alterra academy

Version Control and Branch Management (Git)

alterra
academy

OUR RULES

Silent Mode

Ask Question

Go Toiled

TIME ALLOCATION

Explanation

Challenge

Review

OUTLINE

- What is Versioning?
- Git Install
- Setting Up
    - Git Init, Clone, Config
- Saving Changes
    - Git Status, Add, Commit, Diff, Stash, Ignore
- Inspecting Repository and Undoing
    - Git Log, Checkout, Reset
- Syncing
    - Git Remote, Fetch, Push, Pull
- Branches
- Pull Request
- Workflow Collaboration

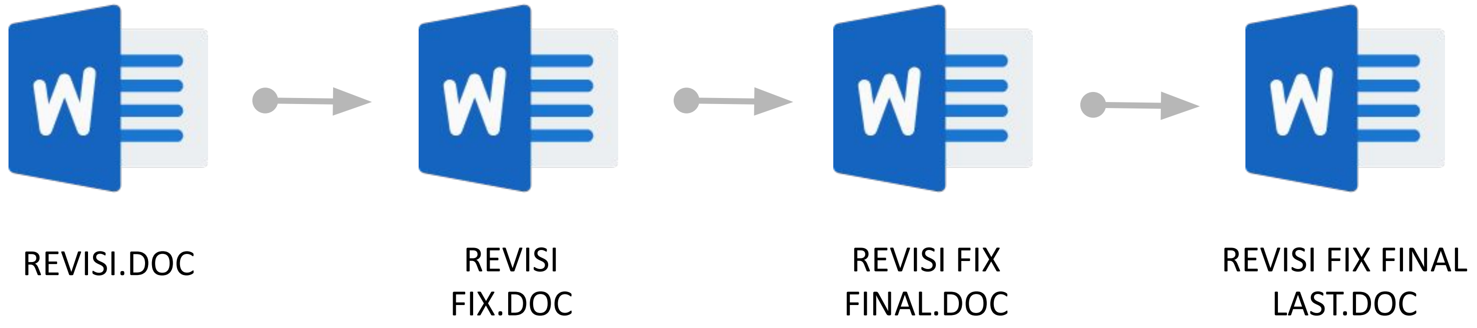
alterra academy

Let's get started!

APA ITU **VERSIONING**?

mengatur versi dari **source code** program

THE PROBLEM


REVISI.DOC
REVISI FIX.DOC
REVISI FIX FINAL.DOC
REVISI FIX FINAL LAST.DOC

alterra academy

“Revision Is A Must,
Don’t Expect Every Code is Perfect”

TOOLS

Version Control System (VCS)
Source Code Manager (SCM)
Revision Control System (RCS)

VERSION CONTROL SYSTEM

Single User

SCCS - 1972 Unix only
RCS - 1982 Cross platform, text only

Centralized

CVS - 1986 File focus
Perforce - 1995
Subversion - 2000 - track directory structure
Microsoft Team Foundation Server - 2005

Distributed

Git - 2005
Mercurial - 2005
Bazaar - 2005

GIT

Salah satu ***version control system*** populer yang digunakan para developer untuk mengembangkan software secara bersama-bersama

Real World
Collaboration

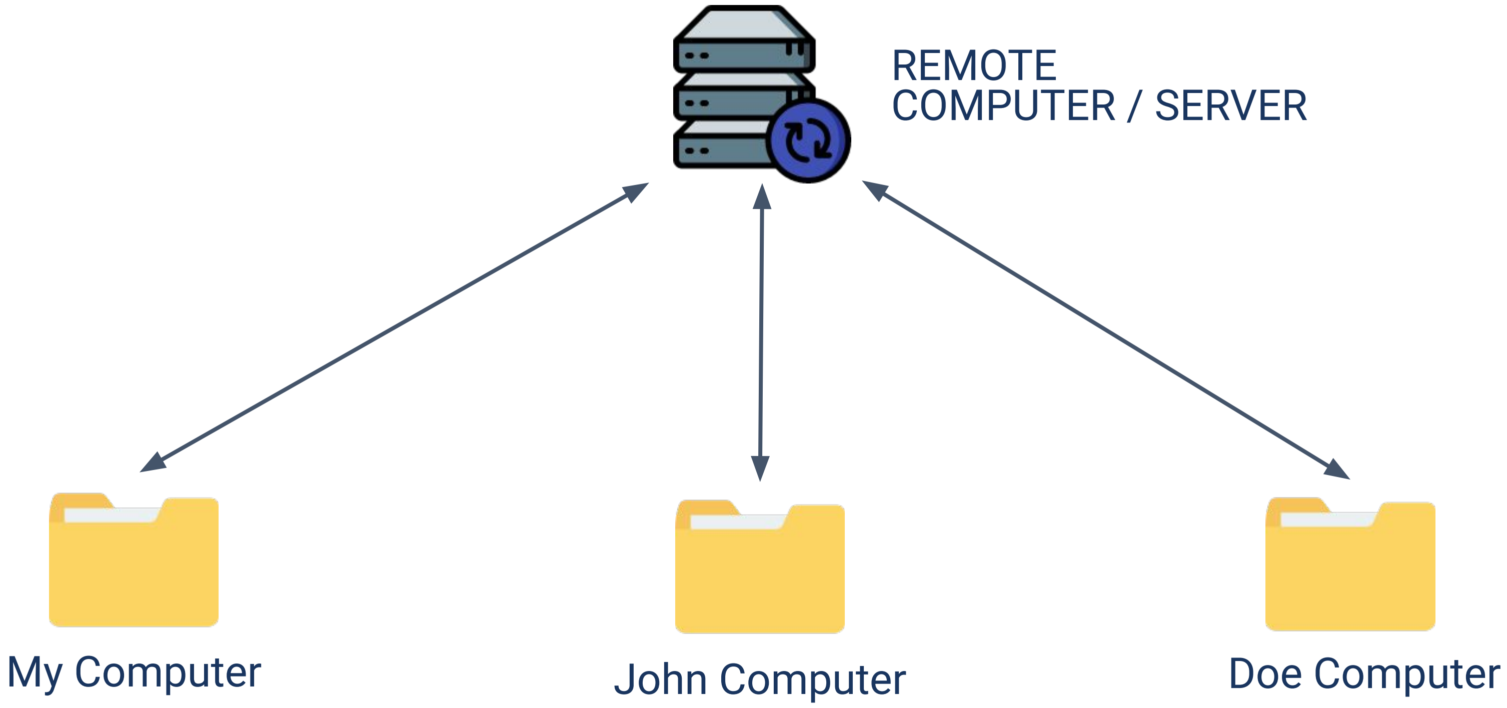
TERDISTRIBUSI

Bukan tersentralisasi

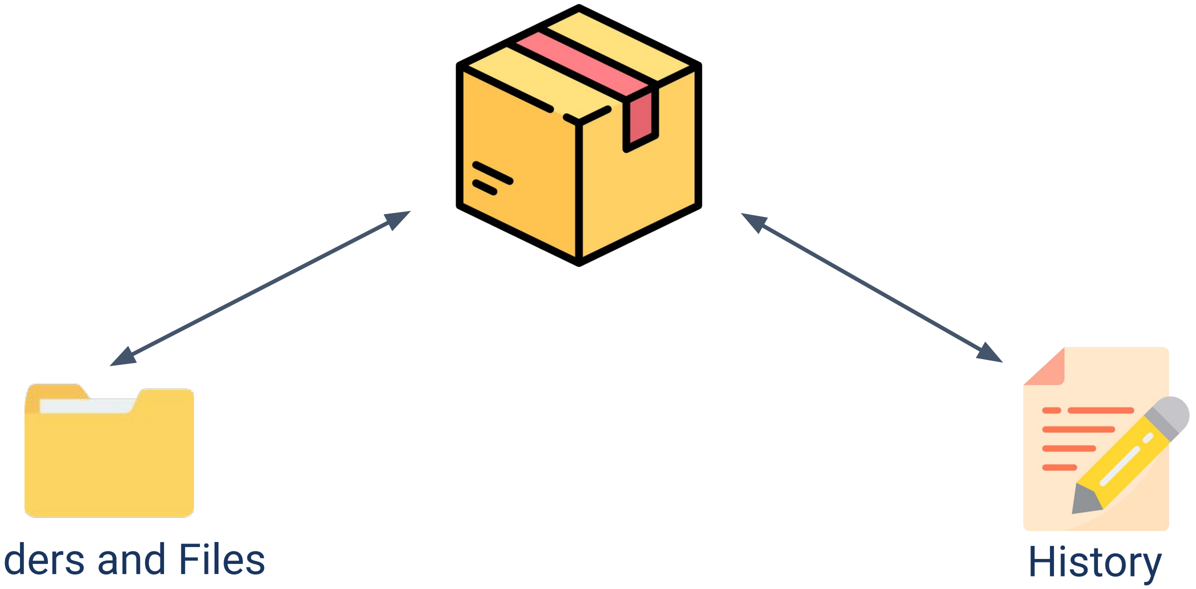
DIBUAT OLEH

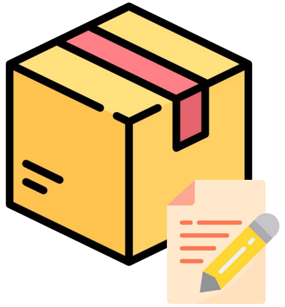
Linus Torvalds (2005) - Linux Kernel
https://github.com/torvalds/linux

