

9.0 Manipulação de arquivos e diretórios

Objetivos

Mostrar formas de manipulação de arquivos; Usar os principais comandos para trabalharmos com arquivo e diretórios; Aprender a trabalhar com leitura e escrita de arquivos, listagem, e criação de variáveis buffer de arquivos.

Assim como outras linguagens de programação, é muito importante trabalharmos com manipulações de arquivos e diretórios em PHP, onde temos a possibilidade de manipular um arquivo ou diretório dentro do servidor web, podendo criar arquivos responsáveis por guardar informações referentes aquele sistema ou página. Essas informações podem ser resgatadas futuramente, ou simplesmente são informações que ao invés de serem gravadas no bando de dados, foram gravadas em um arquivo ou log (arquivos que grava informações sobre o sistema, erros etc.).

Ao trabalhar com arquivos, no mínimo duas operações devem ser realizadas: abrir e fechar o arquivo.

9.1 Criando e Abrindo um Arquivo.

O comando utilizado para criar um arquivo é o mesmo que usamos para abri-lo, porém no Linux temos que dar permissões a pasta no qual o arquivo vai ser guardado.

Abra o console ou terminal do seu sistema(Linux). Digite:

```
chmod 777 fvarfwww
```

O comando chmod 777 dar todas as permissões possíveis na pasta www onde trabalharmos na criação de nossos arquivos.

Para abrir ou criar um arquivo utilizaremos o seguinte comando abaixo:

fopen

Com esse comando podemos abrir um arquivo e retornar um identificador. Sua sintaxe é a seguinte:

```
$identificador = fopen("string_do_arquivo", "modo_do_arquivo");
```

string_do_arquivo

< é definido como o nome do arquivo mais a sua extensão, isso

incluindo o caminho onde esse arquivo é localizado ou não, por exemplo:

“/home/aluno/meu_arquivo.txt”

Podemos observar um arquivo criado dentro da pasta alunos com o nome meu_arquivo.txt.

modo_do_arquivo

< *nesse parâmetro podemos determinar a forma que o arquivo vai* ser aberto com os seguintes valores:

“r”

< read, este modo abre o arquivo somente para leitura.

“w” < write, abre o arquivo somente para escrita, caso o arquivo não exista, tenta criá-lo. “a+” <

append, abre o arquivo para leitura e escrita, caso o arquivo não exista, tenta *

criá-lo.

Existem outros modos, mas trabalharemos somente com estes.



Dica!

Para trabalharmos com arquivos é sempre importante sabermos se a pasta ou o arquivo tem permissões dentro do Linux, caso isso não aconteça, o arquivo não será criado, lido ou até mesmo gravado.

Veja um exemplo do uso do comando fopen:

```
1 <?php
2
3 $a = fopen("meu_arquivo.txt", "w");
4
5 ?>
```

meu_arquivo.txt.

Caso o arquivo não exista, ele é criado dentro da pasta onde o arquivo .php foi criado, ou seja, no nosso exemplo o arquivo se chama index.php e está dentro da pasta www, após executarmos esse comando teremos um novo arquivo com o nome

9.2 Gravando em um arquivo.

Após o uso do comando fopen, temos um identificador apontando para o arquivo, e com ele que podemos fazer alterações ou manipulações. Podemos gravar dados dentro do arquivo com o uso do seguinte comando:

fwrite

sintaxe:

```
fwrite("identificador", "conteúdo");
```

identificador < é o parâmetro retornado pelo comando fopen.

conteúdo < é o conteúdo a ser gravado no arquivo.

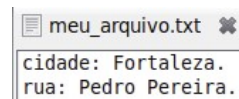
Vale ressaltar que para podermos gravar no arquivo ele deve ter permissão dentro do Linux e além disso ter como parâmetro “w” ou “a+” passado para o comando fopen.

Observe um exemplo onde escrevemos(gravamos) duas linhas dentro de um arquivo de texto criado com os comando visto até agora:

Exemplo:

```
1 <?php
2
3 $a = fopen("meu_arquivo.txt","w");
4 $texto = "cidade: Fortaleza.\nrua: Pedro Pereira.";
5 fwrite($a, $texto);
6
7 ?>
```

Resultado:

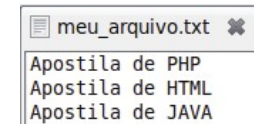


O uso de “\n” antes da palavra rua faz com que ocorra uma quebra de linha escrevendo o resto do conteúdo na linha abaixo. Após a execução do script (colocando <http://localhost> no navegador e o nome do script criado), abrimos o arquivo de texto(meu_arquivo.txt) com um editor e percebemos o resultado final.

