일) diskspace.php**

|  |
| --- |
| <?php  $pathDir = "./";  if (is\_dir($pathDir)) {  echo "OK] $pathDir 디렉토리 작업이 가능합니다.<br>";  $totalSapce = disk\_total\_space($pathDir);  $freeSpace = disk\_free\_space($pathDir);  echo "여유공간:".$freeSpace." / 전체공간: $totalSapce <br>";  } else {  echo "Err] $pathDir 디렉토리가 존재하지 않습니다.<br>";  }  ?> |

**화면출력)**

OK] ./ 디렉토리 작업이 가능합니다.

여유공간:48003436544 / 전체공간: 239147675648

**06.3 권한설정**

====================

요즘들어 대부분의 운영체제들은 멀티 사용자 환경을 지원함에 지원합니다. 따라서 파일시스템은 각각 사용자와 관련된 파일, 디렉토리 접근을 처리할 수 있도록 권한을 부여합니다.

이러한 권한들은 파일시스템을 효과적으로 운영 관리하는데 매우 유용합니다. 만일, PHP를 실행하는 권한이 슈퍼유저를 대상으로 몇 개의 권한관련 내부함수를 제공하고 있습니다.

**06.3.1 소유권 변경**

====================

멀티 유저환경의 파일시스템은 각각의 파일에 대해서 소유권 설정이 되어 있습니다. 즉, 해당 파일을 생성하고 관리하는 사용자를 말합니다.

내부함수 chown() 은 파일들의 소유권을 변경합니다. 소유권을 변경하기 위해서는 PHP실행이 슈퍼유저로 실행되어야 합니다.

**|내부함수|**

|  |
| --- |
| bool **chown** ( string $filename , mixed $user ) |

**예제) chown.php**

|  |
| --- |
| <?php  // 파일명과 사용자  $file\_name= "test.php";  $path = "/home/php7/" . $file\_name ;  $user\_name = "root";  // 소유권을 변경합니다.  chown($path, $user\_name);  // 결과를 확인합니다.  $stat = stat($path);  print\_r(posix\_getpwuid($stat['uid']));  ?> |

소유권을 변경한는 것과 달리 파일의 소유권을 읽어 올 수도 있습니다. 내부함수 fileowner()는 파일의 소유자 정보를 읽어 옵니다.

**|내부함수|**

|  |
| --- |
| int **fileowner** ( string $filename ) |

**예제) fileowner.php**

|  |
| --- |
| <?php  $filename = 'fileowner.php';  print\_r(posix\_getpwuid(fileowner($filename)));  ?> |

**화면출력)**

Array ( [name] => jinyshop [passwd] => x [uid] => 519 [gid] => 520 [gecos] => [dir] => /home/jinyshop [shell] => /bin/bash )

**06.3.2 모드변경**

====================

파일시스템은 소유권이외에도 파일의 접근권한을 가지고 있습니다. 해당 파일을 읽기만 가능하게 할것인지, 아니면 읽기/쓰기 모두를 허용할 것인지를 설정할 수 있습니다. 또한 파일의 실행권한도 부여할 수 있습니다.

내장함수 chmod()는 파일의 모드를 변경할 수 있습니다. 모드를 변경후에는 성공여부를 논리값으로 반환을 합니다.

**|내장함수|**

|  |
| --- |
| bool **chmod** ( string $filename , int $mode ) |

**예제) chmod.php**

|  |
| --- |
| <?php  if (chmod("test.txt",0755)) {  echo "모드변경";  } else {  echo "변경실패";  }  ?> |

내장함수 umask()는 현재의 unmask를 변경합니다.

**|내장함수|**

|  |
| --- |
| int **umask** ([ int $mask ] ) |

**예제) umask.php**

|  |
| --- |
| <?php  $old = umask(0);  chmod("/path/some\_dir/some\_file.txt", 0755);  umask($old);  // Checking  if ($old != umask()) {  die('An error occurred while changing back the umask');  }  ?> |

파일의 퍼미션을 권한을 읽어 올 수 있습니다. 내장함수 fileperms()는 파일의 권한값을 정수값 형태로 반환을 합니다. 반환되는 정수값은 우리가 알기 쉬운형태의 코드로 확인을 하기 위해서는 별도의 처리가 필요로 합니다.

**|내부함수|**

|  |
| --- |
| int **fileperms** ( string $filename ) |

**예제) fileperms.php**

|  |
| --- |
| <?php  $permissions = fileperms("fileperms.php");  echo "permissions = " . $permissions . "<br>";  // Owner  $info .= (($permissions & 0x0100) ? 'r' : '-');  $info .= (($permissions & 0x0080) ? 'w' : '-');  $info .= (($permissions & 0x0040) ?  (($permissions & 0x0800) ? 's' : 'x' ) :  (($permissions & 0x0800) ? 'S' : '-'));  // Group  $info .= (($permissions & 0x0020) ? 'r' : '-');  $info .= (($permissions & 0x0010) ? 'w' : '-');  $info .= (($permissions & 0x0008) ?  (($permissions & 0x0400) ? 's' : 'x' ) :  (($permissions & 0x0400) ? 'S' : '-'));  // World  $info .= (($permissions & 0x0004) ? 'r' : '-');  $info .= (($permissions & 0x0002) ? 'w' : '-');  $info .= (($permissions & 0x0001) ?  (($permissions & 0x0200) ? 't' : 'x' ) :  (($permissions & 0x0200) ? 'T' : '-'));  echo $info;  ?> |

화면출력)

permissions = 33188

rw-r--r--

파일 권한 중에는 실행가능한 권한모드가 있습니다. 입력한 경로의 파일이 실행가능한 상태권한인지를 확인할 수 있는 내부함수 is\_executable()를 제공합니다.

**|내장함수|**

|  |
| --- |
| bool **is\_executable** ( string $filename ) |

파일의 실행가능 여부를 논리값 형태로 반환을 합니다.

**예제) is\_executable.php**

|  |
| --- |
| <?php  $file = 'check.sh';  if (is\_executable($file)) {  echo $file.' is executable';  } else {  echo $file.' is not executable';  }  ?> |

**06.3.3 그룹변경**

====================

리눅스와 같이 멀티 사용자를 지원하는 운영체제, 파일시스템의 경우 그룹이라는 권한 설정이 있습니다. 그룹은 여러 사용자를 하나의 그룹으로 묽어서 권한을 관리하는 개념입니다.

PHP코드를 통하여 파일들의 그룹을 변경할 수 있습니다. 파일 그룹을 변경하기 위해서는 root 권한이 필요로 합니다.

보통 웹페이지들은 nobody 그룹을 많이 사용을 하는데 권한에 따라서 chgrp() 함수를 통하여 파일의 그룹을 변경할 수 있습니다.

