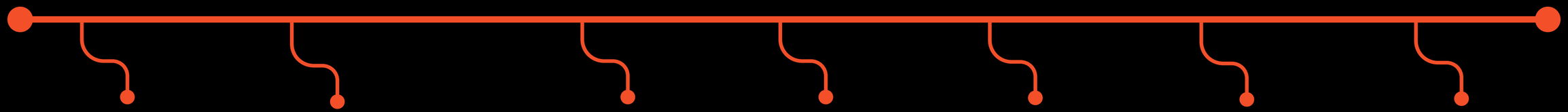
```
 1 file changed, 1 insertion(+)

HP@Nafisa MINGW64 ~/git_training (master)
$ git show HEAD
commit 30c958ef29b50ee485d4e1cb5becaaf0af1c0b24 (HEAD -> master)
Author: Nafisay <73686602+Nafisay@users.noreply.github.com>
Date:   Fri May 19 15:10:14 2023 +0530

    naw

diff --git a/B      /Bob.txt
index 90fe8      0538      0644
--- a/Bob
+++ b/Bo
@@ -1 +
 Hi!! I'      ere for a while now.
+Lmao Bob!
\ No newline      nd      le

HP@Nafisa MINGW6      it_training (ma
$ git commit --amend --date=Sat May 20
error: pathspec 'May' did not match any file(s) known to git
error: pathspec '20' did not match any file(s) known to git

HP@Nafisa MINGW64 ~/git_training (master)
$ git commit --amend --date=20/05/2023
[master e4fd4d2] nawr
 Author: Nafisay <73686602+Nafisay@users.noreply.github.com>
 Date: Sat May 20 16:21:29 2023 +0530
 1 file changed, 1 insertion(+)

HP@Nafisa MINGW64 ~/git_training (master)
```

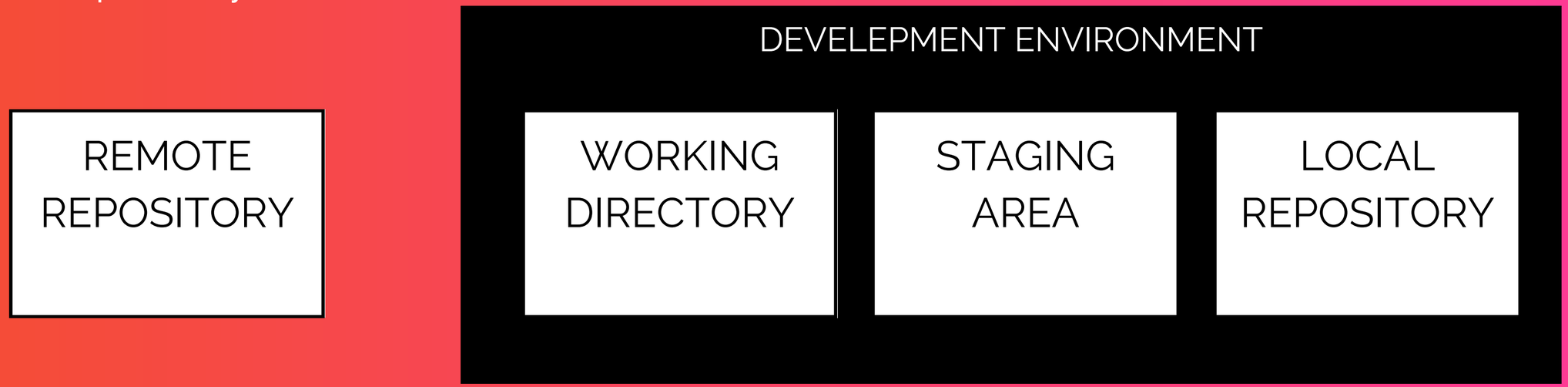
WHAT IS GIT?

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.