Observe mais um *exemplo*:

Resultado:

```
1 <?php
2
3 $a = fopen("meu_arquivo.txt","w");
4 $texto1 = "Apostila de PHP.";
5 $texto2 = "Apostila de HTML.";
6 $texto3 = "Apostila de JAVA.";
7 fwrite($a, $texto1."\n");
8 fwrite($a, $texto2."\n");
9 fwrite($a, $texto3."\n");
10
11 ?>
```



No exemplo, fizemos a chamada do comando **fwrite** três vezes e escrevemos a cada chamada um valor diferente concatenando com “\n”.

9.3 Fechando um arquivo.

Até agora trabalhamos com o comando fopen e não fechamos o arquivo, simplesmente abrimos e executamos os demais comandos. Isso faz com que, caso tenhamos de usar o mesmo arquivo em outra parte do código, ele não poderá ser utilizado, pois para isso é preciso fechá-lo para ele poder ser aberto novamente em outra parte do código. Para isso usamos o seguinte comando:

fclose

```
fclose("identificador");
```

sintaxe:

exemplo:

```
1 <?php
2 $a = fopen("meu_arquivo.txt","w");
3 fwrite($a,"escreva seu conteúdo aqui!!");
4 // fecha o arquivo
5 fclose($a);
6 ?>
```

Toda vez que abrimos um arquivo com fopen, devemos fechá-lo com o comando fclose conforme o exemplo ao lado.

9.4 Lendo um arquivo.

Após abrimos um arquivo, outra operação que podemos efetuar é a leitura do conteúdo existente no arquivo. Essa operação é feita linha por linha, onde podemos resgatar valores existentes de acordo com a chamada do comando fread ou o índice do array criado pelo comando file.

file

Lê um arquivo e retorna um array com todo seu conteúdo, de modo que a cada posição do array representa uma linha do arquivo começando pelo índice 0.

```
$array = file("string_do_arquivo");
```

sintaxe:

string_do_arquivo < da mesma forma que é definida no comando fopen, usa-se o caminho com o nome do arquivo ou simplesmente o nome do arquivo caso ele exista na mesma pasta onde o arquivo PHP que contém o comando foi criado.

Exemplo:

```
1 <?php
2
3 $conteudo = file("meu_arquivo.txt");
4 echo $conteudo[0]."<br>";
5 echo $conteudo[1]."<br>";
6
7 ?>
```

Resultado:

Apostila de PHP
Apostila de HTML

Nesse exemplo utilizamos o arquivo anterior onde foi escrito três linhas, porém efetuamos a leitura somente da linha 1 (índice 0) e linha 2 (índice 1). Outra forma é percorrer o array usando um foreach(), dessa forma podemos ler todas as linhas existentes no arquivo, veja:

Exemplo:

```

1 <?php
2
3 $conteudo = file("meu_arquivo.txt");
4 foreach($conteudo as $valor)
5 echo $valor."<br>";
6
7 ?>

```

Resultado:

Apostila de PHP
Apostila de HTML
Apostila de JAVA

9.5 Copiando, Renomeando e Apagando um Arquivo

Em PHP também é possível copiarmos um arquivo de uma origem para um determinado destino, como também apagar esse arquivo. Para isso usamos os seguintes comando:

copy

Cria um arquivo para outro local/nome. retornando um valor booleano verdadeiro(true) caso a copia tenha ocorrido sem erros ou falhas, caso contrário retorna falso(false). Sintaxe:

```
copy("string_origem", "string_destino");
```

exemplo:

```

1 <?php
2 $origem = "meu_arquivo.txt";
3 $destino = "meu_novo_arquivo.txt";
4 $a = copy($origem, $destino); // copia o arquivo
5 if($a)
6     echo "Cópia efetuada";
7 else
8     echo "Erro ao copiar";
9 ?>