EVERYONE SHOULD SYNC TO THE REMOTE SERVER



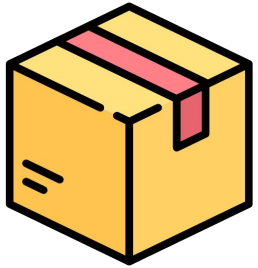
GIT REPOSITORY (FOLDER PROJECT)





Git track every file changes.

Your changes, John’s changes, everyone!

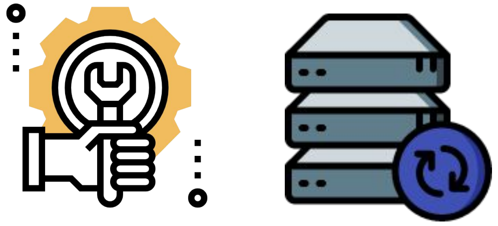


Git can undo to some ‘points’

We call it as Commit

Commit = the record of changes

It is quite complicated to setup git server,
We need service to be the server!



REMOTE COMPUTER / SERVER

github = git hosting service

Go to github.com and create new repository!

alterra
academy

GIT INSTALL

INSTALL GIT ON

1. Download the latest **Git for Mac installer**.
2. Follow the prompts to install Git.
3. Open a terminal and verify the installation was successful by typing: `git --version`:

output

```
$ git --version
git version 2.9.2
```

INSTALL GIT WINDOWS

1. Download the latest **Git for Windows installer**.
2. When you've successfully started the installer, you should see the **Git Setup** wizard screen. Follow the **Next** and **Finish** prompts to complete the installation. The default options are pretty sensible for most users.
3. Open a Command Prompt (or Git Bash if during installation you elected not to use Git from the Windows Command Prompt).

INSTALL GIT LINUX

1. From your shell, install Git using apt-get:

output

```
$ sudo apt-get update
$ sudo apt-get install git
```

2. Verify the installation was successful by typing git `--version`:

output

```
$ git --version
git version 2.9.2
```

SETTING UP

GIT INIT, CLONE, CONFIG

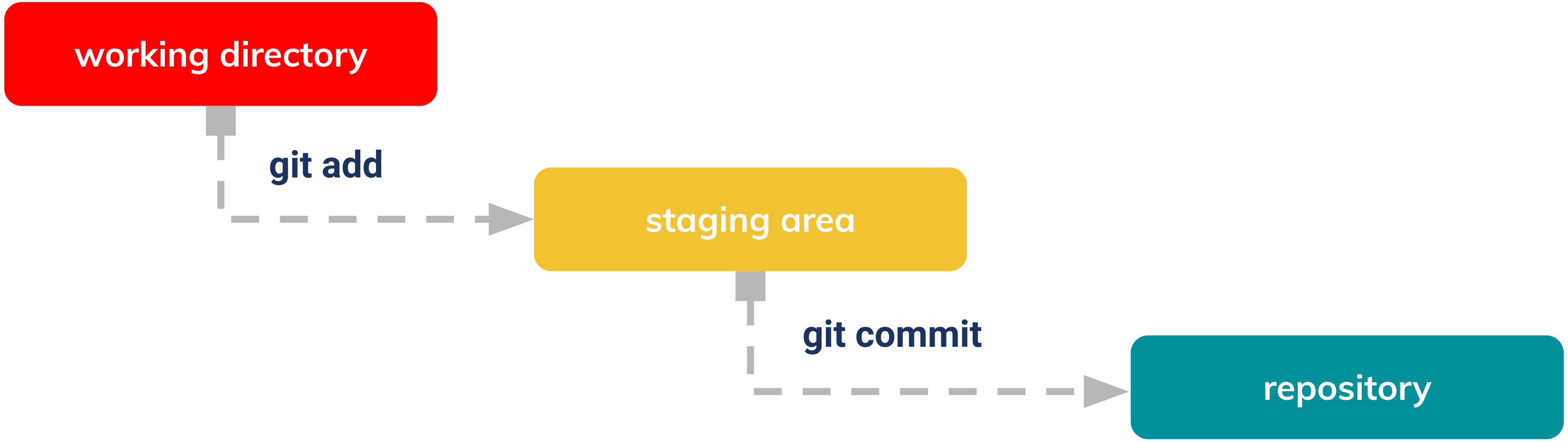
output

```
# git config
$ git config --global user.name “John Done”
$ git config --global user.email “johndoe@email.com”

# start with init
$ git init
$ git remote add <remote_name> <remote_repo_url>
$ git push -u <remote_name> <local_branch_name>

# start with existing project, start working on the project
$ git clone ssh://john@example.com/path/to/my-project.git
$ cd my-project
```

alterra
academy

SAVING CHANGES

THE STAGING AREA



alterra academy

GIT STATUS, ADD, COMMIT

output

```
$ git status

$ git add <directory>
$ git add hello.py
$ git add .

$ git commit -m “add config file”
```

COMMIT MESSAGE

"If applied, this commit will *your subject line here*"

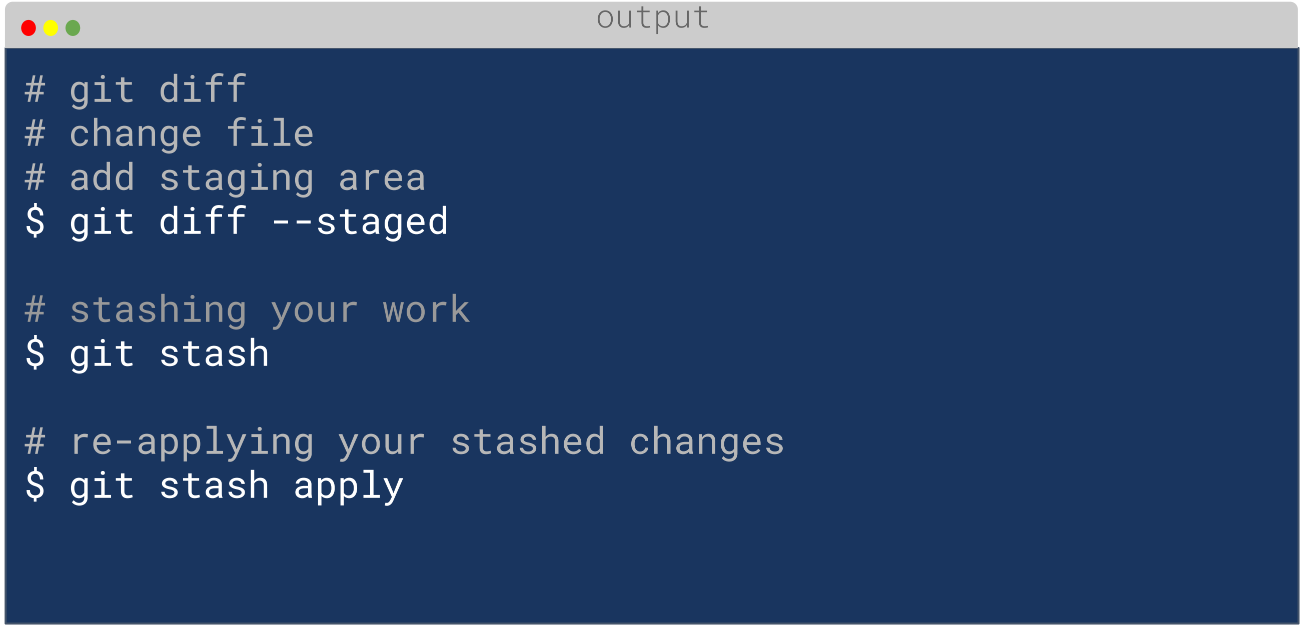
For example:

- If applied, this commit will *refactor subsystem X for readability*
- If applied, this commit will *update getting started documentation*
- If applied, this commit will *remove deprecated methods*
- If applied, this commit will *release version 1.0.0*

Notice how this doesn't work for the other non-imperative forms:

- If applied, this commit will *fixed bug with Y*
- If applied, this commit will *changing behavior of X*
- If applied, this commit will *more fixes for broken stuff*
- If applied, this commit will *sweet new API methods*

GIT DIFF AND STASH


output

```
# git diff
# change file
# add staging area
$ git diff --staged

# stashing your work
$ git stash

# re-applying your stashed changes
$ git stash apply
```

FILE .gitignore

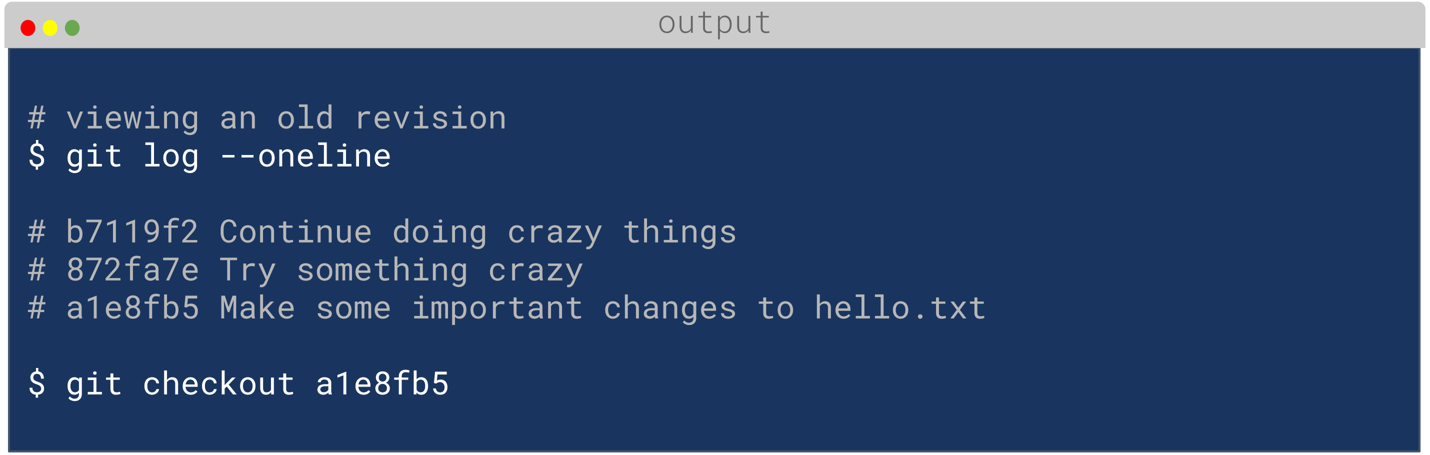
Pattern	Example matches	Explanation*

`*.log`	`Debug.log`
	`foo.log`
	`.log`
	`logs/debug.log`
		An asterisk is a wildcard that matches zero or more characters.

`logs`	`logs`
	`logs/debug.log`
	`logs/latest/foo.bar`
	`build/logs`
	`build/logs/debug.log`
		If you don't append a slash, the pattern will match both files and the contents of directories with that name. In the example matches on the left, both directories and files named *logs* are ignored

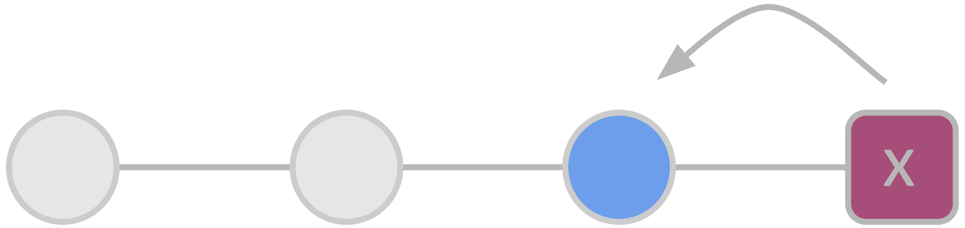
INSPECTING REPOSITORY

GIT LOG, CHECKOUT


output

```
# viewing an old revision
$ git log --oneline

# b7119f2 Continue doing crazy things
# 872fa7e Try something crazy
# a1e8fb5 Make some important changes to hello.txt

$ git checkout a1e8fb5
```

GIT RESET



--soft	--hard
uncommit changes, changes are left staged (index).	uncommit + unstage + delete changes, nothing left.

GIT RESET

output

```
# viewing an old revision
$ git log --oneline

# b7119f2 Continue doing crazy things
# 872fa7e Try something crazy
# a1e8fb5 Make some important changes to hello.txt

$ git reset a1e8fb5 --soft
```

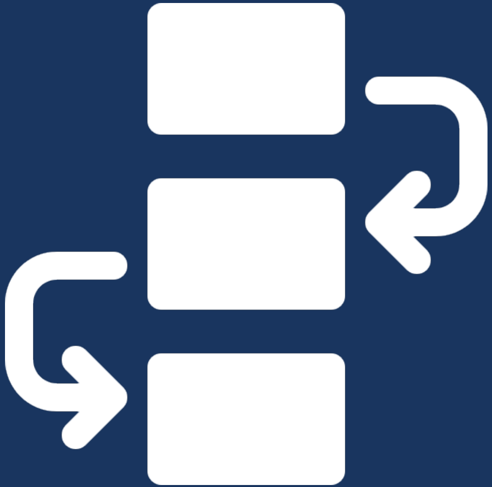
SYNCING

GIT PUSH, FETCH & PULL

output

```
# git remote
$ git remote -v
$ git remote add origin http://dev.example.com/john.git

# fetch and pull
$ git fetch
$ git pull origin master

# push
$ git push origin master
$ git push origin feature/login-user
```

alterra academy



BRACHES

GIT BRANCHING

output

```
# show all branch
$ git branch --list

# create a new branch called <branch>
$ git branch <branch>

# force delete the specified branch
$ git branch -D <branch>

# list remote branch
$ git branch -a
```

GIT MERGE

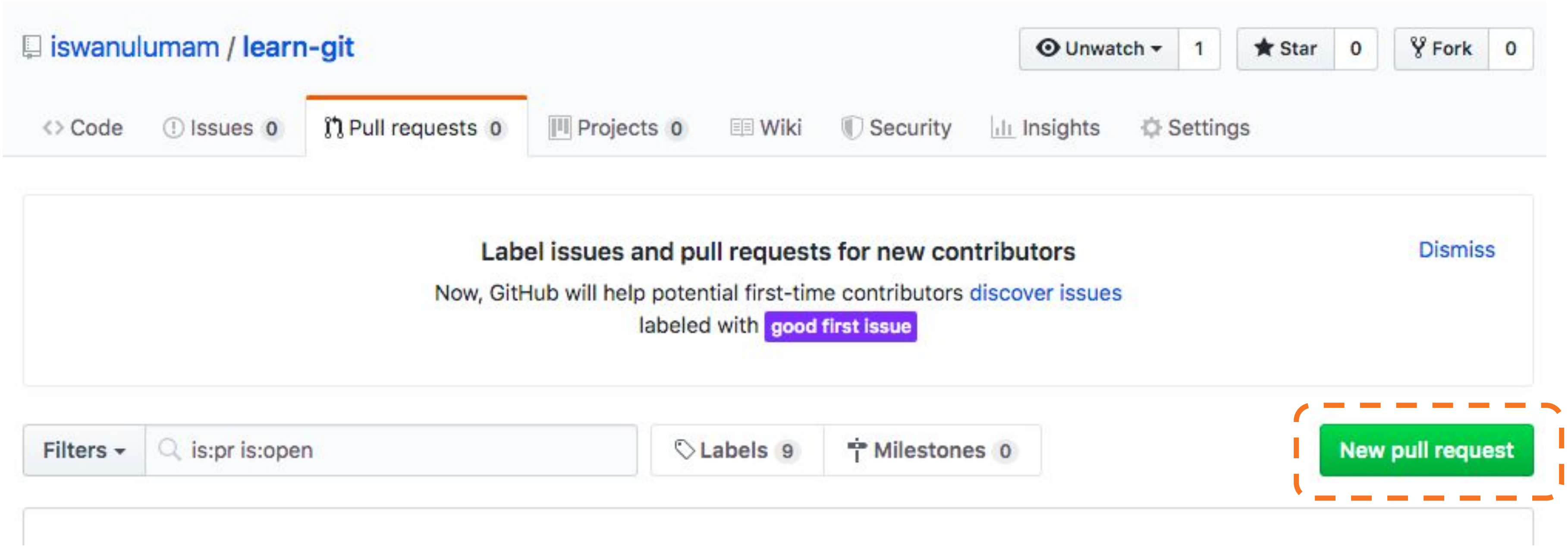
output

```
# Start a new feature
$ git checkout -b new-feature master
# Edit some files
$ git add <file>
$ git commit -m "Start a feature"
# Edit some files
$ git add <file>
$ git commit -m "Finish a feature"
# Merge in the new-feature branch
$ git checkout master
$ git merge new-feature
$ git branch -d new-feature
```