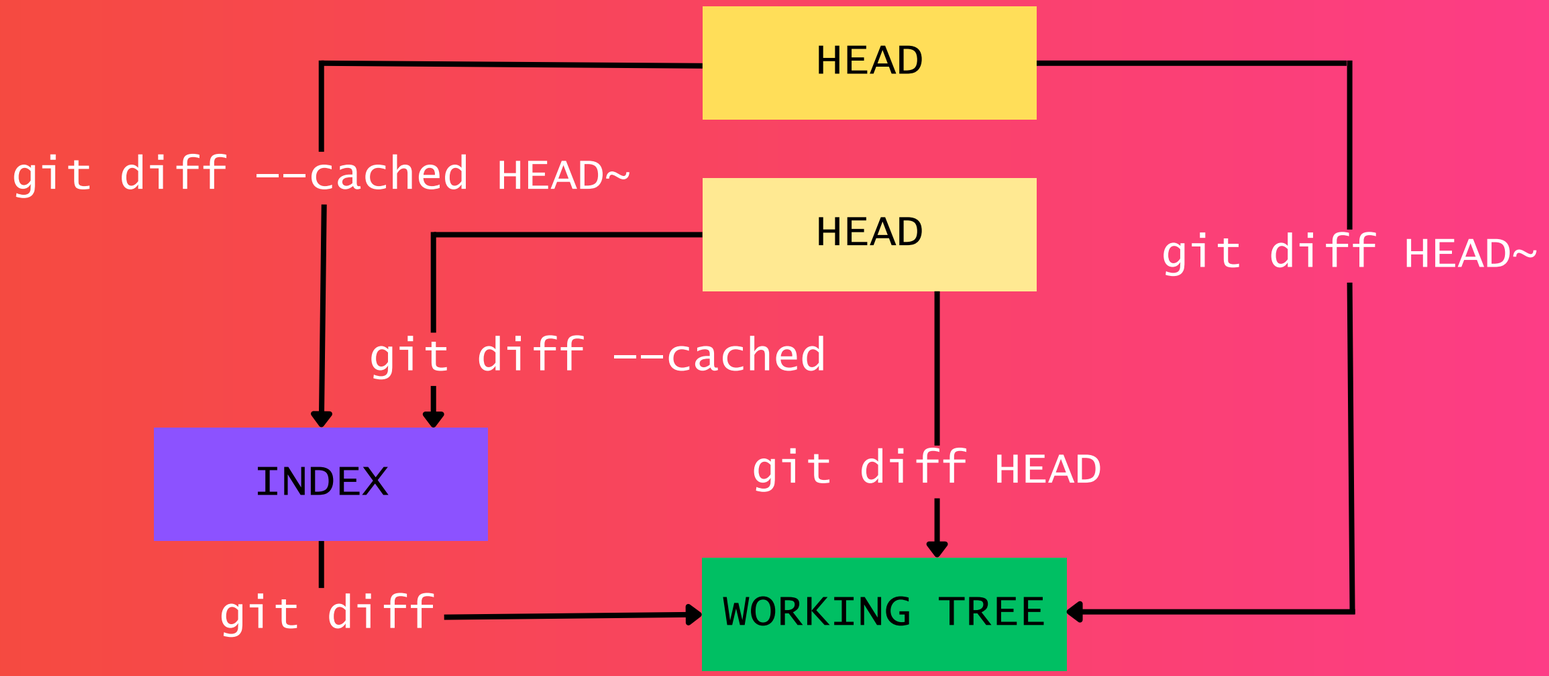
THE BASICS

- Remote Repository- Remote repositories are versions of our project that are hosted on the Internet or network somewhere.
- Development Environment- The Development Environment is what we have on our local machine. The three parts of it are our Working Directory, the Staging Area and the Local Repository.