```

Caso tudo ocorra corretamente, o resultado apresentado no navegador é “Cópia efetuada”, e será criada uma cópia dentro da pasta com o nome “meu_novo_arquivo.txt”. Vale lembrar que podemos também passar o caminho completo para onde deve ser copiado, como por exemplo:

`fhomefalunofmeu_novo_arquivo.txt`

Para renomearmos um arquivo usamos:

rename Sintaxe:

```
rename("nome_do_arquivo", "novo_nome");
```

Para apagar um arquivo usamos:

unlink Sintaxe:

`unlink("nome_do_arquivo");`

Observe um exemplo, onde renomeamos o arquivo “meu_novo_arquivo.txt” para “arquivo_texto.txt” e apagamos o arquivo “meu_arquivo.txt”:

```

1 <?php
2 //renomeia o arquivo.
3 rename("meu_novo_arquivo.txt", "arquivo_texto.txt");
4 //apaga arquivo.
5 $a = unlink("meu_arquivo.txt");
6 if($a)
7     echo "arquivo apagado.";
8 else
9     echo "Erro ao apagar.";
10 ?>

```

Após executarmos isso no navegador, percebemos as mudanças ocorridas dentro do diretório.

9.6 Manipulando Diretório.

Alguns comandos básicos são necessários para manipulação de diretórios, mostraremos apenas como obter o diretório atual, como criar e apagar um diretório, para isso usamos os seguintes comandos:

mkdir

Cria um diretório de acordo com a localização e o modo. Sintaxe:

```
mkdir("string_localização", "int_modos");
```

`string_localização`
nome.

É definido como o caminho com o nome do diretório, ou somente o *int_modos* < é onde definimos as permissões de acesso (como se fosse o `chmod` do Linux). Dessa forma podemos criar um diretório e já atribuímos as permissões a ele.

getcwd

Retorna o diretório corrente, este comando é usado caso precise obter o diretório onde o arquivo PHP que possui este comando está guardado.

sintaxe:

```
getcwd();
```

rmdir

Apaga um diretório. Sintaxe:

```
rmkdir("nome_diretório");
```

Observe o exemplo envolvendo os três comandos abaixo:

Exemplo:

```
1 <?php
2
3 // cria um diretório.
4 $a = mkdir("minha_pasta",0777);
5 if($a)
6     echo "pasta criada.";
7 else
8     echo "erro!";
9 //retorna o diretório onde a pasta foi criada.
10 echo "<br>em ".getcwd()."<br>";
11 //apaga o diretório.
12 $a = rmdir("minha_pasta");
13 if($a)
14     echo "pasta apagada.";
15 else
16     echo "erro!";
17
18 ?>
```

Resultado:

pasta criada.
em /var/www
pasta apagada.

Observe que o comando `getcwd` obtém o caminho completo de onde o arquivo PHP que contém o código-fonte estar guardado.

9.7 Interações com o Browser

PHP também permite interagir com informações do browser automaticamente. Isso pode ser muito útil quando queremos coletar informações sobre o cliente, como por exemplo, o tipo de browser (navegador), ou qual o sistema operacional, dentre outras informações.

O código a seguir mostra informações sobre o browser do usuário:

```
2 <html>
3 <head>
4 <title>Apostila de PHP</title>
5 </head>
6 <body>
7 <?php echo $_SERVER['HTTP_USER_AGENT'] ?>
8 </body>
9 </html>
```

Comando `$_SERVER[HTTP_USER_AGENT]` tem como finalidade retornar informações do cliente que está acessando o arquivo PHP pelo browser, abaixo um exemplo desse retorno:

```
Mozilla/5.0 (X11; U; Linux i686; pt-BR; rv:1.9.2.10) Gecko/20100915 Ubuntu/10.04 (lucid) Firefox/3.6.10
```

Já vimos que o comando “HTTP_USER_AGENT” retorna uma string com várias informações, com isso podemos utilizar a função `strpos()`, que tem como finalidade procurar valores de uma string menor dentro de uma string maior. Podemos desta forma otimizar o nosso código. Sintaxe:

```
strpos( "string_maior" , "string_menor" );
```