**|내장함수|**

|  |
| --- |
| bool **chgrp** ( string $filename , mixed $group ) |

**예제) chgrp.php**

|  |
| --- |
| <?php  $filename = 'install.txt';  $format = "%s's Group ID @ %s: %d <br>";  printf($format, $filename, date('r'), filegroup($filename));  // 100  chgrp($filename, 8);  // filegroup () 결과를 캐시하지 않습니다.  clearstatcache();  printf($format, $filename, date('r'), filegroup($filename));  ?> |

설정된 파일 그룹 정보를 가지고 와서 확인을 할 수 있습니다. 내부함수 filefroup()은 파일이 속해져 있는 급룩의 ID 숫자값을 반환합니다.

|내장함수|

|  |
| --- |
| int **filegroup** ( string $filename ) |

**예제) filegroup.php**

|  |
| --- |
| <?php  $filename = 'index.php';  $filegroup\_Id = filegroup($filename);  echo "filegroup ID = " . $filegroup\_Id . "<br>";  print\_r(posix\_getgrgid($filegroup\_Id));  ?> |

**화면출력)**

filegroup ID = 520

Array ( [name] => jinyshop [passwd] => x [members] => Array ( ) [gid] => 520 )

**06.4 파일정보**

====================

파일은 디스크 공간에 존재하는 데이터입니다. PHP 코드상에서 새로운 파일을 생성할 수 있고, 또한 존재하는 파일을 읽어서 처리를 할 수도 있습니다.

생성된 파일을 좀더 안전하게 접근 처리를 하기 위해서는 파일들의 상태정보들을 잘 확인 하는 것이 중요합니다. PHP는 파일들의 상태들을 확인할 수 있는 다양한 내부함수들을 제공하고 있습니다.

**06.4.1 파일 확인**

====================

안전한 파일처리를 위해서는 유효한 파일경로를 접근하여 처리를 해야 합니다. 입력한 파일경로가 존재하지 않은 파일이거나, 파일이 아닌 디렉토리일 경우에는 오류가 발생됩니다.

잘못된 파일경로로 인하여 발생될 수 있는 오류를 줄이기 위해서는 먼저 파일이 존재여부를 먼저 확인 하는 것이 중요합니다. PHP에서는 기존 디렉토리 삭제를 위한 다음과 같은 내장 함수를 지원합니다.

**|내장함수|**

|  |
| --- |
| bool **file\_exists** ( string $filename ) |

내장함수 file\_exists()는 파일의 존재 여부를 확인하여 논리값으로 결과를 반한합니다.

**예제파일) file\_exists.php**

|  |
| --- |
| <?php  $filename = "./readme.txt";  if (**file\_exists($filename)**) {  echo $filename." 파일명이 존재합니다.<br>";  } else {  echo "파일을 찾을 수 없습니다.<br>";  }  ?> |

**화면출력)**

./readme.txt 파일명이 존재합니다.

입력한 경로의 파일이 존재한다면 다시한번 입력한 경로 파일이 실제 파일인지를 확인하는 것이 좋습니다. 내장함수 is\_file()은 입력한 경로가 파일인지를 확인하여 논리값으로 반환을 합니다.

**|내부함수|**

|  |
| --- |
| bool **is\_file** ( string $filename ) |

**예제) is\_file.php**

|  |
| --- |
| <?php  var\_dump(is\_file('is\_file.php'));  ?> |

**화면출력)**

bool(true)

지정한 경로의 파일에 대한 타입을 확인할 수 있습니다. 내장함수 filetype()은 입력한 경로에 대해서 파일 타입인지, 디렉토리 타입인지를 분별하여 문자열을 반환합니다.

**|내장함수|**

|  |
| --- |
| string **filetype** ( string $filename ) |

**예제) filetype.php**

|  |
| --- |
| <?php  echo filetype('/etc/passwd') ."<br>";  // file  echo filetype('/etc/') ."<br>";  // dir  ?> |

**화면출력)**

file

dir

**06.4.2 날짜와 시간**

====================

모든 파일들은 자신이 생성된 날짜와 최근 수정된 시간 정보를 가지고 있습니다. 이러한 파일들의 날짜와 시간정보들은 최신 파일로 업데이트를 유지하기 위한 정보로 매우 유용합니다.

파일들의 날짜와 시간정보를 PHP 코드를 통하여 확인할 수 있습니다. PHP에서는 파일 날짜 관련 다음과 같은 내장 함수를 지원합니다.

**|내장함수| 접근일자**

|  |
| --- |
| int **fileatime** ( string $filename ) |

내장함수 fileatime()는 해당파일의 마지막 사용된 시간을 반한홥니다.

**|내장함수| 수정일자**

|  |
| --- |
| int **filemtime** ( string $filename ) |

내장함수 filemtime()는 해당파일의 마지막 수정된 시간을 반한홥니다.

**예제파일) fileinfo-01.php**

|  |
| --- |
| <?php  $filename = './readme.txt';  if (file\_exists($filename)){  echo "$filename 마지막 접근일자: " . date("F d Y H:i:s.", fileatime($filename));  echo "<br>";  echo "$filename 마지막 수정일자: " . date ("F d Y H:i:s.", filemtime($filename));  }  ?> |

**화면출력)**

./readme.txt 마지막 접근일자: June 13 2017 06:19:03.

./readme.txt 마지막 수정일자: June 13 2017 06:19:03.

내장함수 filectime()은 파일의 inode 변경 시간을 읽어 옵니다. 시간은 Unix 타임 스탬프로 반환됩니다.

**|내장함수|**

|  |
| --- |
| int **filectime** ( string $filename ) |

**예제) filectime.php**

|  |
| --- |
| <?php  $filename = 'install.txt';  if (file\_exists($filename)) {  echo "$filename 의 변경시간 : " . date("F d Y H:i:s.", **filectime**($filename));  }  ?> |

내부함수 touch()는 파일의 수정 시간을 재설정 할 수 있습니다.

|내부함수|

|  |
| --- |
| bool **touch** ( string $filename [, int $time = time() [, int $atime ]] ) |

**예제) touch.php**

|  |
| --- |
| <?php  // 터치 시간입니다  // 과거 한 시간으로 설정.  $time = time() - 3600;  // 파일의 수정 시간을 재설정합니다.  if (!touch('touch.php', $time)) {  echo 'wrong...';  } else {  echo 'success';  }  ?> |

**화면출력)**

success

내부함수 stat()은 파일에 대한 정보를 반환합니다.