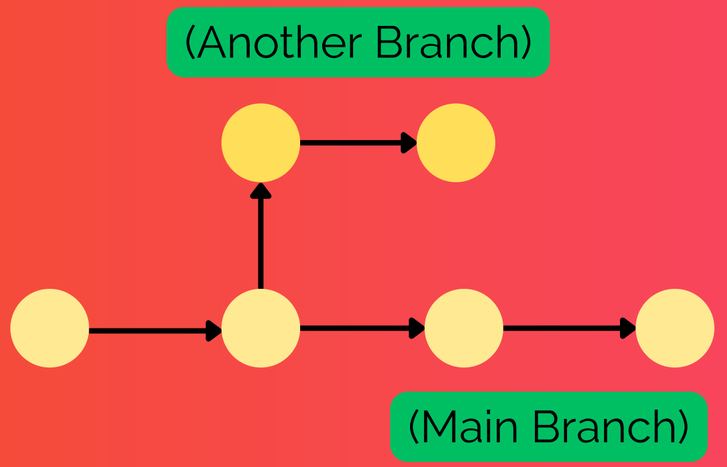
git diff

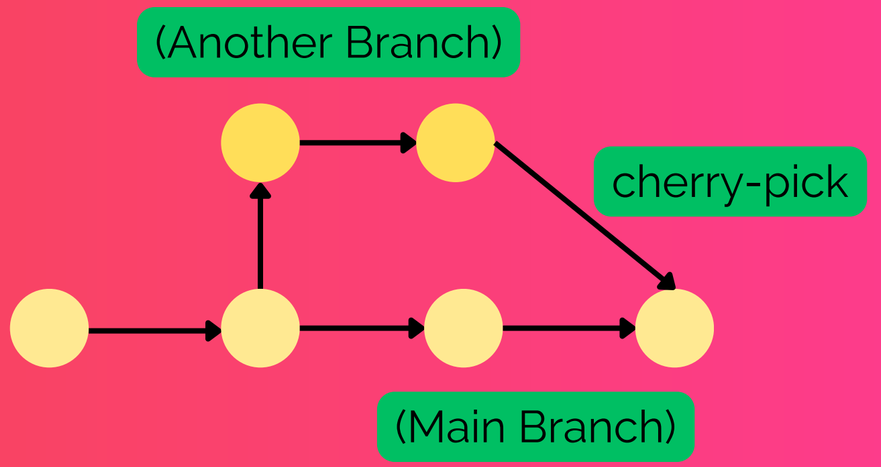
Diffing is a function that takes two input data sets and outputs the changes between them. git diff is a multi-use Git command that when executed runs a diff function on Git data sources. These data sources can be commits, branches, files and more.



`git cherry-pick`

Cherry-picking in git means choosing a commit from one branch and applying it to another branch. This is in contrast with other ways such as merge and rebases which normally apply many commits into another branch. This requires our working tree to be clean (no modifications from the HEAD commit).





Note: While using this command make sure you are on the branch you want to apply the commit.

git cherry-pick contd.

Some important Usecases of Cherry-pick are as follows:

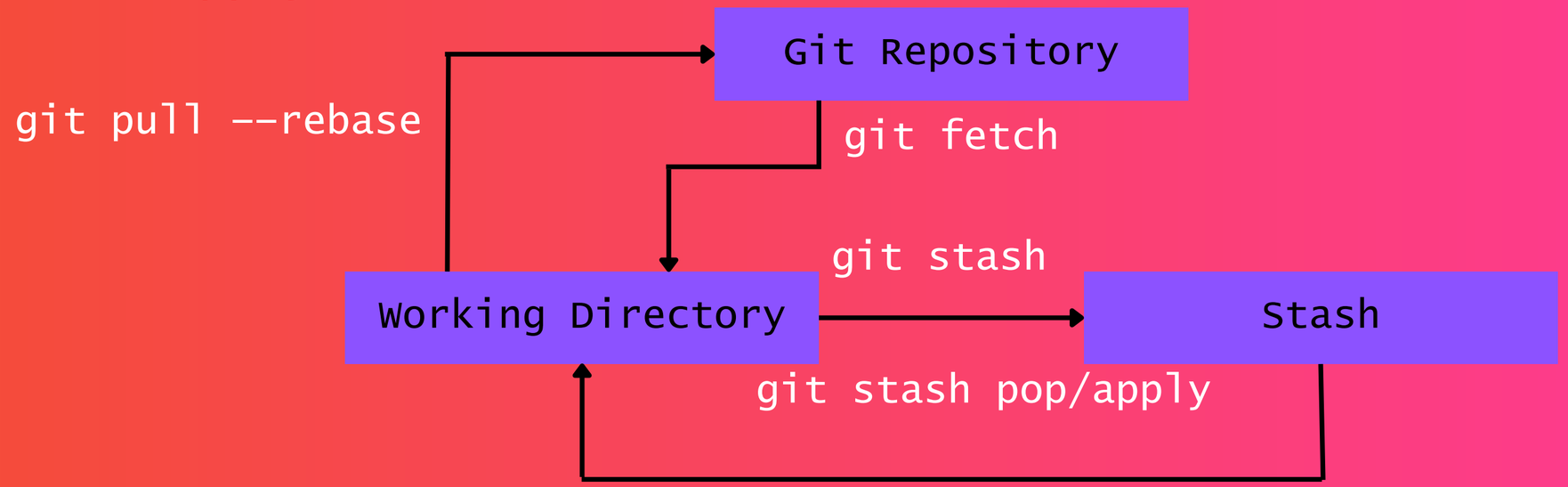
1. If we by mistake make a commit in an incorrect branch, then using cherry-pick we can apply the required changes.
2. Suppose when the same data structure is to be used by both the frontend and backend of a project. Then a developer can use cherry-pick to pick the commit and use it to his/her part of the project.
3. When a bug is found and it is critical to resolve.