Observe outro código com o uso da função `strpos()`:

```
2 <html>
3 <head>
4 <title>Apostila de PHP</title>
5 </head>
6 <body>
7 <?php
8 $info = $_SERVER['HTTP_USER_AGENT'] ;
9 if(strpos($info, "Firefox")!=0){
10     echo "Browser = Firefox";
11 }else if(strpos($info, "Chrome")!=0){
12     echo "Browser = Chrome";
13 }else{
14     echo "Desconhecido";
15 }
16 ?>
17 </body>
18 </html>
```

Nesse exemplo procuramos as palavras “Firefox” e “Chrome” dentro do valor retornado pelo comando `$_SERVER[“HTTP_USER_AGENT”]`. Dessa forma, podemos tratar o resultado comparando-o com “0”, ou seja, se a palavra existir, significa que seu valor é diferente de “0”. O resultado impresso na tela é de acordo com o navegador do cliente.

Exemplo de requisição HTTP_USER_AGENT:



9.10 Cookies

Cookies são mecanismos para armazenar e consultar informações. Eles são armazenados na máquina do cliente que acessa ao servidor php, e possui várias atribuições que são definidas pelo programador, por exemplo: imagine uma loja virtual, onde o cliente colocou em seu carrinho de compras vários produtos, mas por algum motivo ele não concluiu a compra, tendo que desligar a máquina que foi utilizada para fazer o acesso. No dia seguinte o cliente entra no mesmo site e percebe que todos os itens ainda estão no carrinho de compra do jeito que ele

deixou, esperando a conclusão da compra. Nesse exemplo, podemos perceber que as informações foram gravadas na máquina do cliente através dos cookies, que são simplesmente arquivos gerados pela página acessada dentro de alguma pasta do navegador que existe exclusivamente para esses arquivos.

O PHP atribui cookies utilizando a função **setcookie** que deve ser utilizada antes da tag

<html> numa página. Além disso o uso de cookies não é recomendado quando se trata de informações sigilosas. Os dados dos cookies são armazenados no diretório de arquivos temporários do visitante, sendo facilmente visualizado por pessoas mal intencionadas.

Além da opção “aceitar cookies” que pode ser desativada a qualquer momento pelo visitante. Mas em cada navegador essa opção pode mudar de nome. Observe o comando abaixo:

setcookie

Sua sintaxe possui muitos parâmetros, abaixo está representada todos os valores que podem ser atribuído ao setcookie, mas vale ressaltar que não utilizaremos todos eles, somente os principais, veja sua sintaxe.

```
Setcookie("nome_do_cookie","seu_valor","tempo_de_vida","path",  
"domínio","conexão_segura");
```

Onde na tabela abaixo temos a descrição de cada atributo:

Atributo	Descrição
nome_do_cookie	É o nome que, posteriormente, se tornará a variável e o que o servirá de referência para indicar o cookie.
seu_valor	É o valor que a variável possuirá. Esse valor pode ser de todos os tipos.
seu_valor	É o tempo, em segundos, que o cookie existirá no computador do visitante. Uma vez excedido esse prazo o cookie se apaga de modo irrecuperável. Se esse argumento ficar vazio, o cookie se apagará quando o visitante fechar o browser.
path	Endereço da página que gerou o cookie – automático
domínio	Domínio ao qual pertence o cookie – automático
conexão_segura	Indica se o cookie deverá ser transmitido somente em uma conexão segura HTTPS.

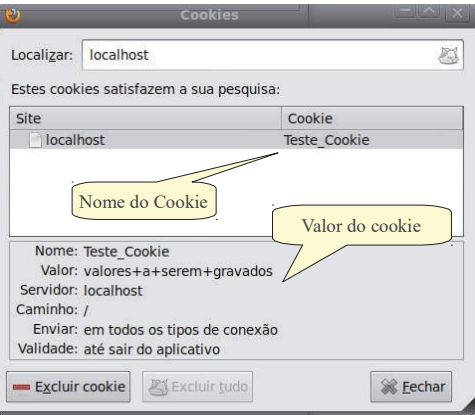
Observe um código onde criamos um cookie:

```
1 <?php  
2  
3 $valor = "valores a serem gravados";  
4 setcookie ( "Teste_Cookie" , $valor );  
5  
6 ?>
```

Criamos então uma string, logo após a função setcookie recebendo como parâmetro somente o

seu nome e o valor a ser gravado.

