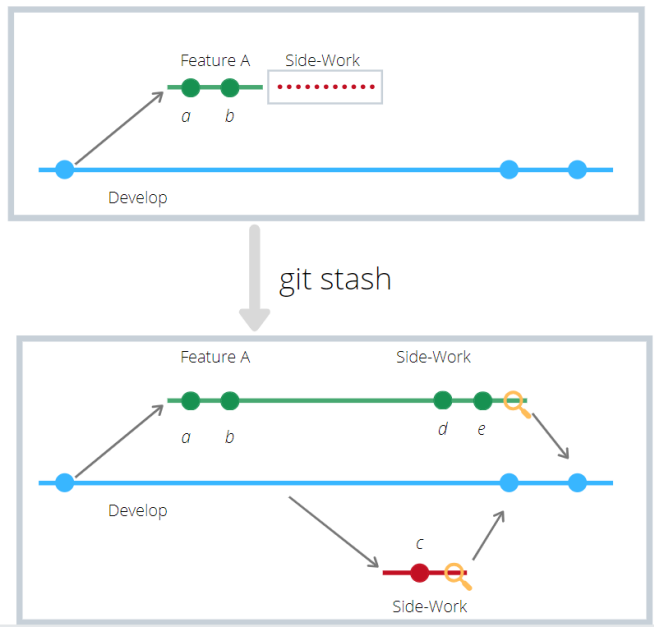
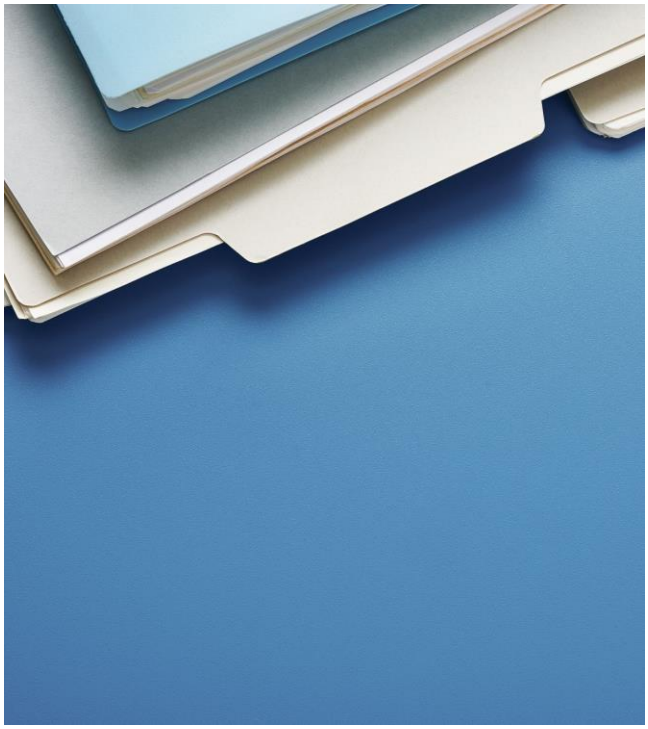
70

Technique 1: Stash



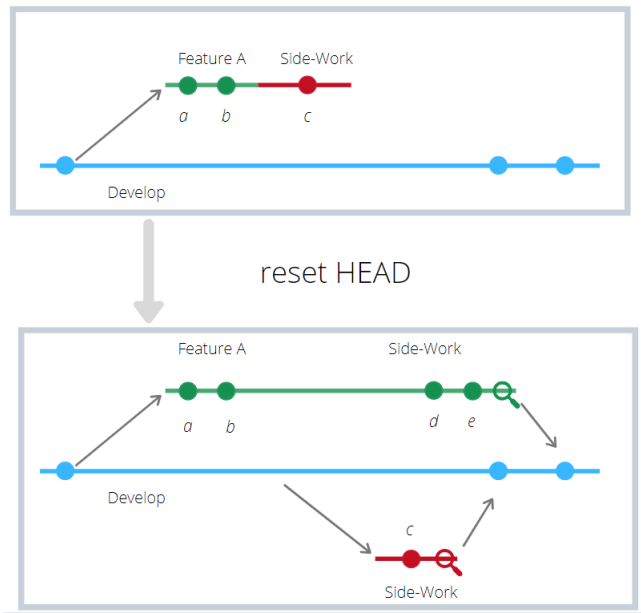
Use git stash

- Git stash allows you to temporary save your current uncommitted changes
- This is handy if you realized you started working on some unrelated part. Then, you can stash your changes, switch to a new branch and commit the changes there – instead, intermingled with your current work.
- Use git stash pop to get the changes back.
- Use stash show to get a list of the files in your stash.