Disadvantages of using Cherry Pick

Cherry-pick should not be used always as it can cause copy commits; conventional merges are liked all things considered. Also, in the situation where the commits from 2 or more branches update similar lines of code and we git cherry-pick one commit to the other branch, it prompts a conflict.

`git stash`

It is used when we want to record the current state of the working directory and the index, but want to go back to a clean working directory. The command saves our local modifications away and reverts the working directory to match the HEAD commit.
The modifications stashed away by this command can be listed with `git stash list`, inspected with `git stash show`, and restored (potentially on top of a different commit) with `git stash apply`.



`git stash contd.`

Creating A Branch From Your Stash (Git Stash Branch)

If we want to create and check out a new branch starting from the commit at which the stash was originally created and apply the changes saved in the stash, we can use `git stash <branch_name> <stash_name>`. It drops the stash which is given as the argument and if no stash is given, it drops the latest one.

Cleaning Up Your Stash (Git Stash Clear)

To delete any particular stash (For ex:- stash@{1}), use `git stash drop stash@{1}`. By default, this command will drop stash@{0} if no argument is provided (`git stash drop`). To delete all stashes at once, use the `git stash clear` command.

`git rebase`

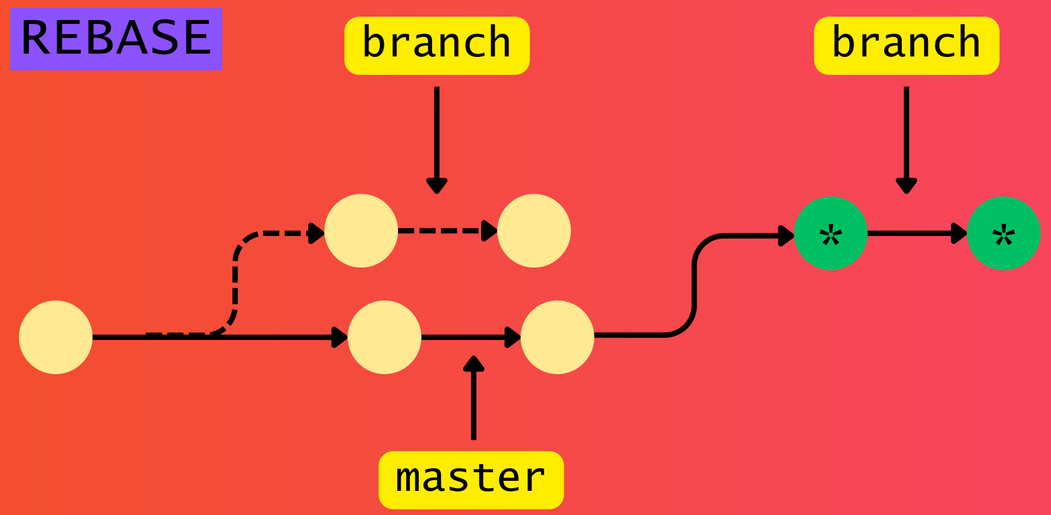
Rebasing is changing the base of our branch from one commit to another making it appear as if we had created our branch from a different commit. Internally, Git accomplishes this by creating new commits and applying them to the specified base.