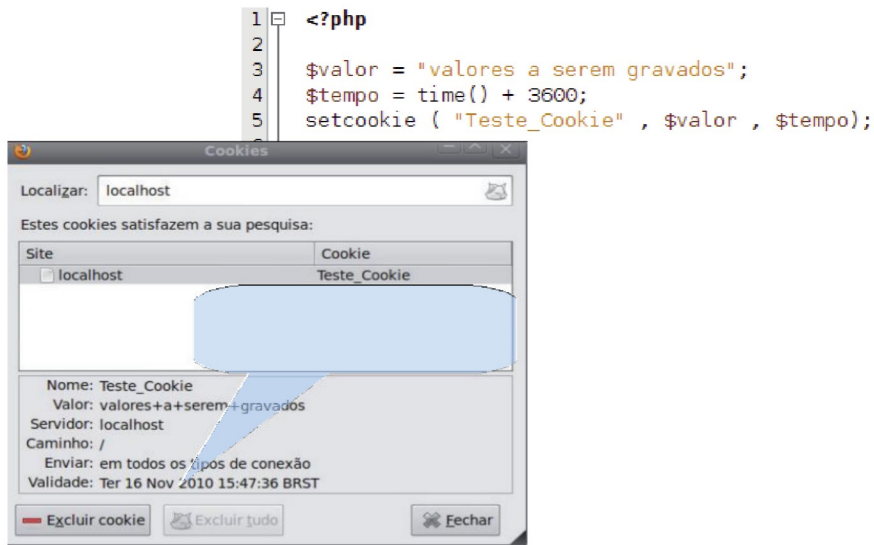
Usaremos o navegador Mozilla Firefox para visualizarmos o cookie criado, para isso basta digitar o endereço <http://localhost> na url, e logo após ir na opção: **Ferramentas < Propriedades da página < Segurança < Exibir cookie**. Lembre-se de criar o código acima primeiro e depois fazer a chamada pelo navegador de sua máquina. Se tudo ocorrer corretamente deverá aparecer a seguinte tela:



Veja que outras informações como caminho, enviar, e validade não foram especificados, porém podemos determiná-los na hora da criação do cookie dentro do código php.

Mostraremos agora um código onde atribuímos o tempo de vida do cookie, para isso devemos capturar o tempo com a função **time()** atual e somá-lo ao tempo que queremos em segundos, isso faz com que o cookie exista na máquina do cliente de acordo com a quantidade de tempo determinado pelo programador, observe um exemplo onde atribuímos mais esse parâmetro o função **setcookie**:

Exemplo:



O novo resultado é o seguinte:

Data de vencimento do cookie, após ela ele é deletado automaticamente

Esse cookie tem a validade de 3600 segundos, ou seja 1 hora, com isso concluímos que o navegador fez seu acesso as 14:47:36. Isso é muito importante para a programação dos cookies. Se quisermos que ele exista por um determinado tempo, temos que calcular tudo em segundos da seguinte forma:

```
$tempo = time() + (3600*24*7);
```

Esse cookie tem seu tempo de vida de 7 dias, pois 3600 segundos = 1 hora, 24 horas = 1 dia e 7 horas de um dia resulta em 7 dias.

Exemplo:

```

1 <?php
2 $valor = "valores a serem gravados";
3 $tempo = time() + (3600*24*7);
4 setcookie ( "Teste_Cookie" , $valor , $tempo);
5 ?>

```

Validade:

Validade: Sex 19 Nov 2010 18:55:29 BRST

Validade de 7 dias a partir do dia 12.

ACESSANDO UM COOKIE:

Para acessarmos o valor gravado em um cookie é bem simples, basta utilizar o comando

`$_COOKIE['coloque_aqui_o_nome_do_cookie']`, exemplo:

```

1 <?php
2 $valor = "valores a serem gravados";
3 $tempo = time() + (3600*24*7); // tempo de vida
4 setcookie ( "Teste_Cookie" , $valor , $tempo); // grava o valor
5 echo $_COOKIE['Teste_Cookie']; //imprime
6 ?>

```

Resultado: *valores a serem gravados*

Observe agora um exemplo de um código utilizado para contar as visitas de um site usando cookie:

```

1 <?php
2 $valor=1;
3 $tempo = time() + (3600*24*7); // tempo de vida
4 $valor = $_COOKIE['contador']+1; // inicia a contagem
5 setcookie ( "contador" , $valor , $tempo); // grava o valor
6 echo $_COOKIE['contador']." vistas no site!"; //imprime
7 ?>

```

O resultado é de acordo com a quantidade de vezes que o cliente entrou no site ou atualizou o mesmo.

