A linguagem de programação PHP é extremamente popular. É simples. O CMS WordPress, por exemplo, é construído em PHP. Um CMS é um sistema gerenciador de conteúdo. O WordPress foi criado inicialmente para blogs. Como podia também ser usado para sites, seu espectro foi expandido.
Com as novas versões do PHP (ex 5.4) foi introduzido um simples servidor web com ele, invocado pelo comando abaixo por exemplo:
php -S 127.0.0.1:80

Um servidor web recebe requisições na porta indicada (normalmente a 80) e responde com o conteúdo desejado. Um famoso servidor web é o Apache. Sempre que temos uma linguagem de programação web, como o PHP, o Ruby ou o Python, por exemplo, temos um servidor web recebendo requisições e encaminhado quando necessário por exemplo para o interpretador da linguagem. Podemos fazer uma página HTML “simples” com o PHP por exemplo, mas é um desperdício, já que o Apache poderia responder diretamente à requisição.

Com o tempo, foram criados conceitos e softwares ligados a tais conceitos. Um deles é o MVC (model, view, controller). O PHP, ao contrário de linguagens como o Ruby e o Python, não tem um framework MVC padrão de fato, como respectivamente o Rails e o Django. Não que não existam outros, mas estes são os mais populares. No PHP temos diversos frameworks “disputando a ponta”. Temos exemplo o Laravel, o Cakephp, o Codeigniter, o Lithium e o Symfony. Os 2 primeiros são inspirados no Rails do Ruby. Existem tantos frameworks em PHP que a Zend, criadora da Zend Engine, usada no PHP, criou o Zend Framework, com chances de virar um padrão de fato para PHP, o que não ocorreu porque o framework é muito complicado no meu ponto de vista. O Rails usa um conceito chamado “Convention Over Configuration”. Ele diz que temos que ter convenções e não configurações desnecessárias para tudo. Isto simplifica o framework. Por exemplo, os diretórios Controller, View e Modelos são em letras minúsculas. Não é necessário configurar isto. É a única possibilidade. É assim e pronto. Simples!

O uso de Rails na comunidade Ruby “é tão padrão” que “é comum” fazer coisas erradas. Por exemplo não usar migrations. É um framework completíssimo. Já em PHP…

O WordPress por exemplo foge bastante dos padrões PHP. Para evitar a bagunça, foram criadas PSRs. Elas contém instruções para tentar se evitar fazer besteira e padronizam diferentes frameworks. Mas o WordPress é tão antigo e tão fora do padrão que preferiu-se usar o Codex deles. Lá há as “boas práticas” do WordPress.

Como bastante do que se pesa é o resultado prático, o WordPress continua bem, sendo usado por exemplo no Washington Post e no Gazeta Esportiva.

Com o tempo surgiu a necessidade de melhora no desempenho de aplicações web. Não que fosse péssimo. Mas esta coisa de se postar para o servidor e voltar uma página idêntica por exemplo poderia ser otimizada.

Então começaram a se usar coisas como postar em JavaScript e se interpretar o retorno e websockets. Websockets são uma tecnologia que faz com que se tenha um canal aberto constantemente com o servidor web, sendo usado por tecnologias como o Socket.io do Node.js (outra linguagem a se rodar no servidor). Através dela e de JavaScript que roda no navegador, pode-se atualizar apenas uma parte da página, sem recarregar ela toda. É uma tecnologia muito boa.

Além disso, com a evolução da Web e o surgimento de novas necessidades, frameworks como o Angular.js e o Ember foram criados. Além disso o famoso Jquery também evoluiu. Enquanto os primeiros se propõem a serem tudo o que o programador precisa para fazer uma página, o Jquery é uma excelente ferramenta para facilitar e padronizar as coisas. Existem padrões web para os navegadores, como o Chrome, o Firefox e o Internet Explorer, mas nem sempre tais padrões são seguidos. O Internet Explorer por exemplo nem sempre segue os padrões, fazendo por exemplo com que páginas não funcionem em tais navegadores. Esta iniciativa parece ser abrupta, mas faz com que os usuários que precisem de tais páginas atualizem seus navegadores, facilitando a vida dos desenvolvedores.