alterra academy

PULL REQUEST

PULL REQUEST


iswanulumam / learn-git

Unwatch 1 | Star 0 | Fork 0

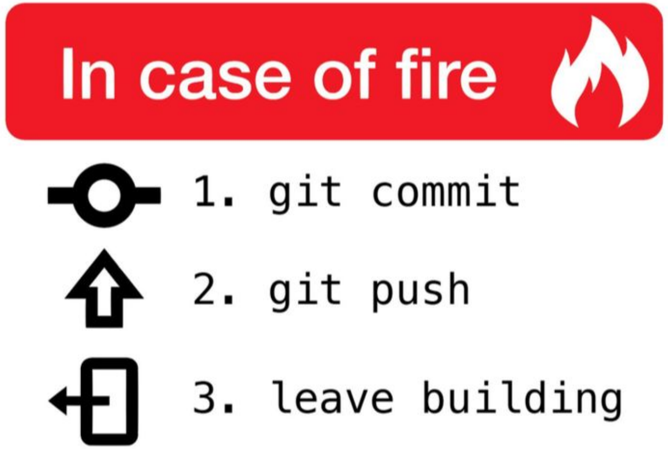
Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Security | Insights | Settings

Label issues and pull requests for new contributors

Dismiss

Now, GitHub will help potential first-time contributors discover issues labeled with **good first issue**

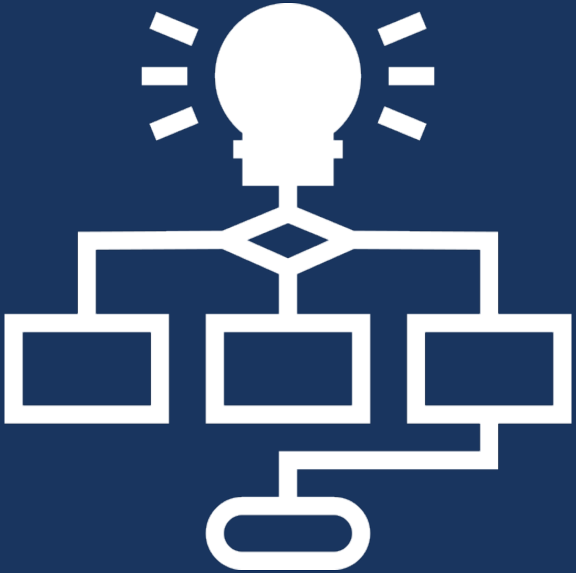
Filters | is:pr is:open | Labels 9 | Milestones 0 | New pull request



In case of fire

1. git commit
2. git push
3. leave building

alterra academy

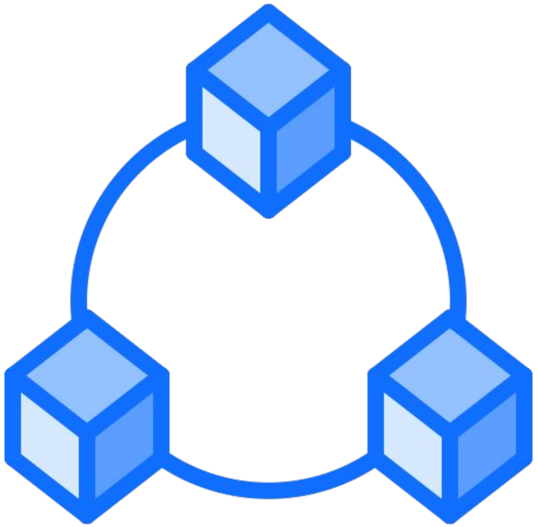


WORKFLOW COLLABORATION

Objectives

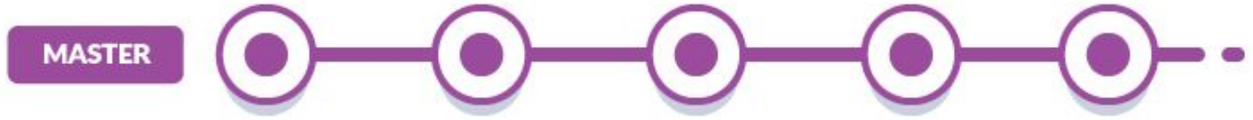
Understand the proper workflow to collaborate with GitHub or Gitlab

alterra academy

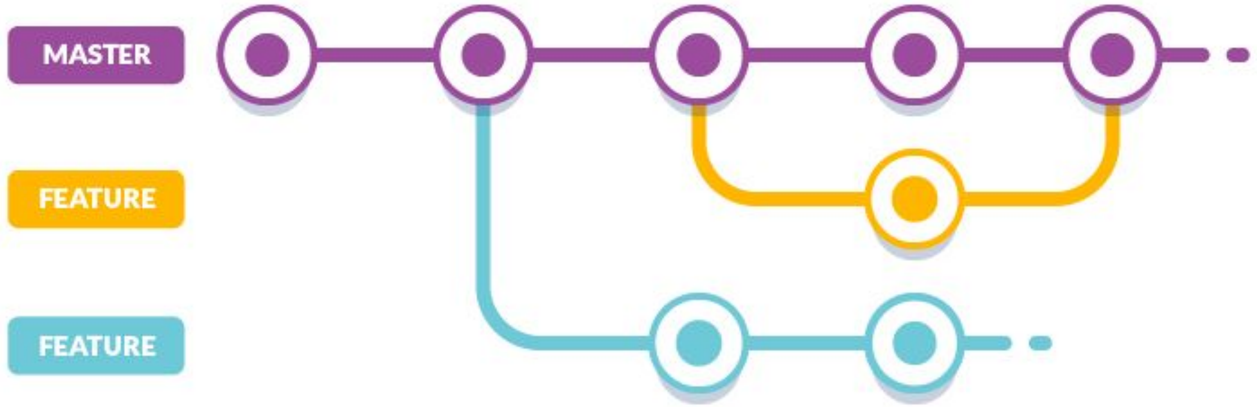


How to optimize your workflow with GitHub or Gitlab?

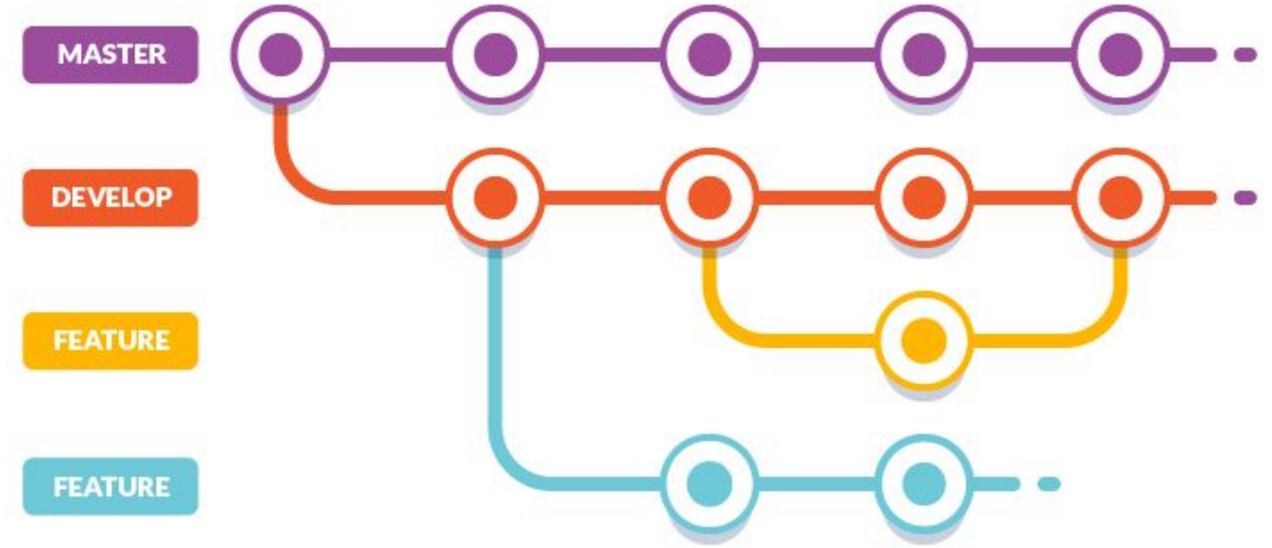
DO YOU WORK LIKE THIS ?