**|내부함수|**

|  |
| --- |
| array **stat** ( string $filename ) |

**예제) stat.php**

|  |
| --- |
| <?php  /\* Get file stat \*/  $stat = stat('C:\php\php.exe');  /\*  \* Print file access time, this is the same  \* as calling fileatime()  \*/  echo 'Access time: ' . $stat['atime'];  /\*  \* Print file modification time, this is the  \* same as calling filemtime()  \*/  echo 'Modification time: ' . $stat['mtime'];  /\* Print the device number \*/  echo 'Device number: ' . $stat['dev'];  ?> |

**06.4.3 파일종류**

====================

PHP에서 파일을 처리할 때 경로, 파일명, 확장자 등의 정보를 path값으로 처리하여 전달합니다. 입력되는 경로값을 기준으로 파일명, 확장자, 경로등의 문자열을 분리하여 처리할 수 있는 유용한 내부함수들을 제공합니다.

**|내부함수|**

|  |
| --- |
| string **basename** ( string $path [, string $suffix ] ) |

내부함수 basename()은 입력된 경로명에서 파일명 부분만을 추출하여 분리합니다.

**예제파일) basename.php**

|  |
| --- |
| <?php  $path = "./aa/bb/cc/eee/fff/thisfile.php";  echo basename($path);  ?> |

**화면출력)**

thisfile.php

**06.4.4 파일 경로 및 확장자**

====================

파일처리시 전달되는 매개변수 파일명은 상대경로 및 절대 경로에 대한 디렉토리 구조와 파일명, 확장자 를가지고 있습니다.

|  |
| --- |
| mixed **pathinfo** ( string $path [, int $options = PATHINFO\_DIRNAME | PATHINFO\_BASENAME | PATHINFO\_EXTENSION | PATHINFO\_FILENAME ] ) |

pathinfo() 함수는 파일명의 경로를 포함한 정보를 배열로 반환을 합니다. 반환된 배열값의 키로

dirname, basename, extension, filename 를 통하여 경로값을 분석할 수 있습니다.

**예제파일) pathinfo.php**

|  |
| --- |
| <?php  $path\_parts = pathinfo('./readme.txt');  echo $path\_parts['dirname'], "<br>";  echo $path\_parts['basename'], "<br>";  echo $path\_parts['extension'], "<br>";  echo $path\_parts['filename'], "<br>"; // since PHP 5.2.0  ?> |

**화면출력)**

readme

.txt

txt

readme

**06.4.5 링크**

====================

리눅스, 맥과 같은 운영체제의 파일시스템은 링크라는 파일처리 개념이 있습니다. 링크는 물리적으로 파일이 존재하는 것이 아니라, 실제 파일과 연결된 형태의 연결 고리 입니다.

입력한 경로의 파일이 심볼링크 형태의 파일 인지를 확인합니다. 내부함수 is\_link()는 입력한 경로가 링크형태의 연결인지를 판별합니다. 반환값으로는 논리값을 출력합니다.

**|내부함수|**

|  |
| --- |
| bool **is\_link** ( string $filename ) |

**예제) is\_link.php**

|  |
| --- |
| <?php  $link = 'symlink';  if (is\_link($link)) {  echo $link . " is symbolic link";  } else {  echo $link . " is not symbolic link";  }  ?> |

**화면출력)**

symlink is symbolic link

파일에 대해서 새로운 심볼 링크를 생성합니다 내부함수 symlink()는 두개의 인자값을 전달받아 심볼 링크를 생성을 합니다.

**|내부함수|**

|  |
| --- |
| bool **symlink** ( string $target , string $link ) |

**예제) symlink.php**

|  |
| --- |
| <?php  $target = 'symlink.php';  $link = 'symlink';  symlink($target, $link);  echo readlink($link);  ?> |

**화면출력)**

symlink.php

내부함수 readlink()는 심볼릭 링크의 대상을 반환합니다.

**|내부함수|**

|  |
| --- |
| string **readlink** ( string $path ) |

하드링크 형태로 파일을 생성합니다. 내부함수 link()는 두개의 인자값을 받아 하드링크를 생성합니다.

**|내부함수|**

|  |
| --- |
| bool **link** ( string $target , string $link ) |

**예제) link.php**

|  |
| --- |
| <?php  $target = 'link.php';  $link = 'hardlink.php';  **link($target, $link);**  ?> |

내부함수 linkinfo()는 하드링크 파일의 정보를 반환합니다.

**|내부함수|**

|  |
| --- |
| int **linkinfo** ( string $path ) |

내부함수 lchgrp()는 심볼링크의 그룹 소유권을 변경합니다.

|**내부함수|**

|  |
| --- |
| bool **lchgrp** ( string $filename , mixed $group ) |

**예제) lchgrp.php**

|  |
| --- |
| <?php  $target = 'output.php';  $link = 'output.html';  symlink($target, $link);  lchgrp($link, 8);  ?> |

내부함수 lchown()은 심볼링크의 사용자 소유권 변경합니다.

**|내부함수|**

|  |
| --- |
| bool **lchown** ( string $filename , mixed $user ) |

**예제) lchown.php**

|  |
| --- |
| <?php  $target = 'output.php';  $link = 'output.html';  symlink($target, $link);  lchown($link, 8);  ?> |

**lstat()**

====================

내부함수 lstat()는 파일 또는 심볼릭 링크에 대한 정보를 반환합니다.

|  |
| --- |
| array lstat ( string $filename ) |

**예제) lstat.php**

|  |
| --- |
| <?php  symlink('uploads.php', 'uploads');  // Contrast information for uploads.php and uploads  array\_diff(stat('uploads'), lstat('uploads'));  ?> |

**06.4.6 그외처리**

====================

리눅스와 같이 유닉스 계열타입의 파일시스템은 파일에 대한 inode값을 가지고 있습니다. 내부함수 fileinode()는 입력한 경로의 inode값을 반환합니다.

**|내부함수|**

|  |
| --- |
| int **fileinode** ( string $filename ) |

**예제) fileinode.php**

|  |
| --- |
| <?php  $filename = 'fileinode.php';  echo **fileinode($filename)**."<br>";  ?> |

화면출력)

6961340

파일 상태 캐시를 지웁니다. clearstatcache () 함수를 사용하여 PHP가 파일에 대해 캐시하는 정보를 지울 수 있습니다.

**|내부함수|**

|  |
| --- |
| void **clearstatcache** ([ bool $clear\_realpath\_cache = false [, string $filename ]] ) |

파일 이름이나 문자열을 지정된 패턴과 비교합니다. fnmatch ()는 전달 된 문자열이 주어진 쉘 와일드 카드 패턴과 일치하는지 확인합니다.

**|내부함수|**

|  |
| --- |
| bool **fnmatch** ( string $pattern , string $string [, int $flags = 0 ] ) |

**예제) fnmatch.php**

|  |
| --- |
| <?php  $filename = "gray";  if (**fnmatch("\*gr[ae]y", $filename)**) {  echo "some form of gray ...";  } else {  echo "not match";  }  ?> |

화면출력)

some form of gray ...