O Jquery, por exemplo, padroniza a postagem de informações em diferentes navegadores através do jQuery.post(). Sem ele é necessário ver se estão disponíveis no navegador algumas coisas, tornando o código mais complexo.

O Angular.js por exemplo tem o $http.post(). Com ele é possível postar um formulário para um servidor.

Na web usamos o protocolo HTTP para trafegar informações. Com ele vem o conceito de verbos. São por exemplo o GET, o POST, o PUT e o DELETE. O GET deve ser usado para se obter informações, já o POST para se enviar um conjunto completo de informações. O PUT para se enviar um conjunto parcial (ex ao se atualizar um registro pode-se corrigir apenas o nome) e o delete para se excluir. Isto deveria ser o padrão. Em PHP conseguimos facilmente dados enviados pelos verbos GET e POST, mas não pelos outros verbos. Você indica um verbo por exemplo pela tag <form> de um formulário. Em Ruby usando o framework Rails (ou Ruby on Rails ou RoR) você facilmente faz um PUT.

Apesar do padrão acima existir, tudo funciona sem ele. Ele existe para organizar as coisas e não para funcionar apenas da forma dele. Você pode por exemplo fazer um POST e não atualizar nada mesmo sem eventuais erros em um formulário.

A cUrl é uma ferramenta de linha de comando que facilita as requisições web. É tão boa que, apesar de existirem outras formas de se fazerem requisições web com PHP, existe um módulo cUrl para Apache.

Então devo escolher uma linguagem de servidor, outra de front-end, como um framework JavaScript para programar web? Então, os puritanos falam que não. Que você deveria escolher a ferramenta conforme a sua necessidade. Mas em alguns casos há diferentes ferramentas para a mesma coisa. Sugiro então que você escolha a melhor ferramenta dependendo do caso (se o que fizer tiver várias ferramentas para tal). Sendo parcial, sugiro

que escolha o RoR com Angular. Fui parcial, mas bem direto :). Para você ter uma ideia, o RoR é tão bom que no trabalho o usamos para APIs que o Sinatra daria conta tranquilamente. A não ser em casos particulares, como WordPress e blogs, sugiro que escolha a ferramenta que tem experiência. Isto fará o desenvolvimento mais simples. Não subestime a complexidade de uma ferramenta (sugestão). No início é comum fazermos sistemas que funcionam, mas com algumas besteiras por não conhecer bem a ferramenta (eu já fiz por exemplo com Cake PHP).

A versão atual do PHP é a 7. Ela é muito mais rápida que a 5.6, presente em hosts como a Locaweb (nada contra, https://ajuda.locaweb.com.br/wiki/alterar-versao-do-php-hospedagem-de-sites/, https://rberaldo.com.br/php-7-9-vezes-mais-rapido-que-php-5-6/). Fora a questão de performance, existem as diferenças como o acréscimo do type hint de escalares e o "operador de coalescência nula" (https://secure.php.net/manual/pt_BR/migration70.new-features.php). Mas sugiro cuidado caso esteja fazendo um upgrade: https://chriseggleston.com/php7-upgrade-wpengine-hositng/ . Não é impossível, apenas aconselho ter cuidado. Não é tão simples.

Rails tem seu servidor web próprio, não indicado para sistemas para usuários finais (ou de produção). Ele é o Webrick. Facilmente pode-se trocar tal servidor, usando por exemplo o Puma.

Sugiro: blogs com WordPress. Sites simples com WordPress. Sistemas genéricos em RoR. Sistemas genéricos que tenham que ser em PHP com CakePHP. Python fica com quase nada, como ferramentas de linha de comando que não sejam construídas com PHP. Angular no front end.

As versões de Angular são engraçadas. Existe o Angular.js, ou Angular 1, o Angular 2 e o Angular 4. Cadê o 3? Um programador fez uma tag errada e com isto decidiu-se usar Angular 4 para tentar amenizar.