9.11 Sessão

Sessões são mecanismos muito parecidos com os tradicionais cookies. Suas diferenças são que sessões são armazenadas no próprio servidor e não expiram a menos que o programador queira apagar a sessão.

As sessões são métodos de manter (ou preservar) determinados dados a mantê-los ativos enquanto o navegador do cliente (o internauta) estiver aberto, ou enquanto a sessão não expirar (por inatividade, ou porque em algum ponto você mandou que ela expirasse).

Para criarmos uma sessão utilizaremos a função abaixo:

```

1 <?php
2
3 session_start();
4
5 ?>

```

Dessa forma estamos iniciando um conjunto de regras. essa função deve sempre estar no início do código-fonte, com exceção de algumas regras.

Agora trabalharemos com essa sessão, primeiro podemos determinar o tempo de vida da sessão com o seguinte comando:

```

1 <?
2 session_cache_expire(5);
3 session_start();
4 ?>

```

Neste caso, `session_cache_expire` vem antes de `session_start`. Porque primeiro ele avisa que a sessão, quando iniciada, deve expirar em 5 minutos, e depois a inicia.

```

1 <?
2 session_start();
3 $var = "Sessão Criada!";
4 $_SESSION['minha_sessao'] = $var;
5 ?>

```

Com o comando `$_SESSION` podemos gravar valores na sessão, veja um exemplo:

Criamos uma sessão com o nome `minha_sessao` (não é uma boa prática de programação usar acentos em nomes de variáveis ou qualquer outra nomeação) e atribuímos a ela o valor gravado na variável string `$var`. Essas informações ficam gravadas no servidor, logo após podemos resgatar o valor da seguinte forma:

```

1 <?php
2 session_start();
3 echo $_SESSION['minha_sessao'];
4 ?>

```

Observe que o comando `$_SESSION` tem seu tratamento igual a uma variável do tipo Array. Para resgatar o valor da sessão, basta fazer a chamada do comando passando o nome da sessão, no caso “`minha_sessao`”. O exemplo anterior foi adicionado em um outro arquivo, por esse motivo temos que chamar novamente o comando `session_start()`, para trazermos ao arquivo todas as regras usadas em sessão no PHP.

Abaixo temos um exemplo com o uso da função **`isset()`**, que verifica se uma variável existe ou não, retornando um valor booleano (true ou false):

```

session_start();
if(isset($_SESSION['minha_sessao'])) {
    echo "Sessão Ativa";
} else {
    echo "Sessão não existe!";
}

```

O resultado é de acordo com a existência ou não da sessão.

Para desativarmos uma sessão podemos utilizar dois tipos de mecanismos: um deles é o uso da função **`session_destroy()`** que tem como finalidade destruir todas as sessões criada pelo usuário, a outra forma é desalocarmos a sessão criada com o uso da função **`unset()`**.

Uso de **`session_destroy()`**:

```

1 <?php
2 session_start();
3 session_destroy();
4 ?>
5

```

Uso de **`unset()`**:

Usamos `unset()` quando queremos desalocar uma determinada sessão, imaginamos que o usuário ao acessar uma determinada página, tenha criado várias sessões com nomes diferentes. Os nomes das sessões são determinadas pelo programador, porém ao clicar em um link, o mesmo tem que destruir a sessão escolhida. O exemplo abaixo destrói a sessão especificada:

```

1 <?php
2 session_start();
3 // Destroi a sessão especificada.
4 unset($_SESSION['minha_sessao']);
5 ?>

```

Dessa forma desalocamos (destruímos) a sessão “`minha_sessao`”, porém se existirem outras, elas ainda continuarão ativas.