**06.5 파일열기**

====================

PHP는 다양한 방법으로 파일을 읽고/쓰기 처리가 가능합니다. 파일의 포인터를 통하여 처리하는 방식은 C언어와 같이 오래전부터 사용을 하던 형태의 파일처리 방법입니다. PHP도 파일 포인터를 오픈하여 파일처리를 할 수 있습니다.

PHP에서 파일을 제어하기 위해서 먼저 파일 포인터를 생성하여야 합니다. 파일 포인터는 파일을 접근 하기 위해서 오픈 하는 것입니다. 단지 파일의 시작되는 포인터를 얻는 형태의 작업입니다.

PHP는 파일의 시작포인트를 관리할 수 있는 다음과 같은 내장함수를 제공합니다.

**|내장함수|**

|  |
| --- |
| resource **fopen** ( string $filename , string $mode [, bool $use\_include\_path = false [, resource $context ]] ) |

내장함수 fopen() 는 파일을 접근하고 사용을 하기 위한 포인트를 반환합니다. 매개변수 인자로 통상 2개의 값을 입력하는데, 첫번째는 경로를 포함한 파일명이고 두번째는 파일의 접근 모드입니다.

**06.5.1 파일모드**

====================

모드란 파일을 읽기용으로 오픈할 것인지, 쓰기용으로 오픈할 것인지등을 정의하는 값입니다. PHP의 파일 모드 상태는 10가지 방식으로 구분됩니다.

* r : 읽기
* r+ : 읽기 / 쓰기
* w : 쓰기
* w+ : 쓰기 / 읽기
* a : 쓰기
* a+ : 쓰기 / 일기
* x : 쓰기
* x+ : 쓰기 / 읽기
* c : 쓰기
* c+ : 쓰기/ 읽기

기본 모드와 기본모드에서 + 기호가 들어가 있는 상태로 크게 구분할 수 있습니다.

r 모드는 단순하게 파일을 읽기만 가능하도록 오픈하는 것입니다. 파일의 첫번째 위치값을 반환을 합니다.

w 모드는 파일을 쓰기전용으로 오픈하는 것입니다.파일의 포인터는 첫번째의 위치값을 반환을 합니다. w모드는 새로운 파일을 생성을 합니다. 만일 기존에 동일한 이름의 파일이 존재할 경우 모든 내용은 덥어쓰기로 지워집니다.

a 모드는 기존에 있는 파일뒤에 새로운 내용을 추가하여 쓸수 있습니다. 파일의 위치값은 맨 마지막을 가르키게 됩니다. 만일 파일이 업는 경우에는 새로운 파일을 생성합니다.

x 모드는 새로운 파일을 생성하여 쓰기로 오픈합니다. 만일, 기존에 동일한 이름의 파일이 있다면 실패로 FALSE를 반환합니다.

r+ 모드는 읽기 모드에 쓰기모드를 추가합니다. 파일의 포인터는 처음 시작 위치값을 반환합니다.

w+ 모드는 읽기와 쓰기를 지원합니다. 기존 파일의 내용을 모두 삭제하고 반환되는 시작 포인터는 처음을 가르킵니다. 만일, 파일이 존재하지 않으면 새로운 파일을 생성합니다.

a+ 모드는 기존 모드에 추가로 쓰기 기능과 읽기를 지원합니다. 파일의 반환 포인터는 맨 마지막을 가르킵니다. 만일 파일이 존재하지 않으면 새로운 파일을 생성합니다.

x+ 모드는 파일을 읽기와 쓰기를 지원합니다. 만일 기존에 파일이 존재하면 FALSE를 반환합니다.

**06.5.2 파일닫기**

====================

내장함수 fopen() 로 오픈한 파일포인트는 리소스 관리를 위해서 사용후에 반드시 닫아 주어야 합니다. fclose() 함수 매개변수로 fopen() 함수로 받은 리소스 포인터를 전달하면 됩니다.

**|내장함수|**

|  |
| --- |
| bool **fclose** ( resource $handle ) |

**예제파일) fclose.php**

|  |
| --- |
| <?php  $filename = './readme.txt';  if (file\_exists($filename)) {  if ($fp = fopen($filename, "r")) {  echo "파일 포인트를 읽기 전용으로 오픈합니다.<br>";  **fclose($fp);**  } else {  echo "파일을 읽을 수 없습니다.<br>";  }  } else {  echo "파일이 존재하지 않습니다.";  }    ?> |

**화면출력)**

파일 포인트를 읽기 전용으로 오픈합니다.

**06.6 파일 데이터읽기**

====================

PHP 코드를 통하여 파일을 읽어 처리해 보겠습니다. 이전에 학습한 내장함수 fopen()를 통하여 파일포인터를 오픈하였습니다. 이렇게 오픈된 파일포인터를 통하여 파일에 접근을 하여 내용을 읽어 올 수 있습니다. 접근된 파일에 데이터를 읽어올 수 있는 방법은 여러가지 형태가 있습니다. 데이터처리를 3가지 방법으로 구분하여 설명을 합니다.

**06.6.1 파일 끝**

====================

데이터를 가지고 있는 파일은 크기는 매우 다양합니다. 간단히 몇줄을 포함할 수도 있고, 몇 GB의 엄청난 파일도 있을 것입니다. 접근한 파일의 데이터를 읽기 위해서는 반복된 자료 읽기 작업이 수행이 됩니다.

자료읽기 작업을 반복 수행을 할 때 얼만큼의 반복을 수행하는 것을 정의해야 합니다. 쉬운 방법으로는 파일을 끝을 나타내는 기호를 확인하여 처리하는 것입니다.

파일들은 대부분 파일의 끝을 표시하는 EOF 라는 코드값(-1)을 가지고 있습니다. EOF는 End of File 의 약자 입니다. 파일을 읽을때 이 EOF 여부를 체크하여 파일의 끝을 확인할 수 있습니다.

PHP는 파일의 끝을 확인 할 수 있는 특별한 내장함수를 제공합니다.

**|내장함수|**

|  |
| --- |
| bool **feof** ( resource $handle ) |

foef()함수는 EOF 코드를 확인하여 논리값으로 출력을 합니다. feof() 함수를 사용할때는 반복문으로 while 구문을 사용합니다.

내부함수 fpassthru()는 EOF까지 열려있는 파일을 읽고, 결과를 출력 버퍼에 저장합니다.

**|내부함수|**

|  |
| --- |
| int **fpassthru** ( resource $handle ) |

**예제) fpassthru.php**

|  |
| --- |
| <?php  // open the file in a binary mode  $name = './img/ok.png';  $fp = fopen($name, 'rb');  // send the right headers  header("Content-Type: image/png");  header("Content-Length: " . filesize($name));  // dump the picture and stop the script  fpassthru($fp);  exit;  ?> |

**06.6.2 한글자 읽기**

====================

내장함수 fopen()함수로 얻은 파일포인터에서 실제적인 데이터를 읽어 보도록 하겠습니다. 데이터를 읽는 방법은 다양하지만, 우선 파일에서 한글자의 character 를 읽어 보도록 하겠습니다.

