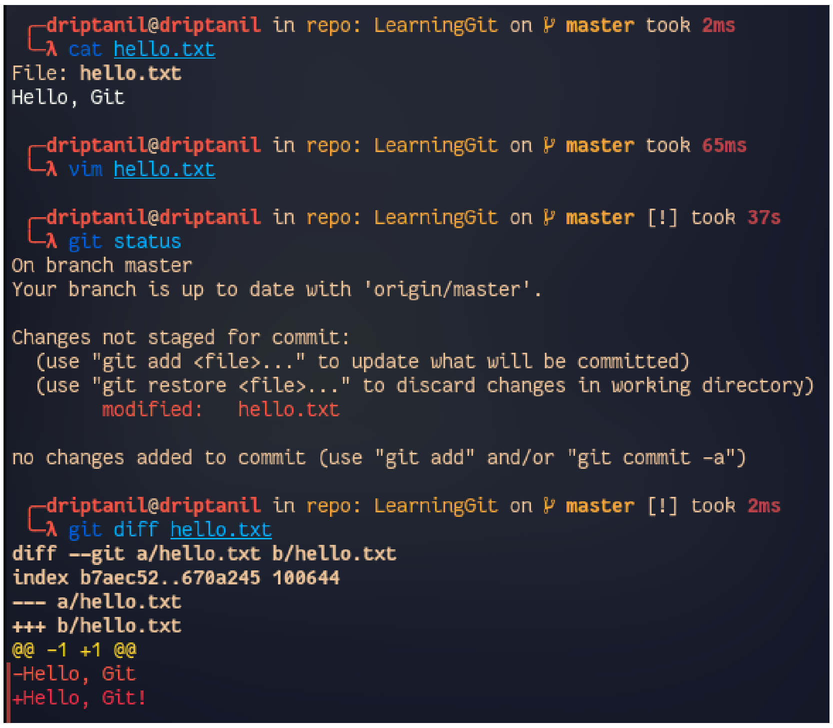
Git for Professionals

Perfect Git Commit

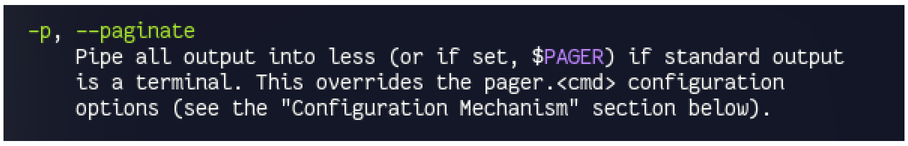
- File staging helps in better commits, (easier to manage, review code and adding commit messages).

- Use `git diff <file>`, to view the changes made to the not staged file.