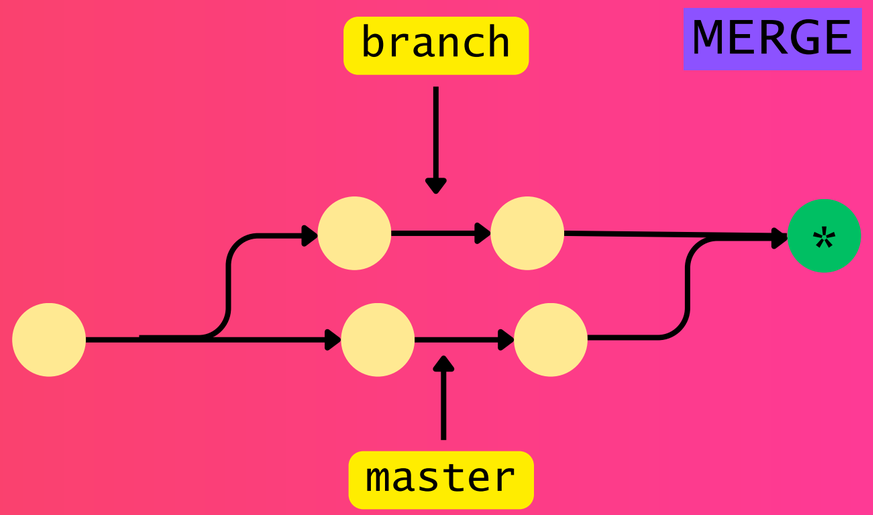
`git rebase -i`

Git rebase interactive is when git rebase accepts an - i | --interactive argument.
Running git rebase with the -i flag begins an interactive rebasing session. Instead of blindly moving all of the commits to the new base, interactive rebasing gives you the opportunity to alter individual commits in the process. This lets you clean up history by removing, splitting, and altering an existing series of commits.

git rebase vs. git merge

Both of these commands are designed to integrate changes from one branch into another branch—they just do it in very different ways.
Rebasing is a common way to integrate upstream changes into our local repository. Pulling in upstream changes with git merge results in a superfluous merge commit every time we want to see how the project has progressed.

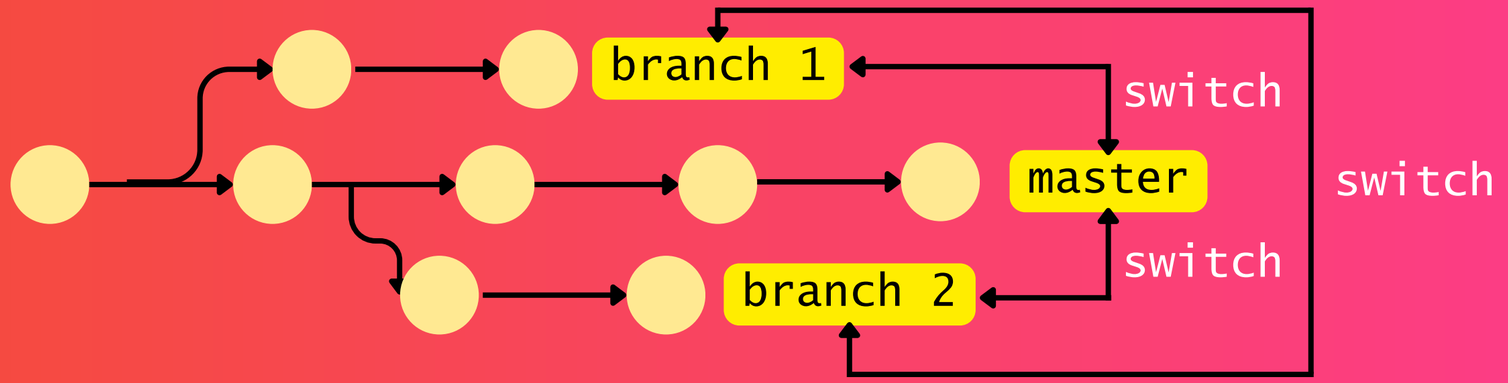




git switch

It is used to switch to a specified branch. The working tree and the index are updated to match the branch. All new commits will be added to the tip of this branch. Switching branches does not require a clean index and working tree.

Optionally a new branch could be created with either -c, -C, automatically from a remote branch of same name, or detach the working tree from any branch with --detach, along with switching.



`git switch contd.`

The git switch command replaced git checkout in 2020, although git checkout is still a supported command. The git checkout command performs two functionalities; "switch branch" and "restore working tree files". To separate these two functionalities, Git introduced the git switch command, which replaces the "switch branch" feature of "git checkout".