72

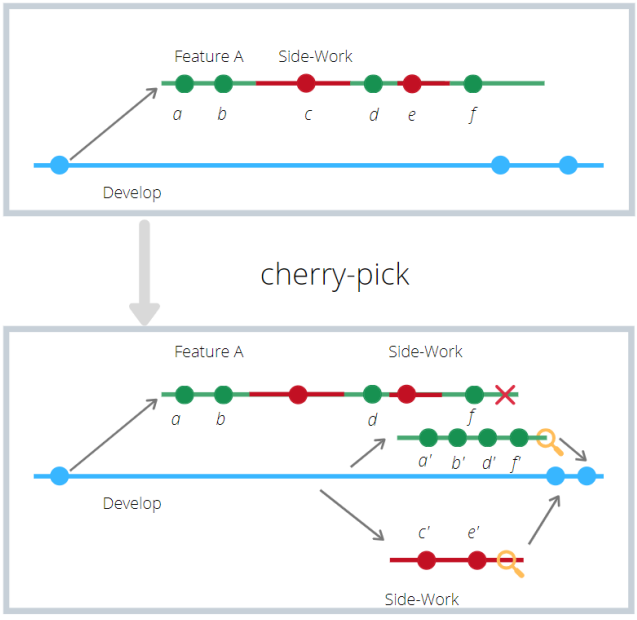
Technique 2: reset HEAD



Removing last commit from your large PR

1. If you have already committed your unrelated work, you can checkout a new branch, and reset the HEAD by following these steps:
2. Being on the feature_branch, create the new branch side_work (but not checking it out)
3. Set the HEAD of branch feature_branch back by one commit (to b):
4. git reset --hard HEAD~1
5. Rebase all commits that happened between branch feature_branch and branch side_work (which is only commit c) onto develop.
6. git rebase --onto develop feature_branch side_work
7. This results in two branches that each contain only related changes.

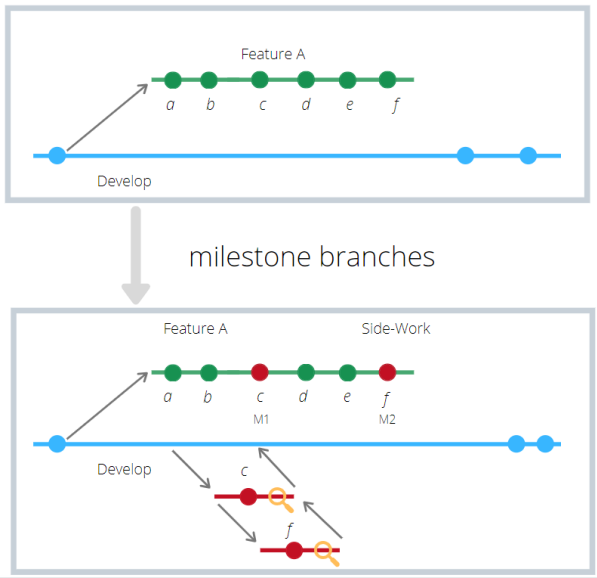
Technique 3: cherry-pick



Cherry-Pick

1. Checkout source branch:
   git checkout develop
2. Create a new branch:
   git checkout -b new_branch
3. Cherry-pick a subset of commits (eg. regarding some refactoring):
   git cherry-pick [commit_id_1] [commit_id_2]
4. Create a Merge/Pull Request
5. After this gets accepted, you must update your feature branch (merge master)
6. There's less code to check now
7. Repeat 2-5 until there's little enough code on base Pull Request (or no code at all)

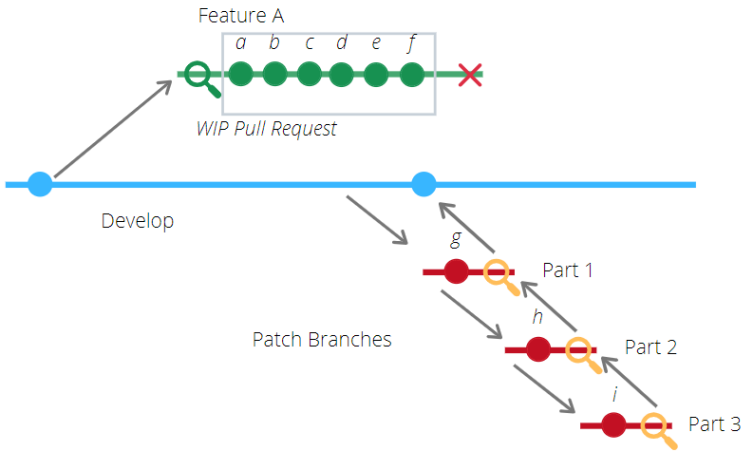
Technique 4: Milestone Branches



Milestone Branches

1.Checkout source branch
    git checkout feature_branch
2.Find milestone commits
3.Check out next earliest milestone commit
    git checkout <commit_id>
4.Create a new branch
    git checkout -b milestone_branch
5.Create a WIP-Merge Request
6.Repeat 3-6 until there are no more milestone commits

Technique 5: Patch Branches



Patch Branches

1.Create new feature branch
```
git checkout -b somefeature1 manyfeatures
```
2.Create unstaged changes
```
git reset master
```
3.Decide what should go in new commit
```
git add –patch
git stash --include-untracked --keep-index
git commit -m "[somefeature1 - PART1]
git push origin somefeature1
```
5.Create a WIP-Merge/Pull Request
6.Create new feature branch
```
git checkout -b somefeature2
git stash pop
```
7.Repeat 3-6 until there are no more changes

Git Patch Mode

Patch Mode

If you use git, you've used the command git add. But do you know about git add's "patch mode" using git add -p or git add --patch?

Patch mode allows you to stage *parts* of a changed file, instead of the entire file. This allows you to make concise, well-crafted commits that make for an easier to read history.

You can stage individual hunks of code changes within one file.

```
git add --patch
•y - Yes, add this hunk
•n - No, don't add this hunk
•d - No, don't add this hunk and all other remaining hunks.
•s - Split the hunk into smaller hunks.
•e - Manually edit the hunk.
```

Technique N: Combinations of Techniques

Stash – Cherry-picking – Milestone checkout – New composition