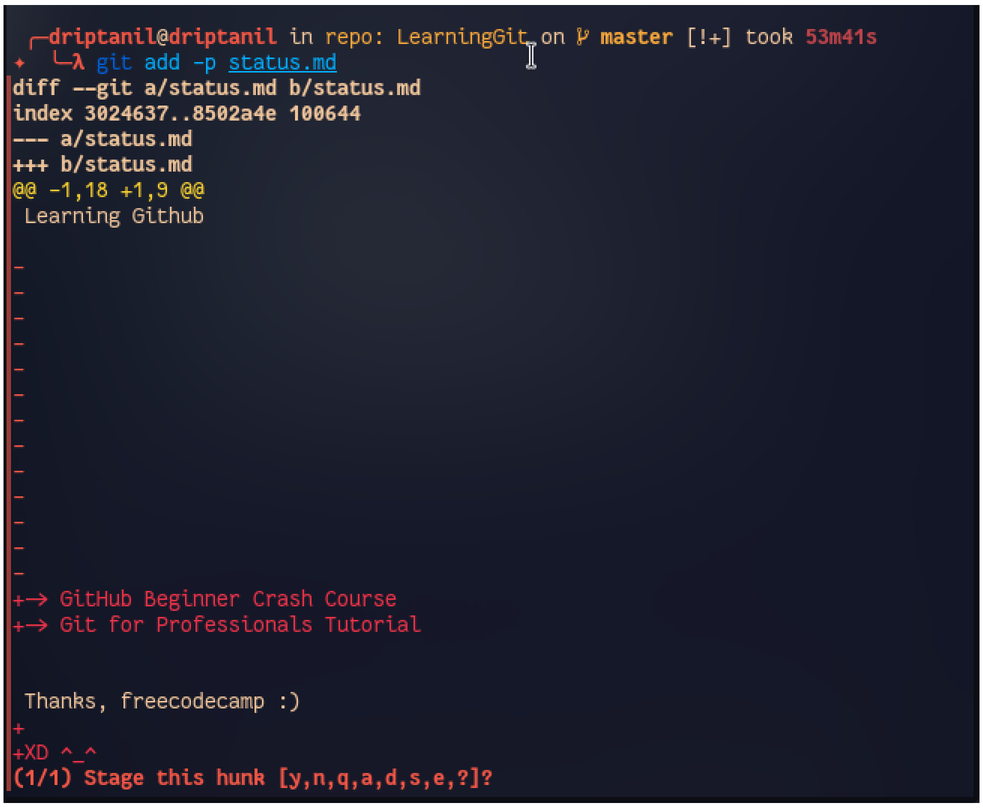


```
┌─driptanil@driptanil in repo: LearningGit on ᚠ master took 2ms
└─λ cat hello.txt
File: hello.txt
Hello, Git

┌─driptanil@driptanil in repo: LearningGit on ᚠ master took 65ms
└─λ vim hello.txt

┌─driptanil@driptanil in repo: LearningGit on ᚠ master [!] took 37s
└─λ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.txt

no changes added to commit (use "git add" and/or "git commit -a")

┌─driptanil@driptanil in repo: LearningGit on ᚠ master [!] took 2ms
└─λ git diff hello.txt
diff --git a/hello.txt b/hello.txt
index b7aec52..670a245 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1 @@
-Hello, Git
+Hello, Git!
```

- Use `git -p` to stage certain chucks of changes in a file.



```
  -p, --paginate
      Pipe all output into less (or if set, $PAGER) if standard output
      is a terminal. This overrides the pager.<cmd> configuration
      options (see the "Configuration Mechanism" section below).
```

- 

```
┌─driptanil@driptanil in repo: LearningGit on ᚠ master [!+] took 53m41s
✦ └─λ git add -p status.md
diff --git a/status.md b/status.md
index 3024637..8502a4e 100644
--- a/status.md
+++ b/status.md
@@ -1,18 +1,9 @@
 Learning Github

-
-
-
-
-
-
-
-
-
-
-
-
-
+-> GitHub Beginner Crash Course
+-> Git for Professionals Tutorial


 Thanks, freecodecamp :)
+
+XD ^_^
(1/1) Stage this hunk [y,n,q,a,d,s,e,?]?
```

  - `y` -> stage hunk
  - `n` -> do not stage hunk
  - `a` -> stage all remaining hunks
  - `d` -> do not stage any remaining hunks
  - `s` -> split the current hunk
  - `e` -> manually edit the current hunk

- Use `git commit` to open a text editor (default: vi editor, or use `git config --global core.editor <editor_name>`)

- Format for commit message:

  1. 1st line: Subject
  2. 2nd line: ``
  3. 3rd line: Body
     a. What is now different than before?
     b. What is the reason for the change?
     c. Is there anything to watch out for / anything particularly remarkable

Branching Strategies

Convention:

(agree on a Branching Work-flow in a team)