내장함수 fget()은 파일포인터에서 한글자의 문자를 읽어서 반환을 합니다. 한글자를 읽은 후에는 다음 글자를 읽을 수 있도록 파일포인터를 한단계식 이동을 합니다.

**|내장함수|**

|  |
| --- |
| string **fgetc** ( resource $handle ) |

한글자씩 읽는 fgetc() 함수는 파일을 처리하는데 매우 유용합니다. 파일의 내용을 분석하고 파싱처리를 하는데도 편리합니다. 하지만 파일 전체를 한글자씩 읽어야 하기 때문에 처리 시간이 오래 걸립니다.

**예제파일) fgetc.php**

|  |
| --- |
| <?php  $filename = './readme.txt';  $fp = fopen($filename, "r") or die("파일을 읽을 수 없습니다!");  // EOF 체크  while (!**feof($fp)**) {    $i++;  echo $i.": ".**fgetc($fp)**."<br>";  }  echo "파일 카운트 : $i";  fclose($fp);  ?> |

**화면출력)**

1: h

2: e

3: l

4: l

5: o

6:

7: j

8: i

9: n

10: y

11: P

12: H

13: P

14: 3

15:

파일 카운트 : 15

위의 실습은 파일에서 한글자씩 읽어서 화면에 출력을 하는 예제 입니다. 파일의 포인터를 얻어 foef()의 조건을 만족하는 많큼 while 반복문을 실행합니다. 이때 반복문으로 while()문을 이용하는 것은 파일의 크기를 미리 알 수 없기 때문입니다. 계속적으로 반복을 하면서 feof() 함수에서 EOF를 확인하고 true를 반환할때 while문은 종료됩니다.

**06.6.2 한줄읽기**

====================

내장함수 fgetc()는 파일 포인터에서 한글자씩 읽어 옵니다. 한글자씩 읽어오는 처리는 실행속도가 매우 느리게 됩니다. 이를 보완하고자 블록개념으로 파일의 데이터를 읽을 수 있는 fgets() 함수를 제공합니다.

내장함수 fgets()는 파일에서 한줄의 데이터를 읽어서 반환을 합니다. fgets() 함수는 환경설정 파일, 로그파일등을 읽고 처리하는데 좀더 유용합니다. 이러한 파일들은 정보를 한줄 단위로 구분하여 저장을 하기 때문에 한줄씩 읽는 fgets()는 보다 유용합니다.

**|내장함수|**

|  |
| --- |
| string **fgets** ( resource $handle [, int $length ] ) |

파일에서 한줄의 끝은 리턴문자(\n)를 포함합니다. gfets() 함수는 텍스트파일에서 리턴된 문장(\n)을 확인하여 데이터의 한줄을 읽어 옵니다. 또는 두번째 인자로 일정크기의 값을 지정한면, 지정된 데이터의 크기만 읽어 올 수도 있습니다.

**예제파일) fgets.php**

|  |
| --- |
| <?php  $filename = './readme2.txt';  // @ 표시가 앞에 있으면 오류메시지를 표시하지 않겠다는 의미 입니다.  $fp = @fopen($filename, "r");    if ($fp) {  while (($buffer = **fgets($fp, 4096)**) !== false) {  echo "라인 ".$i++.": ".$buffer."<br>";  }    if (!feof($fp)) {  echo "Err] fgets() 읽기 실패\n";  }    fclose($fp);  }  ?> |

**화면출력)**

라인 : 안녕하세요.

라인 1: 여러줄로 구성된 파일을

라인 2: 읽어 올 수 있습니다.

내장함수 fgetss()는 파일에서 HTML 및 PHP 태그가 제거 된 행을 반환합니다.

**|내장함수|**

|  |
| --- |
| string **fgetss** ( resource $handle [, int $length [, string $allowable\_tags ]] ) |

**예제) fgetss.php**

|  |
| --- |
| <?php  $str = <<<EOD  <html><body>  <p>Welcome! Today is the <?php echo(date('jS')); ?> of <?= date('F'); ?>.</p>  </body></html>  Text outside of the HTML block.  EOD;  file\_put\_contents('sample.php', $str);  $handle = @fopen("sample.php", "r");  if ($handle) {  while (!feof($handle)) {  $buffer = **fgetss($handle, 4096);**  echo $buffer;  }  fclose($handle);  }  ?> |

**화면출력)**

Welcome! Today is the of . Text outside of the HTML block.

**06.6.3 파일의 크기**

====================

파일을 읽고 처리하는데 있어서 EOF를 검출하여 처리를 하였습니다. PHP에서는 EOF를 확인하지 않고도 지정한 파일의 크기를 알수 있는 내부 함수를 제공합니다.

|내장함수|

|  |
| --- |
| int **filesize** ( string $filename ) |

내장함수 filesize() 는 파일의 크기를 쉽게 구할 수 있습니다. filesize() 함수는 파일의 크기를 정수값 으로 반환 합니다. 만일 파일이 존재하지 않으면 Warning 오류를 발생합니다.

**예제파일) filesize.php**

|  |
| --- |
| <?php  $filename = "./readme.txt";  if (file\_exists($filename)) {  echo $filename." 파일의 크기는".filesize($filename)." 입니다..<br>";  } else {  echo "파일을 찾을 수 없습니다.<br>";  }  $filename = "./readme2.txt";  if (file\_exists($filename)) {  echo $filename." 파일의 크기는".filesize($filename)." 입니다..<br>";  } else {  echo "파일을 찾을 수 없습니다.<br>";  }  ?> |

**화면출력)**

./readme.txt 파일의 크기는14 입니다..

./readme2.txt 파일의 크기는80 입니다..

**06.6.4 지정용량 읽기**

====================

fgetc(), fgets() 함수로 용량이 큰 파일을 글자단위로 읽는 것은 서버에와 스크립트 작업에 많은 부하를 줄 수 있습니다. 이런경우 효율적으로 데이터를 읽기 위해서 일정크기의 버퍼링을 통하여 파일을 읽을 수 있습니다.

한번에 버퍼 형태로 데이터를 읽어 오는 것이 빠른 성능 처리를 할 수 있습니다. fread() 함수는 지정한 바이트 크기 만큼의 데이터를 읽어와 데이터를 반환합니다.