Tags e branches são bem simples no Git. Este é um versionador indicadíssimo para comunidade web. O branch principal é qualquer um, mas convenciona-se usar o "master". Tags não estão relacionadas diretamente a código, mas são marcações para se lembrar facilmente de coisas específicas, como a versão 1.0.0 do sistema. Por exemplo:
git remote add origin git@bitbucket.org/apterceiro/sistema.git
diz onde é atualizado o sistema remotamente

atualização de um arquivo
git add arquivo.txt

supondo o arquivo.txt o atualizado no passo anterior. A mensagem de commit é "teste"
git commit -m "teste"

Enviando para o host remoto:

git push origin master

git tag v1.0.0
criada a TAG v1.0.0

git push origin --tags
enviada a tag para o servidor

Caso queira usar um commit ou uma TAG específica, pode-se fazer um checkout. Por exemplo:
git checkout v1.0.0

O git log te mostra o histórico local de commits. Ex:
git log

Caso queira ou precise criar um branch, use por exemplo:
git checkout -b nome_branch

Sugere-se ver se as alterações que fez são as que deveria. Para isto usa-se o git diff. Ele mostra a diferença entre o que está em staging (colocado lá com o git add) e o que não está. Ou seja, o que será “commitado”. A não ser que você seja maluco como meu chefe, é interessante você usar um diff viewer, que facilitará ver as diferenças. Eu uso e aconselho por exemplo o Meld Merge. Se não usar, verá (se conseguir) em uma ferramenta de linha de comando, que indicará as alterações através de + e -.

Opções de hosts remotos para o Git são por exemplo o BitBucket, o Github e o Gitlab.

O Git pode ser usado em fluxos específicos. Usamos onde trabalho o que determina o https://nvie.com/posts/a-successful-git-branching-model/
Para nós funcionou muito bem.

Antes do Git (não que sejam totalmente obsoletas) usava-se ferramentas como o CVS e o SVN. Este último tem conceitos como branches, trunk (o branch principal) e as tags. As alterações ou deltas são computadas de forma diferente do Git para versionamento.

Na web temos umas coisas meio geniais e malucas. Por exemplo o Git foi criado pelo Linus Torvalds, o gênio atrás do Linux, em 1 semana, pois ele estava insatisfeito com o SVN. Não gostei do SVN. Então vou criar em 1 semana o versionador mais usado no mundo… O JavaScript foi rapidamente criado pelo Brendan Eich. Não é Java. Mas comercialmente seria interessante usar tal nome. Só o fantástico Google foi uma tese de doutorado. Brincadeira.

O Git hoje conta com ferramentas web sensacionais, como o BitBucket e o Github. A escolha normalmente está relacionada à limitação da ferramenta (ou o contratante é maluco…). O BitBucket por exemplo, usado onde trabalho, não tem problema com a

limitação de usuários. Já o Github não tem problema, na versão paga, com o número de projetos. Ambos têm versões gratuitas, sendo o Github o mais usado neste caso.

Temos também o não tão usado “mercurial” ou “hg”. Ele está facilmente disponível por exemplo no BitBucket. Eu usei por um bom tempo o SVN. Como no trabalho atual o padrão era o Git, então este foi o usado. Lá não usamos por exemplo o framework PHP que mais gosto, o Cakephp, mas esta é uma decisão, normalmente muito certa adotada pela minha chefia. Lá aprendi Ruby e Python (pelo Coursera, mas na época em que eu estava - e estou - lá).

O SVN e o Git (talvez outros também) tem o conceito de hooks. Este conceito não é básico, mas é legal. Você pode interceptar um commit e analisá-lo. Uma ferramenta que usei para isto foi o PHPCodesniffer no SVN. Tal ferramenta permite avaliar se um código está dentro de um padrão, negando um commit se não estiver. Na verdade esta ferramenta não está limitada a hooks e sim este é um uso dela. É uma ferramenta que considero excepcional.