9.12 Requisição de Arquivos

Assim como em muitas outras linguagens de programação também é possível incluir dentro de um script PHP outros arquivos contendo outras definições, constantes, configurações, ou até mesmo carregar um arquivo contendo a definição de uma classe. Para isso podemos usar os seguintes comandos:

`include<arquivo>`:

A instrução `include()` inclui e avalia o arquivo informado. O código existente no arquivo entra no escopo do programa que foi inserido, tornando-se disponível a partir da linha em que a inclusão ocorre. Se o arquivo não existir, produzirá uma mensagem de advertência (warning).

Exemplo onde temos dois arquivos:

código do arquivo teste.php

```

1 <?php
2
3 echo "<h1>Bem vindo ao Site</h1>";
4
5 ?>
6

```

código do arquivo index.php

```

1 <?php
2
3 include 'arquivo_teste.php';
4 echo "Bom dia";
5
6 ?>

```

Resultado:

Bem vindo ao Site

Bom dia

Nesse exemplo podemos notar que o código existente no “`arquivo_teste.php`” foi inserido dentro do arquivo `index.php`, tendo como resultado a execução dos dois códigos como se fossem apenas um, esse recurso é muito utilizado, pois podemos incluir até mesmo códigos de páginas inteiras em um arquivo.

`require<arquivo>`:

Comando muito parecido ao **include**. Difere somente na manipulação de erros. Enquanto o **include** produz uma warning, o **require** uma mensagem de Fatal Error caso o arquivo não exista.

Sintaxe: **require 'nome_do_arquivo.php';**

include once<arquivo>:

Tem funcionalidade semelhante ao **include**, a diferença é que caso o arquivo informado já esteja incluído, esse comando não refaz a operação, ou seja, o arquivo é incluído apenas uma vez. Este comando é útil para garantir que o arquivo foi carregado apenas uma vez. Caso o programa passe mais de uma vez pela mesma instrução, evitará sobreposição de arquivo.

Sintaxe: **include_once 'nome_do_arquivo.php'; require_once<arquivo>:**

Tem funcionalidade parecida com o comando **require**. A diferença é justamente caso o arquivo já tenha sido incluído no programa, pois ele não carrega novamente o código. É

muito semelhante ao **include_once**, evitando redeclarações ou sobreposições, porém a mensagem exibida caso o arquivo não exista é de Fatal Error.

Sintaxe: **require_once 'nome_do_arquivo.php';**

9.13 Tratamentos de erro

São muito comuns erros na programação PHP, que podem partir do programador como pessoa física, do servidor, ou outros fatores envolvidos que juntos venham ocasionar em um erro. Existem quatro tipos de erros no PHP para indicar a gravidade do erro encontrado ou ocorrido. Eles são:

1. Erros de funções (function errors).
2. Avisos (warnings).
3. Erros de processamento (parser error).
4. Observações (notice).

Os programadores devem prestar muita atenção nas mensagens de erro, afinal nenhum programador quer por no ar um sistema que quando o primeiro visitante entra apareça uma mensagem de erro. Para evitar essas inconveniências use sempre um “@” antes de cada chamada as funções. Se a opção **track_errors** no arquivo **php.ini** estiver habilitada, a mensagem de erro poderá ser encontrada na variável global **\$php_errormsg**. Para exibir a mensagem direta no navegador procure ativar a opção **display_errors**, as funções serão ativadas para **On** (ligada) ou **Off** (desligada).

O arquivo **php.ini** no linux geralmente fica na pasta: “/etc/php5/apache2/php.ini”. Veja o

Exemplo:

```
1 <?
2 strtolower();
3 ?>
```

Imprimindo a mensagem de erro:

```
1 <?
2 strtolower();
3 echo "<h1>".$php_errormsg."</h1>";
4 ?>
```

Resultado:

strtolower() expects exactly 1 parameter, 0 given

Mensagem de erro com a opção **display_errors = On** do **php.ini** ativada:

Warning: strtolower() expects exactly 1 parameter, 0 given in /var/www/index.php on line 2.

O erro mostra que devemos passar um parâmetro para a função **strtolower()** na linha 2, mas podemos ignorar o erro usando o “@”.

```
1 <?
2 @strtolower();
3 ?>
```

Essa função deixaria todos os caracteres em minúsculo, mas seu erro foi ignorado, não exibindo nenhuma mensagem de erro.