**|내장함수|**

|  |
| --- |
| string **fread** ( resource $handle , int $length ) |

**예제파일) fread.php**

|  |
| --- |
| <?php  $filename = "./readme.txt";  if (file\_exists($filename)) {  if ($fp = fopen($filename, "r")) {  $contents = **fread($fp, filesize($filename))**;  echo $contents;  }  fclose($filename);  } else {  echo "파일을 찾을 수 없습니다.<br>";  }  ?> |

**화면출력)**

hello jinyPHP3

**06.6.5 포인터 조작**

====================

fgetc(), fgets() 등을 통하여 파일의 데이터를 읽는 경우에 함수를 반복적으로 호출하여 이용을 하였습니다. 이렇게 함수를 한번씩 호출될 때 마다 파일포인터는 자동으로 다음 위치로 이동을 합니다.

이렇게 이동된 파일 포인트의 위치를 확인하거나, 임의의 포인트로 이동을 할 수도 있습니다. 파일 포인트의 처리를 위한 몇 개의 내장함수를 제공합니다.

**|내장함수|**

|  |
| --- |
| int **ftell** ( resource $handle ) |

내장함수 ftell()는 현재의 파일 포인트의 위치를 확인할 수 있습니다.

**|내장함수|**

|  |
| --- |
| int **fseek** ( resource $handle , int $offset [, int $whence = SEEK\_SET ] ) |

내장함수 fseek()는 파일포인트를 이동할 수 있습니다.

**예제파일) file\_point.php**

|  |
| --- |
| <?php  $filename = './readme.txt';  if ($fp = fopen($filename, "r")){  $data = fgets($fp);  echo $data."<br>";  echo "파일포인트 = ".ftell($fp);  echo "<br>";  // 파일포인트 처음으로 이동  fseek($fp, 0);  $data = fgets($fp);  echo $data."<br>";  echo "파일포인트 = ".ftell($fp);  fclose($fp);  }  ?> |

**화면출력)**

hello jinyPHP3

파일포인트 = 14

hello jinyPHP3

파일포인트 = 14

**06.6.6 파일 콘텐츠 출력**

====================

PHP는 전형적인 복잡한 파일포인트를 사용하지 않고, 간단하게 파일의 내용을 출력할 수 있도록 특별한 내장함수를 지원합니다.

**|내장함수|**

|  |
| --- |
| int **readfile** ( string $filename [, bool $use\_include\_path = false [, resource $context ]] ) |

readfile() 함수는 정상적으로 파일을 읽었을 경우에 컨텐츠의 바이트 수를 반환합니다.

**예제파일) readfile php**

|  |
| --- |
| <?php  echo readfile("log.txt");  ?> |

**06.7 파일쓰기**

====================

fopen() 함수를 통하여 생성한 파일입출력 포인터는 파일을 읽기 와 같이 쓰기 작업도 같이 할 수 있습니다. 파일오픈시 모드를 w / a / x/ c 중의 하나로 설정을 하면, 생성된 포인트를 통하여 파일을 저장할 수 있습니다.

쓰기모드는 파일의 존재여부와 상관은 없습니다. 파일이 있으면 덮어 쓰거나, 추가로 더 연결하여 쓸 수도 있습니다. 또는 파일이 없을 경우에는 새로이 파일이 생성이 됩니다.

**예제파일) file\_write.php**

|  |
| --- |
| <?php  $filename = "./log.txt";  $fp = fopen($filename, "w");  if ($fp) {  echo "파일을 저장 할 수 있습니다.<br>";  } else {  echo "Err] 파일을 오픈할 수 없습니다.<br>";  }  ?> |

**화면출력)**

파일을 저장 할 수 있습니다.

**06.7.1 쓰기(W)**

===================

쓰기 모드로 오픈된 파일포인터에 지정한 크기의 데이터를 출력할 수 있습니다. 포인터로 데이터를 출력을 하는 것은 파일을 쓰기와 동일합니다. 하지만 파일 오픈시 지정한 모드에 따라서 약간씩 쓰기 모드는 다릅니다. 새롭게 파일을 생성하거나, 추가할 수 있습니다.

PHP는 파일에 텍스트/데이터를 삽입을 하기 위한 다음과 같은 내장함수를 제공합니다.

**|내장함수|**

|  |
| --- |
| int **fwrite** ( resource $handle , string $string [, int $length ] ) |

내장함수 fwrite() 함수는 파일포인트에 데이터를 저장하고, 파일 포인트를 다음 글자 저장을 위한 포인터의 위치도 함께 이동합니다.

**예제파일) fwrite.php**

|  |
| --- |
| <?php  $filename = "log.txt";  // W 모드는 새로운 파일을 생성합니다. 파일을 저장하는 위치는 처음 입니다.  $fp = fopen($filename, "w");  if ($fp) {  echo "파일 내용을 저장합니다.<br>";  $log = $\_SERVER['SERVER\_NAME'].";". $\_SERVER['REMOTE\_ADDR'].";"." \n";  $size = fwrite($fp, $log);  echo "저장: $size<br>";    fclose($fp);  } else {  echo "Err] 파일을 오픈할 수 없습니다.<br>";  }  ?> |

**화면출력)**

파일 내용을 저장합니다.

저장: 17

**06.7.2 쓰기추가(a)**

====================

w모드로 오픈된 파일 포인터는 새로운 내용으로 파일을 덥어쓰기 형태로 동작을 합니다. 만일 기존에 있는 파일에 추가로 내용을 쓰고 싶을 경우에는 a모드로 파일포인터를 오픈합니다.

a모드로 오픈된 파일포인터는 파일의 마지막 부분의 포인터를 반환합니다. 따라서 마지막 포인터를 기준으로 데이터를 추가로 삽입을 할 수 있습니다. 만일 지정한 파일이 파일이 없으면 새롭게 파일을 생성합니다. a모드는 항상 추가할 수 있도록 파일포인트는 맨 마지막을 가르킵니다.

**06.7.3 파일권한**

====================

리눅스와 같은 시스템의 경우 권한 설정을 통하여 파일 쓰기가 거부가 될 수도 있습니다. 다양한 운영체제와 권한체계를 확인후에 파일쓰기 작업을 하는 것을 추천합니다. 사전 확인하는 코드를 삽입하여 안정적인 동작처리를 수행할 수 있습니다.

**|내장함수|**

|  |
| --- |
| bool **is\_witeable**( 파일명 ) |

내장함수 is\_witeable() 는 현재의 파일과 디렉토리에서 파일 쓰기가 가능한지를 논리값으로 반환하여 줍니다.

파일 쓰기권한과 비슷하게 읽기 권한도 체크할 수 있는 내장함수를 제공합니다.

|내장함수|

|  |
| --- |
| bool **is\_readable**( 파일명 ) |

내장함수 is\_readable() 는 파일을 읽기 가능한지를 확인합니다.