PHP tem diversas ferramentas de qualidade (QA), como o PHPCopyPast detector, que localiza cópias e colagens de conteúdo, que podiam idealmente serem feitos de forma diferente, o PHPMD, que contém diversas ferramentas para se melhorar o código, o PHPCodesniffer, já citado, o php-cs-fixer, que ajusta o código a um padrão pré estabelecido (diferente do PHPCodesniffer, em que facilmente se define o padrão, mas não se corrige automaticamente o código), o php -l, uma ferramenta de linha de comando que permite se fazer um lint no código e o PHPDepend, uma excelente ferramenta que permite extrair métricas do seu código PHP.

Aconselho a leitura do PHP QA Tools para mais detalhes. Se o site for simples talvez nem linguagem de programação seja necessária e sim um bom designer. Ferramentas de QA são indicadas em projetos não triviais.

Há também o Composer. É uma ferramenta muito legal que permite atualizar bibliotecas em seu sistema. Uma alternativa não muito usada e antiga é a Pear. A diferença para nós, usuários finais, é que a Pear atualiza a máquina como um todo e o Composer é por projeto. Sugiro aprender Composer.

Continuando com projetos não triviais, uma coisa importante são testes. Cada linguagem tem seu “padrão de fato”. Certamente, para a mesma linguagem temos mais de um padrão para coisas diferentes. Um exemplo é Ruby, onde temos o Rspec e o Cucumber. Ambos não concorrem e inclusive é possível usar um com o outro. Mas a mais usada de Ruby é o Rspec e a de PHP é o PHPUnit. No mundo JavaScript estamos em uma época de consolidação. Temos por exemplo o Jasmine e o Mocha, excelentes ferramentas de teste. Além disso, temos excelentes ferramentas auxiliares, como o Sinon.js, que aproveita características da linguagem para fazer uma excelente ferramenta.

Você pode por exemplo combinar uma hook seu versionador com a execução de testes (nunca usei desta forma, mas com sucesso com o PHPCodesniffer). Onde trabalho a

equipe em geral é “sênior” e não esquece geralmente de executar os testes. Um erro nestes pode indicar ou um erro nos testes ou pior: um erro no software.

Tarefas corriqueiras, como as de um blog WordPress, não tem testes. Se um erro ocorresse em tal, os desenvolvedores rapidamente seriam avisados e corrigiriam. Na verdade não sei se tais testes existem, pois a comunidade WordPress tem tentado “não ficar para atrás”. Mas minha afirmação anterior é verdade. Além disso, é muito difícil programadores experientes errarem no básico.

Colocar o código disponível para usuários (ou em produção) depende da forma como o servidor de produção permite. O básico é se ter um serviço FTP no servidor. Através dele, do endereço e de um usuário e senha (sim, não é tão fácil) pode-se ver o sistema de arquivos do servidor e se atualizar lá. Uma ferramenta indicada neste caso é o Filezilla para Windows (tem uma boa visualização de arquivos, além de permitir guardar um histórico de usuários). Outra opção é se ter um Git no servidor e se atualizar o branch master, ou melhor, conforme uma determinada tag. Na WPEngine, um host WordPress estrangeiro, existe no painel do WordPress do usuário principal um staging->production, que permite atualizar o código de produção conforme staging. Outra forma (tá tirando a criatividade da galera?) não sei ou esqueci.

Então o fluxo das alterações é:

alterar -> enviar para o versionador -> enviar para o host remoto (servidor)

Onde trabalho temos um passo opcional adicional que pode substituir o PHPCodesniffer (não com a mesma qualidade…) que é o uso de Pull Request. Através deste você desenvolve o seu conteúdo em um branch e faz um merge com o branch de destino. Mas para isto você tem que passar pela avaliação de colegas. Usamos isto onde trabalho. Pull requests são criados de forma simples no BitBucket e permitem qualquer tipo de avaliação dos colegas (não segue tal padrão, isto poderia ser feito de melhor forma assim, isto dará para nós problemas futuros porque…). Em geral o PHPCodesniffer é muito preciso, mas indicado para um padrão de codificação. Pode até ter seu uso extendido, mas aí aumenta a sua complexidade. Pull requests são mais simples (aee, a chefia vai me amar… brincadeira. São bem mais simples mesmo).