Lembrando: podemos utilizar tag's curtas no PHP “<? ?>” em vez de tags normais “<?php ?>”, mas isso vai depender da configuração do PHP (**php.ini**), em alguns casos não funciona.

Um vendedor de cestas básicas precisa de um sistema ao qual ele possa digitar quantas cestas básicas ele vendeu e qual o salário final que ele irá ter:

Um vendedor tem um salário fixo de R\$ 500,00 mais um bônus de 10% do valor de cada cesta básica vendida . Sabe-se também que cada cesta básica custa R\$30,00.

Crie código PHP que irá calcular e mostrar na página qual será o salário do mês.

OBS: Imagine que você já tem um formulário que têm um campo chamado : **qtdCestasV** que irá enviar essa informação via **POST** para uma “salarioVendedor.php”.

Crie um código PHP para solucionar o problema do vendedor.


```
<?php
// variável que irá receber a informação do usuário
$qtdCestasV = $_POST['qtdCestasV'];
//Variáveis que tratam de valores da cesta e comissão
$valorCesta = 30;
$comissao = 0.1;

//Calculo matemático para obter salário
$salario = 500 + ($qtdCestasV*$valorCesta)*$comissao;

//Imprimindo salário.
echo "Salário Final = R$ ".$salario;
?>
```

EXERCÍCIOS PROPOSTOS

O que é manipulação de arquivos? **EP09.1:** Observe o código-fonte abaixo:

```
1 <?php
2 $arquivo = fopen("all.txt","w");
3 fwrite($arquivo, "oi tudo bem!");
4 fclose($arquivo); 5 ?>
```

- Que tipo de arquivo é gerado na linha 2 e aonde o mesmo é criado?
- Que o parâmetro “w” da linha 2 do código representa?
- Qual a finalidade do comando *fwrite* da linha 3 do código?
- Qual o principal motivo de fecharmos o arquivo com o comando *fclose* da linha 4 do código? *Crie um arquivo de texto chamado “frases.txt” usando o comando fopen, e responda as questões 3,4,5,6,7*

Grave uma mensagem dentro do arquivo criado.

Com base no arquivo criado, utilize o comando *fwrite* para ler o mesmo imprimindo na tela do navegador o conteúdo do arquivo.

Abra o arquivo “frases.txt” com um editor de texto, adicione cinco palavras, cada uma em uma linha diferente, após isso utilize o comando *file*, para efetuar a leitura do arquivo, e imprima na tela a primeira e última palavras com o comando *echo*.

Crie uma cópia do arquivo renomeando o novo arquivo para “palavras.txt”.

Crie um diretório com o comando *mkdir* e copie o arquivo “palavras.txt” para a pasta criada e apague o anterior, tudo com comandos PHP.

Crie um código que imprima na tela todo o caminho de pastas onde se localiza o arquivo “palavras.txt”.

Crie um formulário HTML com os seguintes campos: “nome”, “endereço”, “e-mail”, “senha”, e o botão “enviar”.

Utilize o método Get para visualizar os dados do array na URL do navegador ao clicar no botão enviar.

Qual a diferença do método POST e GET?

Como podemos receber dados via método GET? Exemplifique.

Como podemos receber dados via método POST? Exemplifique.

Crie um arquivo chamado dados.php, nesse arquivo deverá conter os seguintes requisitos:

-Os dados do formulário da questão 1 deverá ser enviado para esse arquivo via método POST.

-O arquivo dados.php deverá conter cinco variáveis, cada uma para determinado campo, exemplo:

```
$nome = $_POST['nome'];
```

-Os valores deverão ser impresso na tela.

Qual a finalidade do comando \$_SERVER [“HTTP_USER_AGENTE”]?

Crie um cookie gravando nele o seu nome, logo após abra o Firefox e Exiba o valor gravado.

Crie um arquivo chamado “criar_sessao.php”, utilize comando PHP para criar uma sessão com a durabilidade de 3 minutos e adicione o valor “sessão ativa”.

Crie um novo arquivo chamado “ler_sessao.php” para verificar se a sessão criada na questão 9 existe ou não.

Utilize o comando \$_SESSION para ler o valor nela contido e imprima na tela.

Quais os possíveis erros existentes em PHP e qual a definição de cada um deles?