**예제파일) file\_perms.php**

|  |
| --- |
| <?php  $filename = "log.txt";  // a 모드는 기존 파일에 새로운 추가 내용을 삽입합니다.  $fp = fopen($filename, "a");  if ($fp) {  if (**is\_writeable($filename)**) {  echo "파일 저장허용.<br>";  $log = $\_SERVER['SERVER\_NAME'].";". $\_SERVER['REMOTE\_ADDR'].";"." \n";  $size = fwrite($fp, $log);  echo "저장: $size<br>";    fclose($fp);  } else {  echo "파일 쓰기 권한이 없습니다.<br>";  }    } else {  echo "Err] 파일을 오픈할 수 없습니다.<br>";  }  ?> |

**화면출력)**

파일 저장허용.

저장: 16

**06.7.4 임시파일**

====================

작업 디렉토리에 파일을 생성하여 쓰기등의 출력 작업을 합니다. 이와 별개로 임시로 작업을 위해서 작업 디렉토리에 파일을 생성하는 불필요한 파일들을 생성할 수 있습니다. 이런경우, 임시 파일을 생성하여 처리를 할 수 있습니다.

내장함수 tmpfile()는 임시파일을 생성할 수 있습니다.

**|내장함수|**

|  |
| --- |
| resource **tmpfile** ( void ) |

예제파일) **tmpfile.php**

|  |
| --- |
| <?php  $temp = tmpfile();  fwrite($temp, "writing to tempfile");  fseek($temp, 0);  echo fread($temp, 1024);  fclose($temp); // this removes the file  ?> |

**|내장함수|**

|  |
| --- |
| string **tempnam** ( string $dir , string $prefix ) |

내장함수 tempnam()은 고유 한 임시 파일을 만듭니다.

예제파일) **tempnam.php**

|  |
| --- |
| <?php  $tmpfname = tempnam("/tmp", "FOO");  $handle = fopen($tmpfname, "w");  fwrite($handle, "writing to tempfile");  fclose($handle);  // do here something  unlink($tmpfname);  ?> |

**|내부함수|**

|  |
| --- |
| string **sys\_get\_temp\_dir** ( void ) |

내부함수 sys\_get\_temp\_dir ()는 임시 파일로 사용되는 디렉토리 경로를 반환합니다.

**예제) sys\_get\_temp\_dir.php**

|  |
| --- |
| <?php  $temp\_file = sys\_get\_temp\_dir();  echo $temp\_file;  ?> |

**화면출력)**

C:\Users\infoh\AppData\Local\Temp

**06.7.5 파일잠금 과 해제**

====================

파일은 여러사람들이 읽거나 쓸수가 있습니다. 만일 동시에 파일을 쓰거나, 읽을 경우에는 서로의 우선순위와 동시작업등을 인하여 액세스 충돌이 발생될 수 있습니다.

이런경우 먼저 파일의 엑세스등의 충돌을 방지하기 위하여 파일을 잠시 잠금을 설정할 수 있습니다.

**|내장함수|**

|  |
| --- |
| bool **flock** ( resource $handle , int $operation [, int &$wouldblock ] ) |

내장함수 flock() 은 해당 파일 포인트를 잠금을 처리합니다.

* LOCK\_SH : 공유잠금, 쓰기만 차단합니다.
* LOCK\_EX : 독점잠금, 읽기/쓰기 모두 차단합니다.
* LOCK\_UN : 잠금해제
* LOCK\_NB : 잠겨져 있을 경우 false 를 반환합니다.

flock()함수의 LOCK\_UN 옵션을 통하여 잠금을 해제할 수 있습니다. 하지만, 잠금을 해제하기 전에 fflush() 함수를 통하여 버퍼를 정리해준 후에 잠금을 해제해 주는 것을 권장합니다.

**|내장함수|**

|  |
| --- |
| bool **fflush** ( resource $handle ) |

**예제파일) file\_lock.php**

|  |
| --- |
| <?php  $filename = "log.txt";  if (is\_writeable($filename)) {  echo "파일 저장허용.<br>";  // a 모드는 기존 파일에 새로운 추가 내용을 삽입합니다.  $fp = fopen($filename, "a");  if (is\_resource($fp)) {  // 잠금 설정  flock($fp, LOCK\_EX);  $log = $\_SERVER['SERVER\_NAME'].";". $\_SERVER['REMOTE\_ADDR'].";"." \n";  $size = fwrite($fp, $log);  echo "저장: $size<br>";    // 버퍼 정리후에 잠금해제  fflush($fp);  flock($fp, LOCK\_UN);  fclose($fp);    } else {  echo "Err] 파일을 오픈할 수 없습니다.<br>";  }  } else {  echo "파일 쓰기 권한이 없습니다.<br>";  }  ?> |

**화면출력)**

파일 저장허용.

저장: 16

**05.8 파일삭제**

====================

생성된 파일 또는 기존의 파일을 PHP 코드를 통하여 삭제를 할 수 있습니다. 파일을 삭제하기 전에 반드시 백업 및 주의를 하여 삭제하는 습관이 필요합니다.

PHP에서는 파일 삭제와 관련된 다음과 같은 내부함수를 제공합니다.

**|내부함수|**

|  |
| --- |
| bool **unlink** ( string $filename [, resource $context ] ) |

내부함수 unlink() 는 서버의 파일을 삭제합니다. 파일을 삭제하기 위해서는 해당 디렉토리에 쓰기 권한이 있어야 합니다.

**예제파일) unlink.php**

|  |
| --- |
| <?php  $filename = "log.txt";  if (file\_exists($filename)) {  if (is\_writeable($filename)) {  unlink($filename);  echo $filename." 파일을 삭제합니다.<br>";  } else {  echo "삭제 권한이 없습니다.<br>";  }  } else {  echo "Err] 파일이 존재하지 않습니다.<br>";  }  ?> |

**화면출력)**

log.txt 파일을 삭제합니다.

**05.9 파일복사**

====================

기존 존재하는 파일을 새로운 파일이름으로 복사를 할 수 있습니다. 파일을 복사할때 파일명이 중복되면 덥어쓰기로 복사하게 됩니다.

PHP에서는 파일 복사와 관련된 다음과 같은 내부함수를 제공합니다.

**|내부함수|**

|  |
| --- |
| bool **copy** ( string $source , string $dest [, resource $context ] ) |

내부함수 copy() 는 파일을 복사후 성공여부를 논리값으로 반환을 합니다.

**예제파일) copy.php**

|  |
| --- |
| <?php  $filename = "readme.txt";  if (file\_exists($filename)) {  if (copy($filename, "readme\_copy.txt")) {  echo "복사 성공!<br>";  } else {  echo "복사 실패!<br>";  }  } else {  echo "Err] 파일이 존재하지 않습니다.<br>";  }  ?> |

**화면출력)**

복사 성공!

내부함수 is\_uploaded\_file ()는 HTTP POST를 통해 파일이 업로드되었는지 확인합니다.