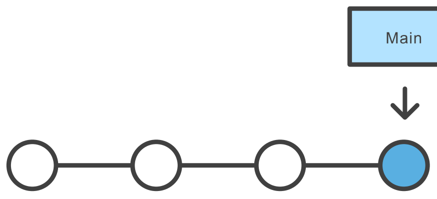
1. Git allows users to create branches - learning it for better usage would improve the work-flow
2. A written best practice / strategy to ideally structure work in between team members
3. Team members, team size and type of project plays a huge role in managing project releases.
4. Helping new on-board team members to understand the work-flow to reduce conflicts

Integrating Changes & Structuring Releases

Mainline Development

(always be integrating)

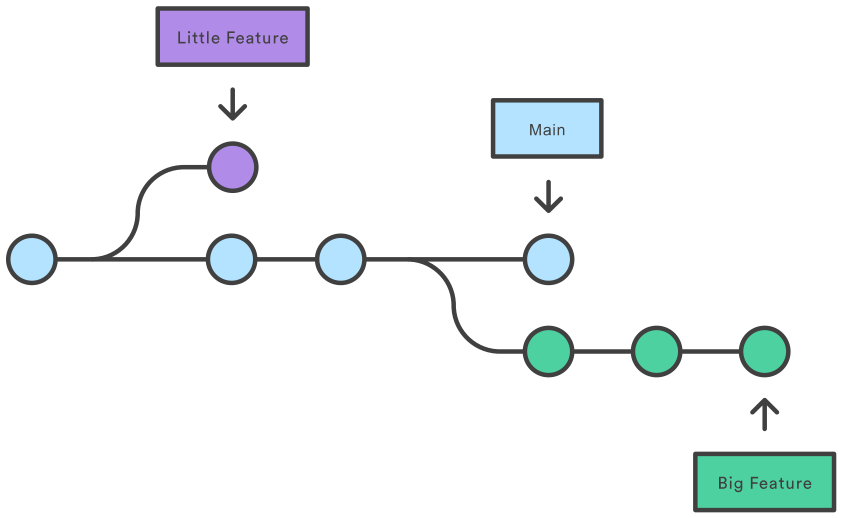
-> few branches
-> relatively small commit
-> high-quality testing & QA standards



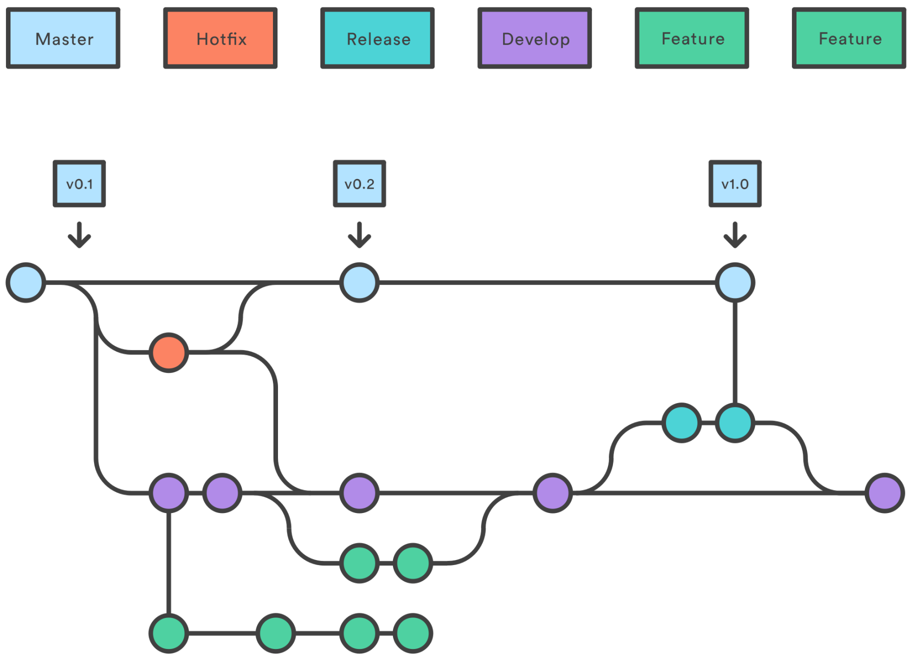
State, Release, and Feature Branches

(Branches Enhance Structures & Workflows)