OR LIKE THIS ?

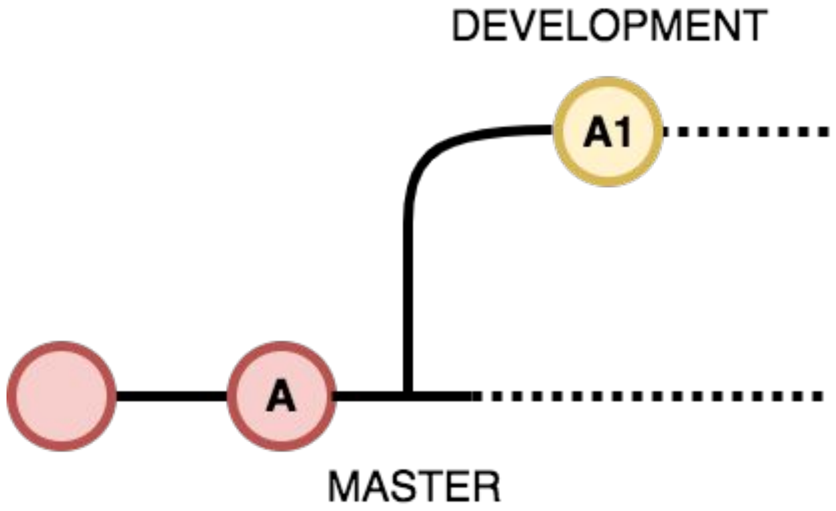


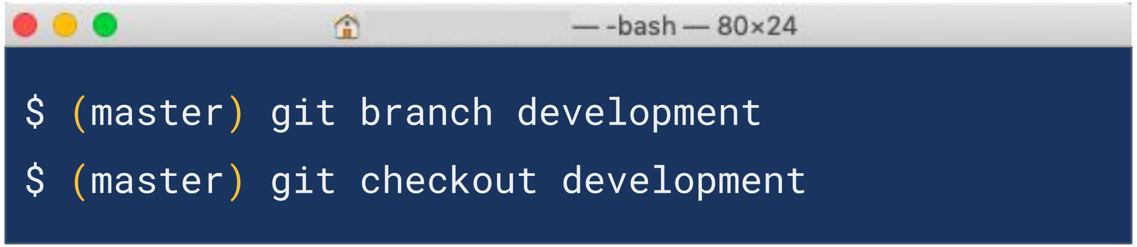
THE BEST WAY LIKE THIS



1

LET THE MASTER BRANCH UNDISTURBED



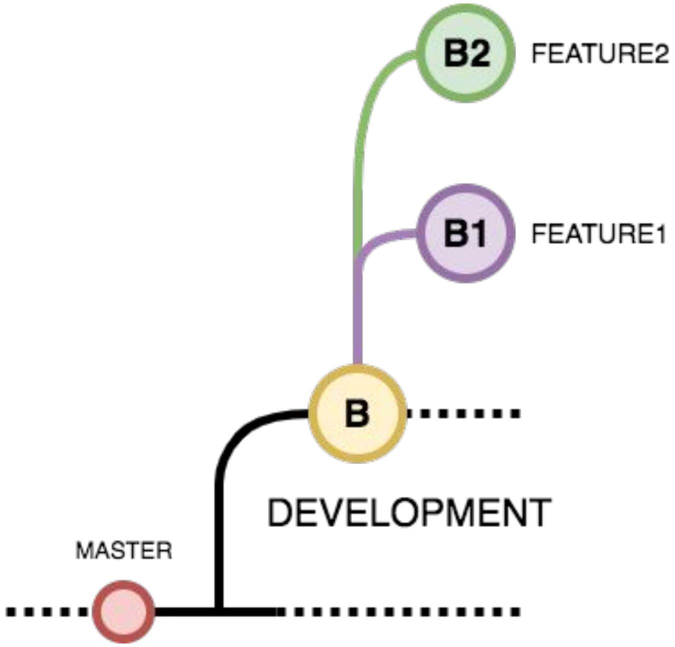


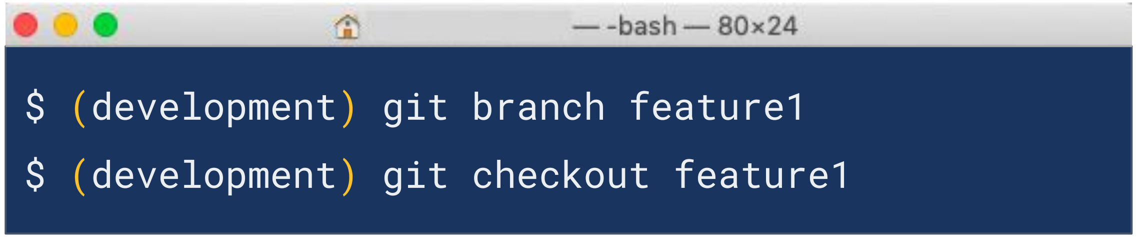
```
$ (master) git branch development
$ (master) git checkout development
```

alterra academy

2

AVOID DIRECT EDIT ON DEVELOPMENT



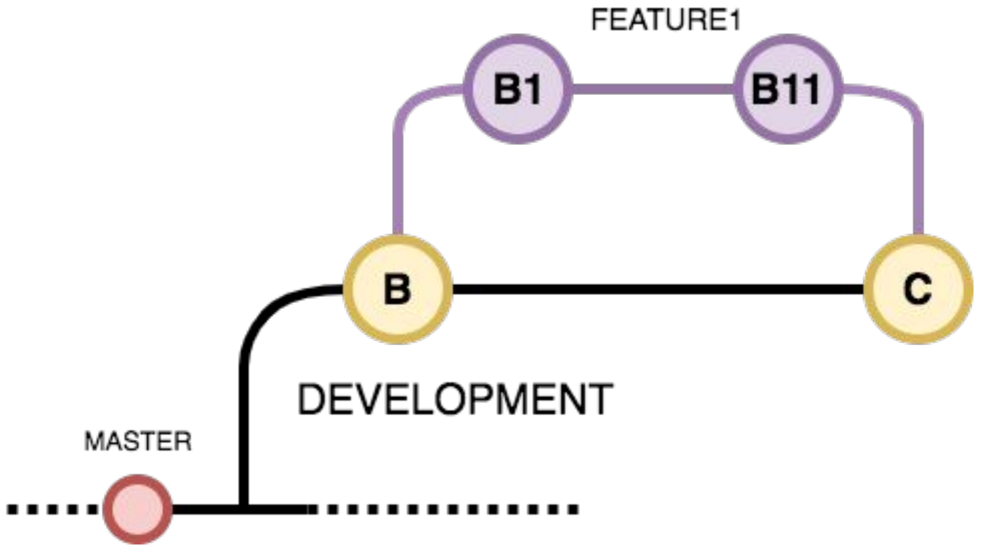


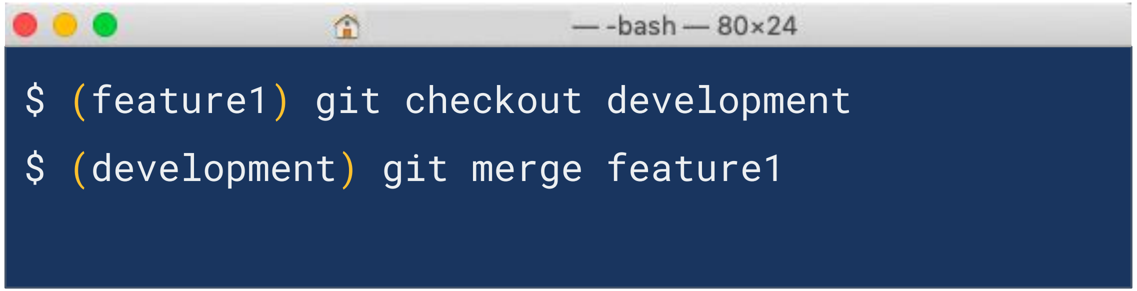
```
$ (development) git branch feature1
$ (development) git checkout feature1
```