**|내부함수|**

|  |
| --- |
| bool **is\_uploaded\_file** ( string $filename ) |

**예제) is\_uploaded\_file.php**

|  |
| --- |
| <?php  if (is\_uploaded\_file($\_FILES['userfile']['tmp\_name'])) {  echo "File ". $\_FILES['userfile']['name'] ." uploaded successfully.\n";  echo "Displaying contents\n";  readfile($\_FILES['userfile']['tmp\_name']);  } else {  echo "Possible file upload attack: ";  echo "filename '". $\_FILES['userfile']['tmp\_name'] . "'.";  }  ?> |

내장함수 move\_uploaded\_file()는 업로드 된 파일을 새 위치로 이동합니다.

|  |
| --- |
| bool **move\_uploaded\_file** ( string $filename , string $destination ) |

**예제) move\_uploaded\_file.php**

|  |
| --- |
| <?php  $uploads\_dir = '/uploads';  foreach ($\_FILES["pictures"]["error"] as $key => $error) {  if ($error == UPLOAD\_ERR\_OK) {  $tmp\_name = $\_FILES["pictures"]["tmp\_name"][$key];    $name = basename($\_FILES["pictures"]["name"][$key]);  move\_uploaded\_file($tmp\_name, "$uploads\_dir/$name");  }  }  ?> |

**05.10 file**

====================

PHP는 파일포인터를 통하여 입/출력을 하는것과 달리 보다 쉽게 파일을 출력할 수 있는 함수들을 몇 개 제공을 합니다.

**|내부함수| 한줄단위 출력**

|  |
| --- |
| array **file** ( string $filename [, int $flags = 0 [, resource $context ]] ) |

내부함수 file()는 fgets()함수처럼 파일의 한줄씩 데이터를 읽어서 배열로 반환합니다. 반환된 데이터는 반복문을 통하여 한줄씩 출력할 수 있습니다.

**|내부함수| 파일 전체입력**

|  |
| --- |
| string **file\_get\_contents** ( string $filename [, bool $use\_include\_path = false [, resource $context [, int $offset = 0 [, int $maxlen ]]]] ) |

내부함수 file\_get\_contents()는 readfile() 함수와 비슷하게 파일의 전체 컨덴츠 내용을 읽어옵니다.

**|내부함수| 파일 전체출력**

|  |
| --- |
| int **file\_put\_contents** ( string $filename , mixed $data [, int $flags = 0 [, resource $context ]] ) |

내부함수 file\_put\_contents()는 fwrite()와 같이 콘텐츠를 파일로 저장할 수 있는 기능을 제공합니다.

**예제파일) files.php**

|  |
| --- |
| <?php  $filename = "readme.txt";  if (file\_exists($filename)) {  // 파일 전체 출력  echo "=== 파일 전체 출력 ===<br>";  $body = file\_get\_contents($filename);  echo $body;  // 한줄씩 출력  echo "<br> === 한줄씩 출력 ===<br>";  $lines = file("readme2.txt");  foreach ($lines as $line\_num => $line) {  echo "Line #<b>{$line\_num}</b> : " . htmlspecialchars($line) . "<br />\n";  }  // 파일 저장  echo "<br> === 파일 저장 ===<br>";  file\_put\_contents("readme3.txt", $body);  } else {  echo "Err] 파일이 존재하지 않습니다.<br>";  }  ?> |

**화면출력)**

=== 파일 전체 출력 ===

hello jinyPHP3

=== 한줄씩 출력 ===

Line #**0** : 안녕하세요.

Line #**1** : 여러줄로 구성된 파일을

Line #**2** : 읽어 올 수 있습니다.

=== 파일 저장 ===

단, file 관련 함수를 통하면 fopen() 함수로 처리를 하던 잠금 설정은 할 수 없습니다. 하지만 쉽게 파일의 입출력을 할 수 있는 장점을 가지고 있습니다.

**05.11 그외 함수들**

====================

내부함수 popen()은 파이프 오픈을 처리합니다.

**|내부함수|**

|  |
| --- |
| resource **popen** ( string $command , string $mode ) |

**예제) popen.php**

|  |
| --- |
| <?php  $handle = popen("/bin/ls", "r");  ?> |

내부함수 popen() 는 오픈된 파이프를 닫습니다.

**|내부함수|**

|  |
| --- |
| int **pclose** ( resource $handle ) |

**예제) pclose.php**

|  |
| --- |
| <?php  $handle = popen('/bin/ls', 'r');  pclose($handle);  ?> |

내부함수 fscanf()는 형식에 따라 파일에서 입력을 구문을 분석합니다.

**|내부함수|**

|  |
| --- |
| mixed **fscanf** ( resource $handle , string $format [, mixed &$... ] ) |

**예제) fscanf.php**

|  |
| --- |
| <?php  $handle = fopen("users.txt", "r");  while ($userinfo = fscanf($handle, "%s\t%s\t%s\n")) {  list ($name, $profession, $countrycode) = $userinfo;  //... do something with the values  }  fclose($handle);  ?> |

내장함수 fstat()는 열린 파일에 대한 정보를 반환합니다.

**|내장함수|**

|  |
| --- |
| array **fstat** ( resource $handle ) |

**예제) fstat.php**

|  |
| --- |
| <?php  // open a file  $fp = fopen("/etc/passwd", "r");  // gather statistics  $fstat = fstat($fp);  // close the file  fclose($fp);  // print only the associative part  print\_r(array\_slice($fstat, 13));  ?> |

내장함수 ftruncate()는 열린 파일을 지정된 길이로 절단합니다.

**|내장함수|**

|  |
| --- |
| bool **ftruncate** ( resource $handle , int $size ) |

**예제) ftruncate.php**

|  |
| --- |
| <?php  $filename = 'lorem\_ipsum.txt';  $handle = fopen($filename, 'r+');  ftruncate($handle, rand(1, filesize($filename)));  rewind($handle);  echo fread($handle, filesize($filename));  fclose($handle);  ?> |

내장함수 parse\_ini\_file()는 구성 파일을 구문 분석합니다.

**|내장함수|**

|  |
| --- |
| array **parse\_ini\_file** ( string $filename [, bool $process\_sections = false [, int $scanner\_mode = INI\_SCANNER\_NORMAL ]] ) |

**예제) parse\_ini\_file.php**

|  |
| --- |
| <?php  define('BIRD', 'Dodo bird');  // Parse without sections  $ini\_array = parse\_ini\_file("sample.ini");  print\_r($ini\_array);  // Parse with sections  $ini\_array = parse\_ini\_file("sample.ini", true);  print\_r($ini\_array);  ?> |

내장함수 parse\_ini\_string()은 구성 문자열을 구문 분석합니다.

**|내장함수|**

|  |
| --- |
| array **parse\_ini\_string** ( string $ini [, bool $process\_sections = false [, int $scanner\_mode = INI\_SCANNER\_NORMAL ]] ) |