As a **DevOps engineer**, you'll encounter a variety of file extensions depending on the tools and tasks you're working with. Here's a comprehensive list of important file extensions you should know:

**✅ Important File Extensions for DevOps Engineers**

1. **Configuration Files:**
   * .yml, .yaml: YAML files, used for configuration (e.g., Ansible, Kubernetes, Docker Compose).
   * .json: JSON files, commonly used for configurations (e.g., Kubernetes, AWS).
   * .conf: Configuration files used by various applications and servers (e.g., Nginx, Apache).
   * .ini: Configuration files used in many systems (e.g., PHP, MySQL).
   * .properties: Java properties files used for configuration (e.g., Spring Framework).
2. **Scripts:**
   * .sh: Shell script, commonly used for Linux or macOS command-line tasks.
   * .bash: Bash script.
   * .ps1: PowerShell script, commonly used for Windows automation.
   * .py: Python script, often used for automation and orchestration.
   * .pl: Perl script, less common but used in some legacy systems.
3. **Version Control:**
   * .gitignore: Specifies files to be ignored by Git.
   * .gitattributes: Defines attributes for paths in Git repositories.
   * .patch: Git patch files, used for diffing and applying changes.
   * .gitmodules: Configuration file for Git submodules.
4. **Containerization and Orchestration:**
   * .Dockerfile: Instructions for creating Docker images.
   * .dockerignore: Specifies files to be ignored when building Docker images.
   * .kube: Kubernetes configuration files (often .yaml).
   * .helm: Helm charts for Kubernetes deployment.
   * .json or .yaml: Used for Kubernetes resources like pods, deployments, services.
5. **CI/CD:**
   * .gitlab-ci.yml: GitLab CI configuration file.
   * .travis.yml: Travis CI configuration file.
   * .circleci/config.yml: CircleCI configuration file.
   * .jenkinsfile: Jenkins pipeline configuration (usually in Groovy syntax).
   * .azure-pipelines.yml: Azure DevOps pipeline configuration.
6. **Logs:**
   * .log: Generic log files (e.g., application logs, server logs).
   * .out: Output log files.
   * .err: Error log files.
7. **Infrastructure as Code:**
   * .tf: Terraform configuration files.
   * .json: For AWS CloudFormation templates.
   * .yaml: For Kubernetes or Ansible playbooks.
   * .cfn: CloudFormation templates.
   * .arm: Azure Resource Manager templates.
8. **Monitoring & Alerts:**
   * .conf: Config files for monitoring tools (e.g., Prometheus, Nagios).
   * .json: Alerting and monitoring configuration (e.g., Datadog).
9. **Database:**
   * .sql: SQL scripts for database management (e.g., MySQL, PostgreSQL).
   * .db: Database file (e.g., SQLite).
10. **Others:**
    * .tar, .tar.gz, .tgz: Archive files (used for packaging).
    * .zip, .rar: Compressed files.
    * .pem, .crt, .key: SSL certificates and private keys.
    * .csr: Certificate Signing Request file.

**✅ Git Branching Strategy**

Effective Git branching strategies are key to smooth collaboration in teams. Here’s an overview of common strategies:

1. **Git Flow**:
   * **Main branches**:
     + master (or main): The stable, production-ready code.
     + develop: The main development branch where features are integrated.
   * **Supporting branches**:
     + **Feature branches** (feature/feature-name): Used for developing new features. They branch off develop and are merged back into develop when complete.
     + **Release branches** (release/1.0.0): Created from develop when preparing for a release. Bug fixes and final tweaks happen here before merging into master and develop.
     + **Hotfix branches** (hotfix/1.0.1): Created from master to fix urgent issues in production. Once fixed, they are merged back into both master and develop.

**Use Case**: Best for teams with strict release cycles and versioning.

1. **GitHub Flow**:
   * This is a simpler, more streamlined version compared to Git Flow.
   * **Main branch**: main (production-ready code).
   * **Feature branches** (feature/feature-name): Each feature or bug fix is developed in its own branch, then merged back into main after review.
   * Typically used with **continuous deployment**.

**Use Case**: Best for teams that deploy frequently and prefer a simpler workflow.

1. **GitLab Flow**:
   * Combines **Git Flow** and **GitHub Flow** into a more customizable approach.
   * Incorporates environment-based workflows (e.g., staging, production).
   * It has different variations depending on deployment strategy (like environment branches).

**Use Case**: Works well for teams that want to manage releases based on environments like staging, production, etc.

1. **Trunk-Based Development**:
   * Focuses on a single **main** branch (trunk), with short-lived feature branches.
   * Developers commit directly to main after completing small pieces of work or after each day’s work.
   * Works best when paired with **continuous integration** and **continuous delivery** (CI/CD).

**Use Case**: Best for teams that need rapid development and continuous deployment with high collaboration.

1. **Release Candidate (RC) Flow**:
   * Used when managing release candidates (pre-production versions).
   * Similar to Git Flow but focused on RCs before they become production-ready.

**Use Case**: Best for teams that need to manage pre-release versions and want to distinguish between candidate and stable code.