alterra academy

3

APPLY THE FEATURE TO DEVELOPMENT ONLY



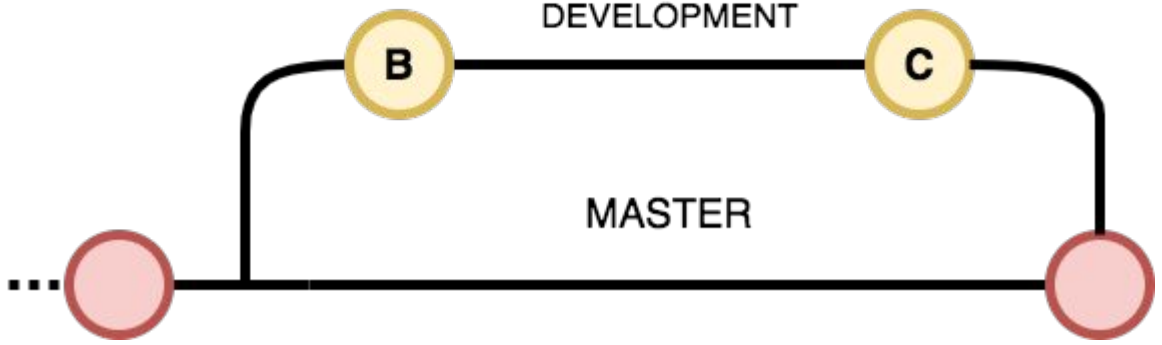


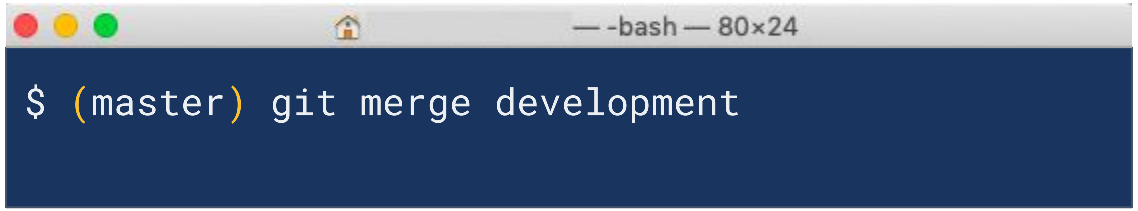
```
$ (feature1) git checkout development
$ (development) git merge feature1
```

**) git rebase will prevent you from conflicts when merging feature branch to development branch*

4

APPLY DEVELOPMENT TO MASTER WHEN IT’S DONE



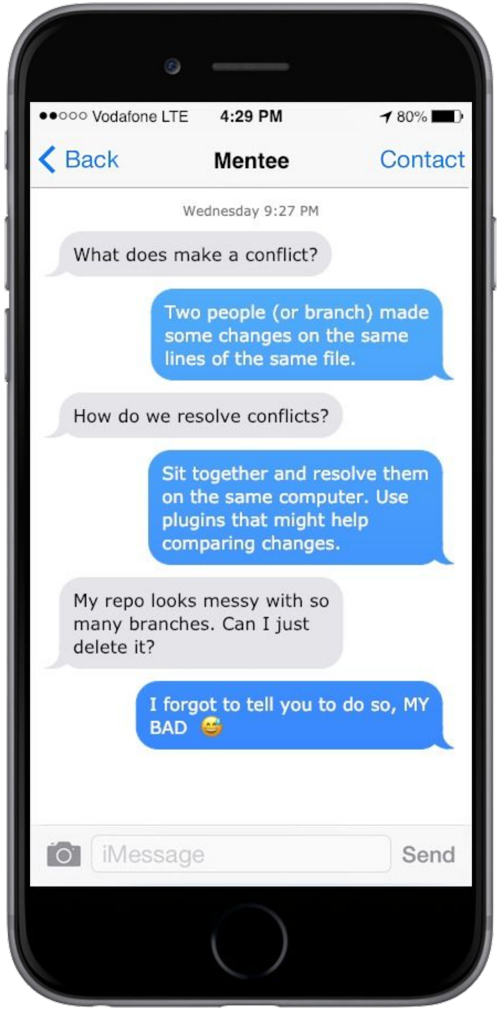


```
$ (master) git merge development
```

alterra academy

ANY QUESTION

Frequently Asked Question



Wednesday 9:27 PM

What does make a conflict?

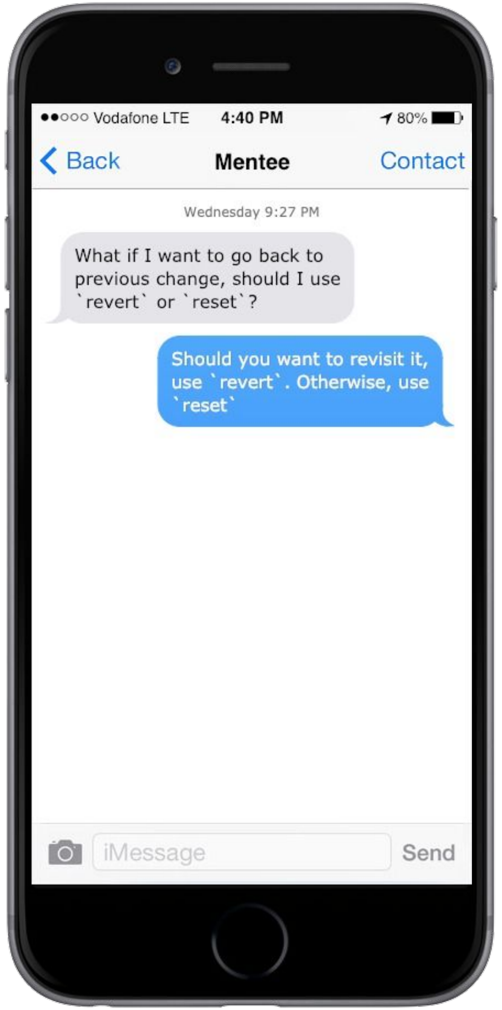
Two people (or branch) made some changes on the same lines of the same file.

How do we resolve conflicts?

Sit together and resolve them on the same computer. Use plugins that might help comparing changes.

My repo looks messy with so many branches. Can I just delete it?

I forgot to tell you to do so, MY BAD 😅



Wednesday 9:27 PM

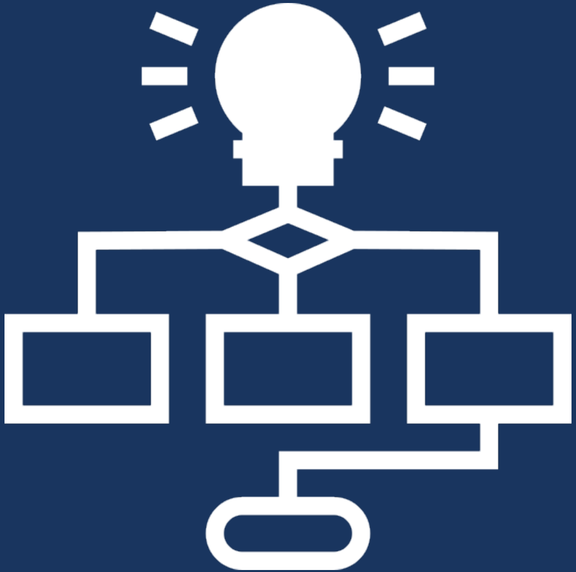
What if I want to go back to previous change, should I use `revert` or `reset`?