Blz, então em resumo o que tenho que estudar?
O básicão é no mínimo saber o que se quer fazer. Se é layout, você deverá manjar de desenho, Photoshop, Ilustrator, jogar este guia fora (brincadeira). Se você gosta de programação, deverá saber se quer o front (o que o usuários vê) ou o backend (o que o usuário não vê). Para front end temos que conhecer Javescript, CSS e HTML (básico). O Sass é aconselhável. Como te disse, o front-end está evoluindo e dependendo da vaga você pode ter que saber também Angular ou Ember (menos popular). Ferramentas também tem sido criadas, como o Gulp. Para não ser um backend gambiarreiro, sobrinho como dizem, é interessante saber no mínimo uma linguagem de programação e testes. Tanto para back quanto para front, obviamente deve se conhecer um bom editor de código, como o Atom, o Sublime Text (sem piratear, por favor) ou o VSCode. Além disso, em ambos os

casos deve-se ter o simples aprendizado de um versionador. É muito a parte do versionador. Quando for possível e permitido, é possível usar ferramentas gráficas como o TortoiseSVN (muito boa).

E como aprender uma linguagem?
Se estiver começando, sugiro (grande merda) o Codecademy. Codeschool também é legal. Há cursos gratuitos em ambos. Mais para frente sugiro o Egghead.io ou o Pluralsight. Não seja cabeção: não esqueça a parte dos testes. Ou você é um fenômeno que nunca erra e precisará provar isto ou precisará dos testes. Eles não são essenciais, mas te ajudarão.

E frameworks? Cara, all they suck. Brincadeira, queria apenas ter meu momento Rasmus. Adoro eles. Se for Ruby o básico é saber Rails e Rspec. Se for Python, Django. Se for PHP, depende. Onde trabalho temos sistemas com Zend 1, Zend 2 (ô praga), Laravel e Symphony. Destes eu preferia os 2 últimos, apesar que sou core commiter em um sistema que adoro com Zend 1. Não temos nada com CakePHP, mas usei em meu trabalho anterior e gostei. Saber PHPUnit também é muito importante, a não ser que queira ser um gambiarreiro.

Outra ferramenta interessante para testes web é o Selenium. Ele, através da ferramenta que está construindo pronta, consegue dar cliques na interface por exemplo e preencher formulários. Você normalmente opera por uma boa interface. Adivinha onde temos? Naquele sistema com Zend 1 que falei…

Em geral no trabalho temos o seguinte: APIs em Rails, sistemas quaisquer: dá para usar WordPress? Alegria da chefia. Então usa, praga. Se não, dá para usar Rails? Então já sabe… Se não, propostas em equipe. Se convencer a galera, vai do seu jeito. Eu particularmente não gosto de alterar coisas antigas funcionando. Exemplo o sistema em Zend 1. Já recebi propostas para reescrever em Rails. E o que disse? Vai dar merda. Vou errar em coisas que estão funcionando. Aí a 1a coisa que vão me perguntar é: porque você está fazendo isto? Estava funcionando. E eu vou ficar quieto…

No front end você precisa saber ao menos o básico de JavaScript. Jquery é muito bom também, mas nem sempre necessário. CSS é o básico e HTML também (até para nós do backend, o HTML). Power-ups são os frameworks como o Angular e o Ember e as ferramentas, como o Jasmine, o Mocha, o Simon e o Gulp.