-> different types of branches
-> fulfil different types of jobs



Long-Running & Short-Lived Branches



Long Running Branches

- Long Running Branches are the branches which exist through out the project.
- Every Git repository has a long running branches like main or master branch.
- Some repositories have develop or staging long running branches, generally for stability testing before merging to master branch.
- These branches represent the different stages of release and deployment.
- Commit are not directly made to long-running branches.

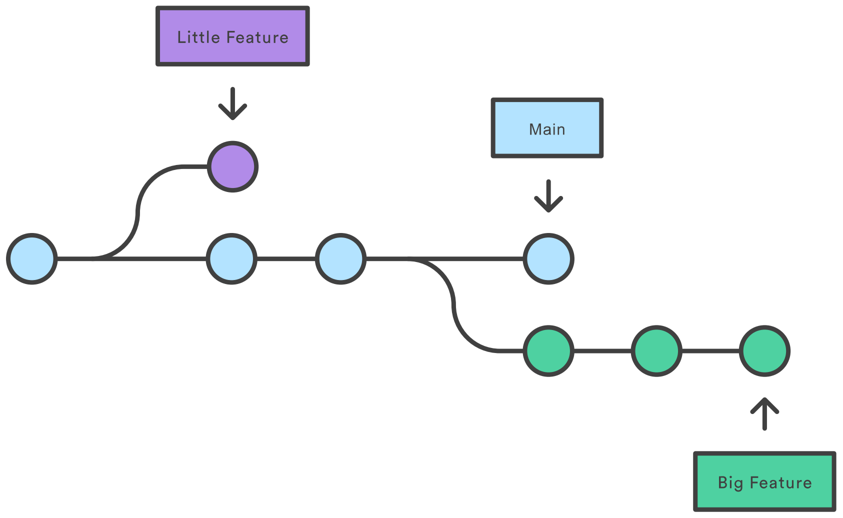
Short Lived Branches

- They are created for certain purposed (like new features, bug fixes, refactoring).
- They are deleted after being merged or rebased to the long running branch.
- A short lived branch is based on a long running branch.

Git Workflows

GitHub Flow

- It consists of only 1 long running branch (main) and working changes are made in short lived branches.

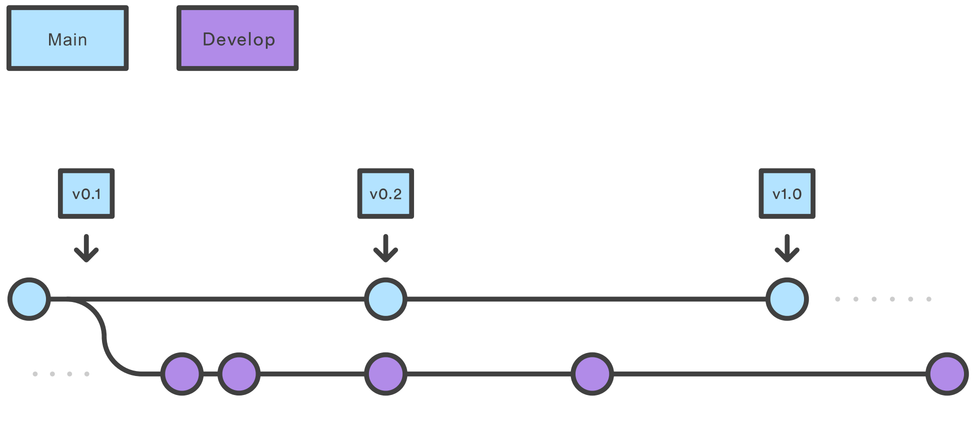


Git-Flow

- It consists of 2 long running branches (main + develop) and short lived branches (feature, release, hotfixes).