Should you want to revisit it, use `revert`. Otherwise, use `reset`

alterra academy

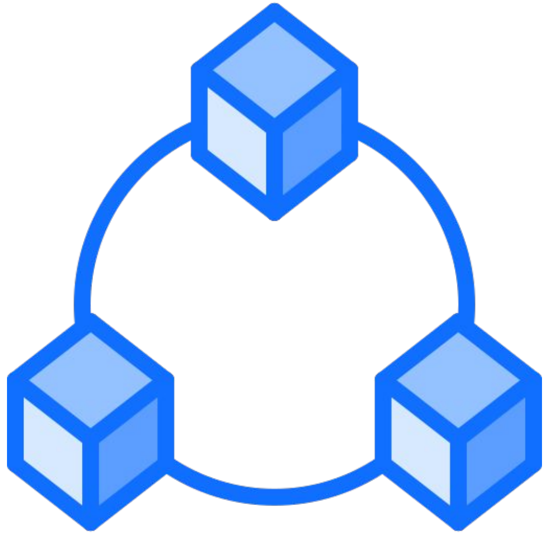
“The Best Way of Learning about Anything is by Doing”

alterra
academy

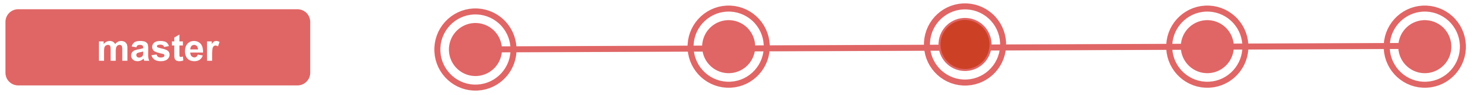


WORKFLOW COLLABORATION

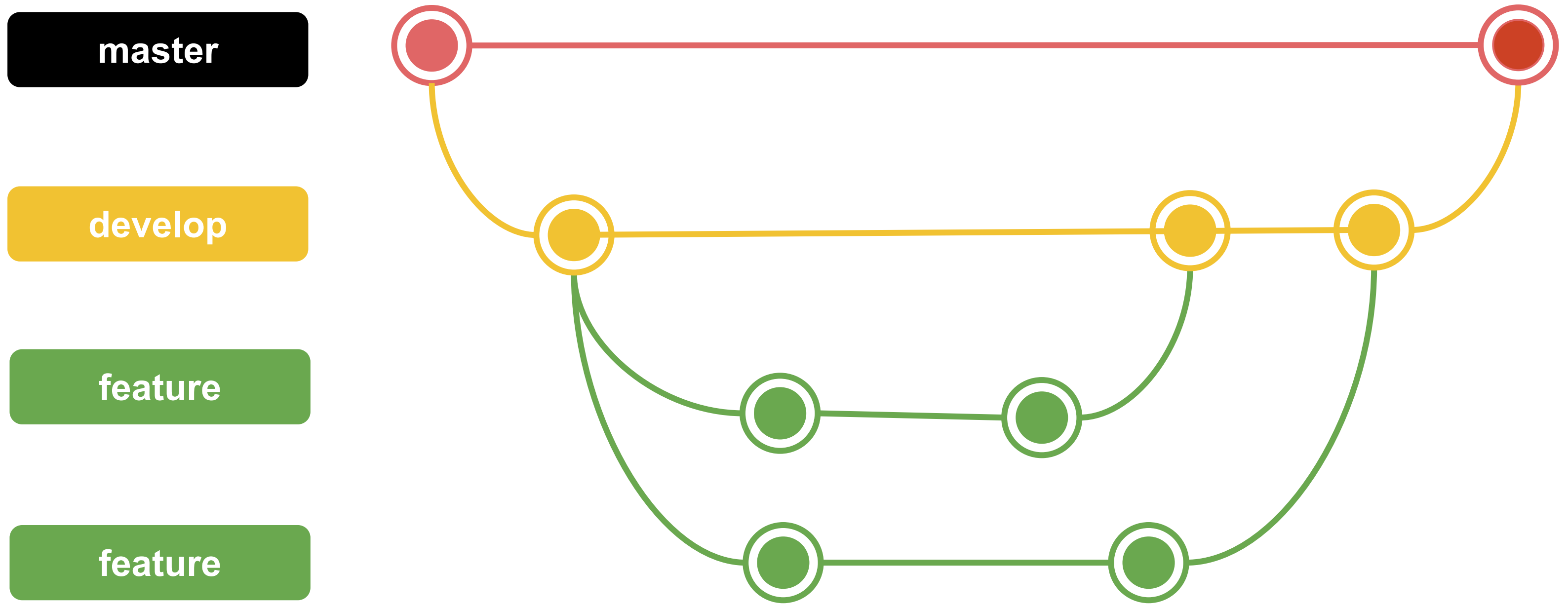
Bagaimana cara mengoptimalkan kolaborasi dalam development?



dalam kolaborasi tim kita tidak bisa hanya bekerja dalam satu branch

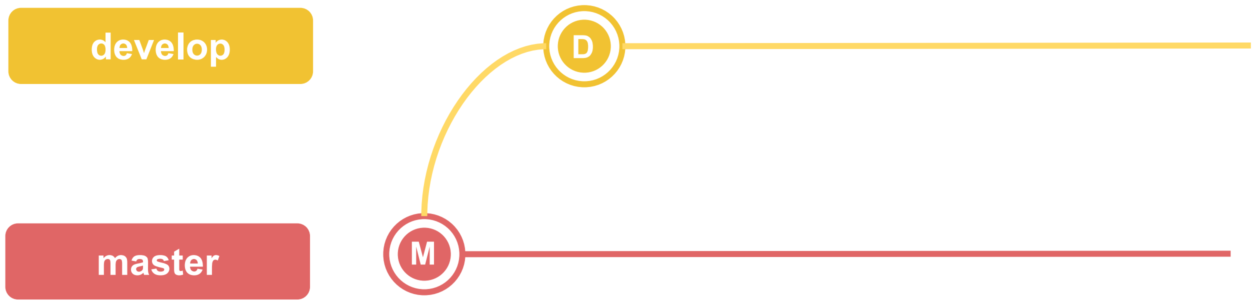


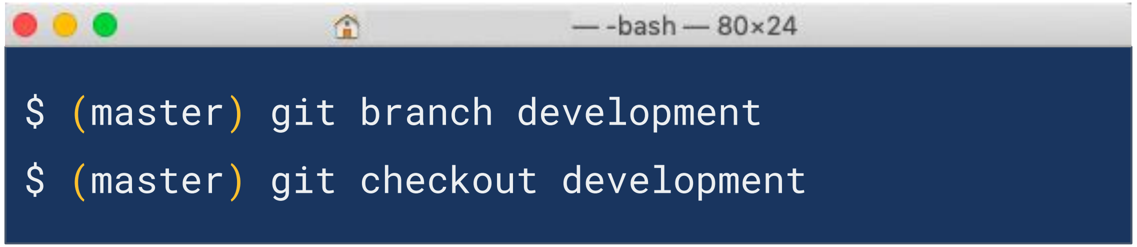
perlu dibuat beberapa branch agar kolaborasi dapat berjalan dengan optimal