Let's assume we have a file named "test.txt" and at the same time, we have a branch named "test". If we are on master branch and we want to checkout to branch "test", we would use the command `git checkout test` but this would checkout the file "test", this is where git switch comes in. `git switch test` will switch to branch "test" even if you have a file "test"

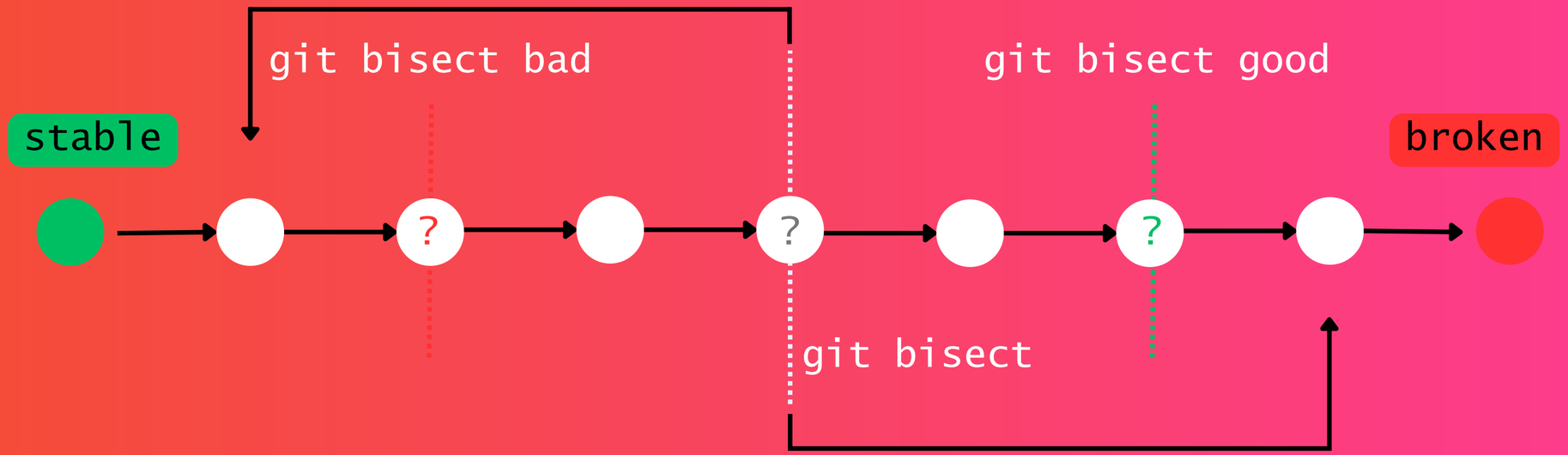
`git reflog`

This command manages the information recorded in the reflogs.
Reference logs, or "reflogs", record when the tips of branches and other references were updated in the local repository. Reflogs are useful in various Git commands, to specify the old value of a reference. For example, `HEAD@{2}` means "where HEAD used to be two moves ago", `master@{one.week.ago}` means "where master used to point to one week ago in this local repository", and so on.
The reflog covers all recent actions, and in addition the HEAD reflog records branch switching.

- The "expire" subcommand prunes older reflog entries.
- The "delete" subcommand deletes single entries from the reflog.

git bisect

This command uses a binary search algorithm to find which commit in our project's history introduced a bug. We use it by first telling it a "bad" commit that is known to contain the bug, and a "good" commit that is known to be before the bug was introduced. Then git bisect picks a commit between those two endpoints and asks us whether the selected commit is "good" or "bad". It continues narrowing down the range until it finds the exact commit that introduced the change.



`git bisect` contd.

In fact, git bisect can be used to find the commit that changed any property of our project; e.g., the commit that fixed a bug, or the commit that caused a benchmark's performance to improve.

Advantage: If the search range is of N commits, we should expect to test 1 + log2N commits with git bisect instead of roughly N / 2 commits with a linear search. This is especially advantageous when there are a large number of commits to be checked.

References

- https://git-scm.com/docs
- https://www.atlassian.com/git
- https://stackoverflow.com
- https://www.toptal.com/git/the-advanced-git-guide
- https://dev.to/unseenwizzard/learn-git-concepts-not-commands-4gjc
- https://dev.to/g_abud/advanced-git-reference-109j
- https://www.dennyzhang.com/git-concepts
- https://www.geeksforgeeks.org

Thank You!

~Nafisa Parvin