Develop Branch

- The develop branch serves as an integration branch of feature.

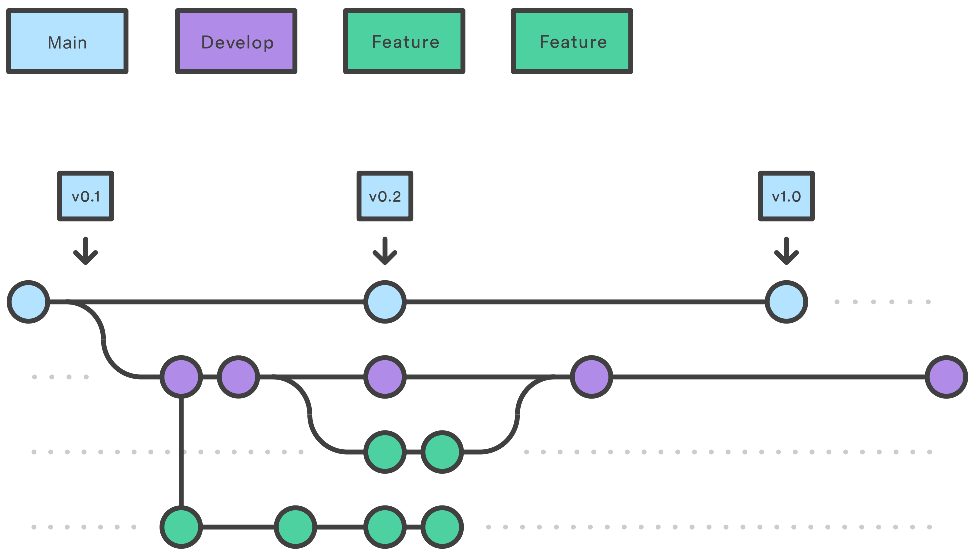


Creating develop branch

- Using git, `git branch develop` & `git push -u origin develop`
- Using git-flow extension, `git flow init`

Feature Branch

- The *feature* branch uses *develop* as their parent branch and feature never directly interacts with *main*.



Creating feature branch

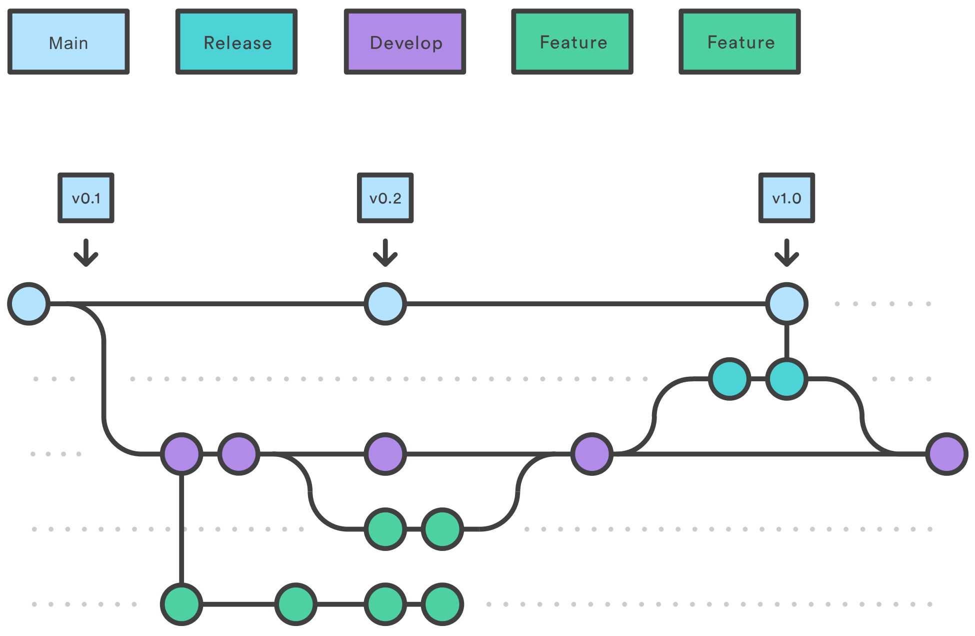
- Using git, `git checkout develop` & `git checkout -b feature_branch`
- Using git-flow extension, `git flow feature start feature_branch`