1

BUAT BRANCH MASTER DARI BRANCH DEVELOPMENT



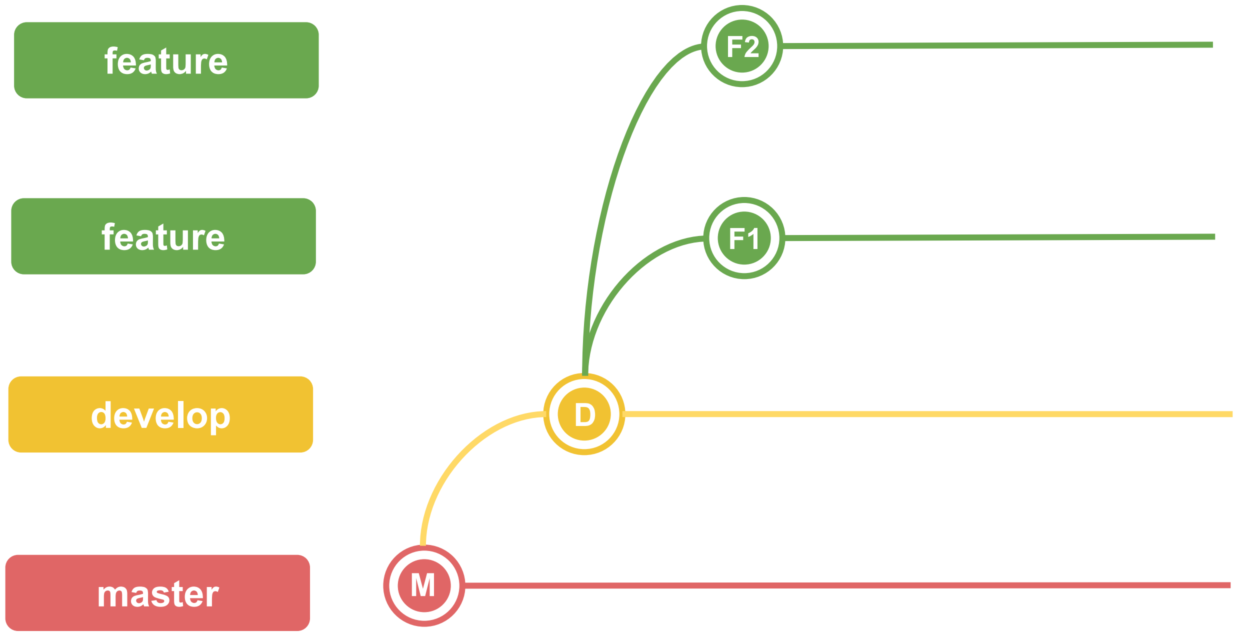


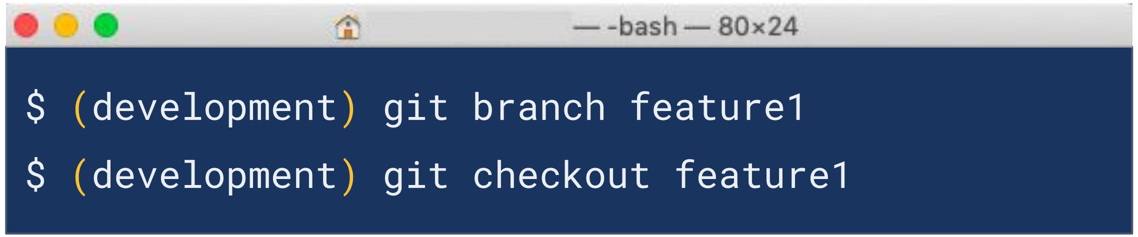
```
$ (master) git branch development
$ (master) git checkout development
```

alterra academy

2

HINDARI DIRECT EDIT KE BRANCH DEVELOPMENT

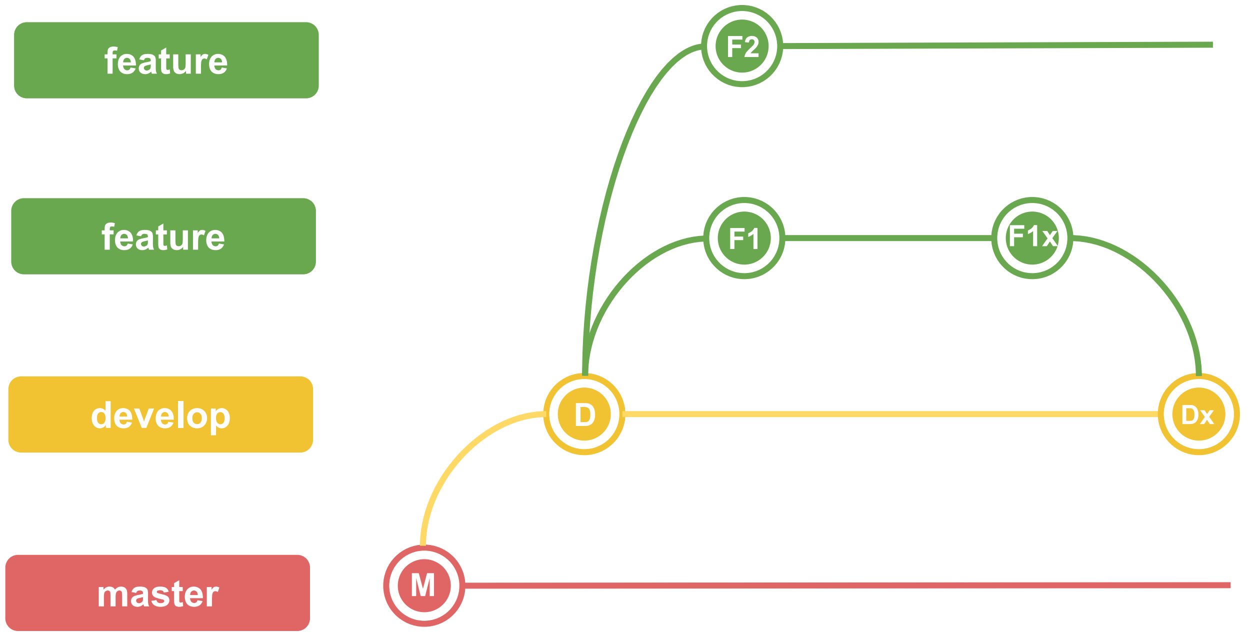


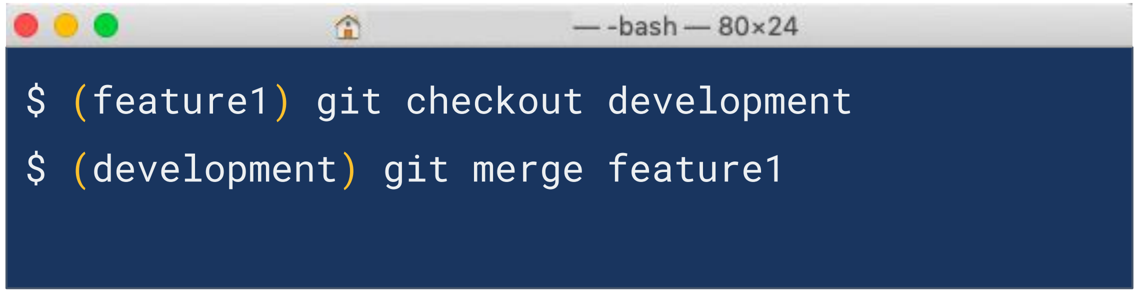


```
$ (development) git branch feature1
$ (development) git checkout feature1
```

3

MERGE BRANCH FEATURE HANYA KE BRANCH DEVELOPMENT



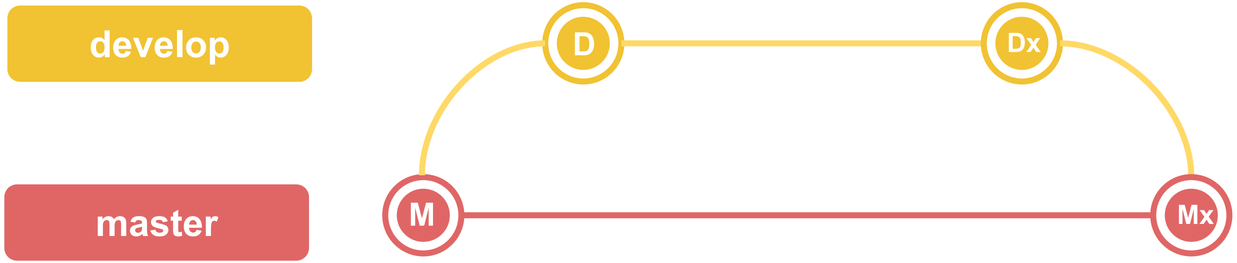


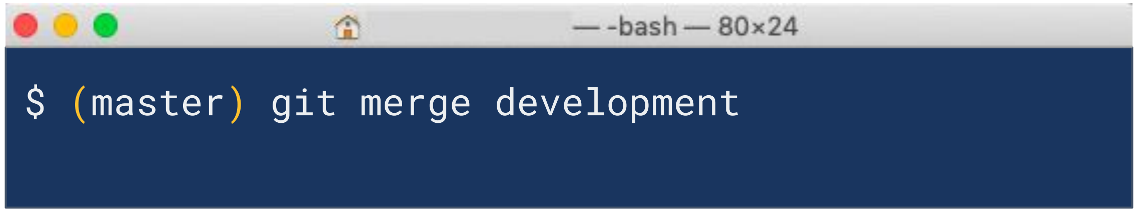
```
$ (feature1) git checkout development
$ (development) git merge feature1
```

*) *git rebase akan mencegah konflik ketika merge dari branch feature ke branch development*

4

MERGE BRANCH DEVELOPMENT KE BRANCH MASTER JIKA DEVELOPMENT SELESAI





```
$ (master) git merge development
```

alterra academy

FAQ

Apa yang membuat git conflict?
Ketika ada dua orang yang membuat perubahan di file dan baris yang sama

Bagaimana cara resolve git conflict?
Resolve dengan satu computer, gunakan plugin untuk membandingkan perubahan

Jika terlalu banyak branch, apakah branch boleh dihapus?
Ya tentu boleh, hapus saja branch yang sudah lama.

alterra
academy

“The Best Way of Learning about Anything is by Doing”

Tugas

1. Buat sebuah repository di Github
2. Implementasikan penggunaan branching yang terdiri dari master, development, featureA, dan featureB
3. Implementasikan intruksi git untuk push, pull, stash dan merge
4. Implementasikan sebuah penanganan **conflict** di branch developement ketika setelah merge dari branch featureA lalu merge dari branch featureB (Conflict bisa terjadi jika kedua branch mengerjakan di file dan line code yang sama)
5. Gunakan merge no fast forward
6. Kirimkan alamat repository github di :

Any Question

?