E versionador? Cara, é bem simples. Te sugiro o Git, mas depende de onde trabalhar. Eu usava Windows e TortoiseSVN. Não tinha problemas. “Tinha” que usar Windows por causa do AS3 (tá bom, não tinha…). Agora uso um adorado Mac Mini. Todos de desenvolvimento, exceto eu, o bestão, usam Linux onde trabalho. Eu, neste caso, gosto do Xubuntu que uso em casa. SVN e CVS tendem cada vez a serem menos usados. Isto porque o Linus é lindo. Brincadeira. Creio que seja pelo sucesso do Github. Apenas creio. Não sei como o Mercurial evoluirá. Não subestimo a ferramenta, mas não ouvi falar nem vi coisas que ela faz que as outras não fazem. Se o Linus tivesse caganeira naquela semana… Brincadeira.

E editor de código? Cara para o básicão só eu uso o Geany. Não é muito popular. Uso o Sublime Text para outras coisas (a maioria). Como não sou hipster (brincadeira), não aprendi outra. O Sublime Text 2 faz o que eu preciso. A maioria é codificar. Não necessariamente é o padrão. Bastante gente associa ferramentas ao editor de código. Vi bastante gente usando o VSCode. Sugiro. Tem gente usando também o Atom e o VI. Boas ferramentas.

E deploy? Cara, cada um com seus problemas… Brincadeira. O básico é saber usar um cliente FTP. Mas é muito simples. A não ser que você goste de sofrer e faça na linha de comando… O Filezilla é muito simples… Já se você fizer deploy com Git, use os conceitos que aprendeu com a ferramenta. Terá que basicamente fazer o pull de uma tag… Simples.

E o tal do Node.js?
Bom, tiveram a brilhante ideia (brincadeira, já usei Node) de usar no backend a mesma linguagem do frontend. Aí temos JavaScript no backend. Mas não muda só isso… Não é como uma mudança de PHP para Ruby… JavaScript por exemplo tem um conceito de herança prototipal que não existe em outras linguagens. E os callbacks? E a emoção? Brincadeira. Mas como é bom saber para o frontend, então boas… Dá pra usar Flash? 10 anos atrás dava, mas hoje não… Nem tem suporte no Mac… Ia mais te trazer problemas…

Quando usei, a ferramenta mais conhecida para websockets era o Socket.io, para (surpresa) Node. Hoje temos por exemplo para Rails e para PHP (http://socketo.me/). Mas na época usei o que tínhamos…

Links recomendados (influenciados pelo autor):

Boas práticas:

- phptherightway.com
- jstherightway.com
- php-fig.org

Cursos:
- codeschool.com
- codecademy.com
- pluralsight.com
- egghead.io
- https://pt.coursera.org/learn/interactive-python-1
- https://pt.coursera.org/learn/interactive-python-2
- Falta o curso do Code Barbarian
- Falta o Curso de SaaS com Rails

Testes
- phpunit.de
- rspec.info
- Seleniumhq.org

- jasmine.github.io
- mochajs.org
- sinonjs.org
- karma-runner.github.io

Frameworks:
- rails.org
- cakephp.org (ops, “sem querer”. Podia ser gosto pessoal?)
- symfony.com
- zend. qq coisa (joga fora. Brincadeira)
- Não é bem um framework, mas é muito importante: br.wordpress.org

Ferramentas diversas:
- nodejs.org
- phpqa.io
- github.com (não é o local para “baixar” o Git)

Boas práticas
- Ruby: não precisa (brincadeira). https://github.com/rubocop-hq/rubocop
- codex.wordpress.org
- php-fig.org

Alguns caras para seguir (sem desmerecer os demais. É que senão ia ter umas 1000 linhas…)
- Erick Wendel (novíssimo, muito bom)
- Wiliam Grasel (gente finíssima)
- Eu (brincadeira, estou o Bozo hoje)
- Fabien Potencier (muito bom, um dos melhores de PHP)
- Maujor (todos amamos o Maujor)
- Meu chefe (ixi, não tem blog…)
- Kim Kapawan (personagem inexistente de jogo não pode? E o Ken? E o Ryu? E o Blanka?)
- Augusto Pascutti (blog.augustopascutti.com)

Composer
- globoesporte.com (brincadeira)
- getcomposer.org

Bom, agora chega. Já te torrei demais!