Merging feature to develop

- Using git, `git checkout develop` & `git merge feature_branch`
- Using git-flow, `git flow feature finish feature_branch`

Release Branch

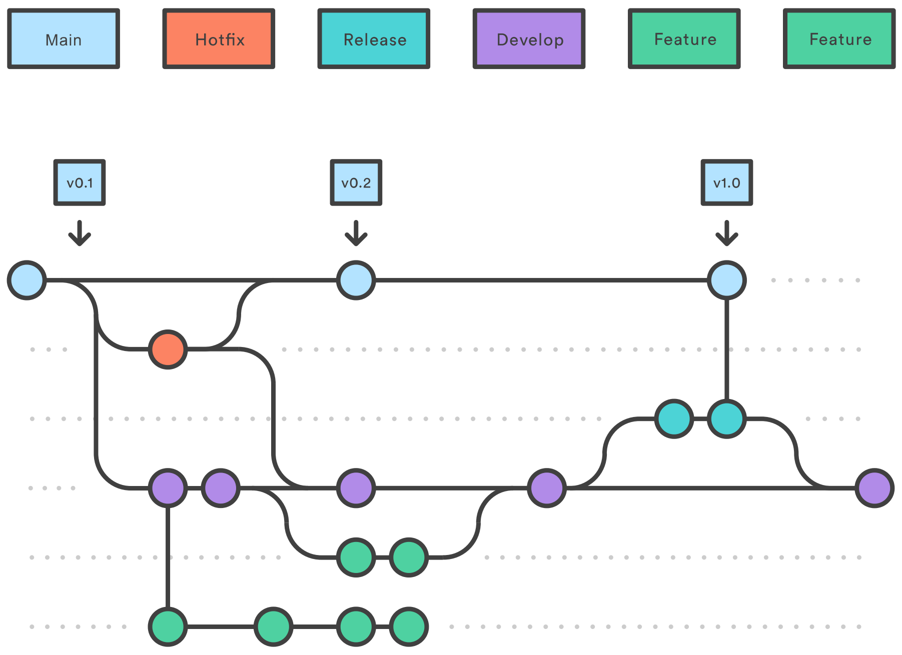
- After *develop* branch has acquired enough features for a release.
- A *release* branch is branched off *develop* branch.
- Documentations, bug fixes, and release-oriented tasks of new features are pushed to *release* branch.



- Using git, `git checkout develop` & `git checkout -b release/<version>`.
- Using git flow extension, `git flow release start <version>`.
- Once *release* branch is ready, it is merged to *main* branch.
- *release* branch is merged back to *develop* branch and *release* branch is deleted
- Using git, `git checkout main` & `git merge release/<version>`
- Using git flow extension, `git flow release finish <version>`

Hotfix Branch

- The *hotfix* branched off *main* branch, for quick patch production releases.
- As soon as the fix is complete, it is merged into *main* & *develop* branch.
- A dedicated branch for bug fixes, helps team to address issues and speed up the release cycle.



- Using git, `git checkout main` & `git checkout -b hotfix_branch`.
- Using git flow extension `git flow hotfix start hotfix_branch`.

Pull Requests

- Pull request is request to merge changes to other repositories (pushing changes are not allowed).
- It gets many developers involved for reviewing the code and fix bugs (if any) before merging it to *main* branch.

Fork Repository

- Fork creates a copy of a repository with administrator permissions, to which we are allowed to make changes.