

Version Control

Silver - Chapter 1 - Topic 7

Selamat datang di **Chapter 1 Topic 7**
online course **Back End Java** dari
Binar Academy!

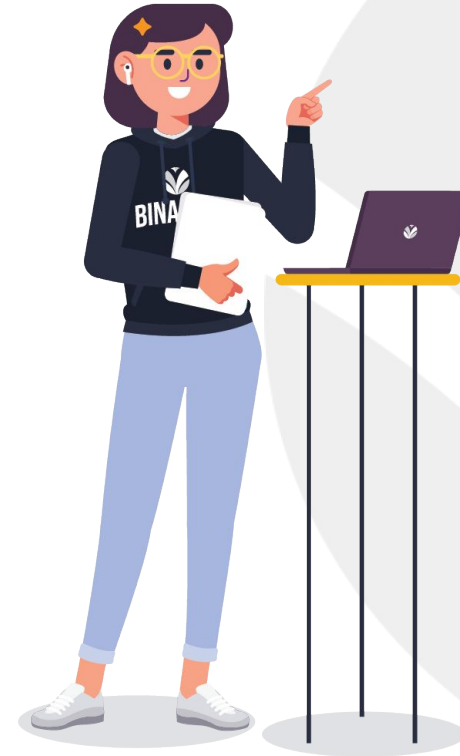


Holaaa!!

Pada topik sebelumnya, kita udah belajar konsep tentang Clean Code.

Pada topik kelima alias terakhir ini, kita bakal mengelaborasi tentang **Version Control**. Mulai dari konsep Version Control, GIT dan macam-macam jenis Version Control lain, konsep GIT sampai dengan melakukan set up Git

Yuk kita intip dulu peta materinya~



Detailnya, kita bakal bahas hal-hal berikut ini:

- Konsep Version Control
- GIT dan macam-macam Version Control lain
- Set up GIT

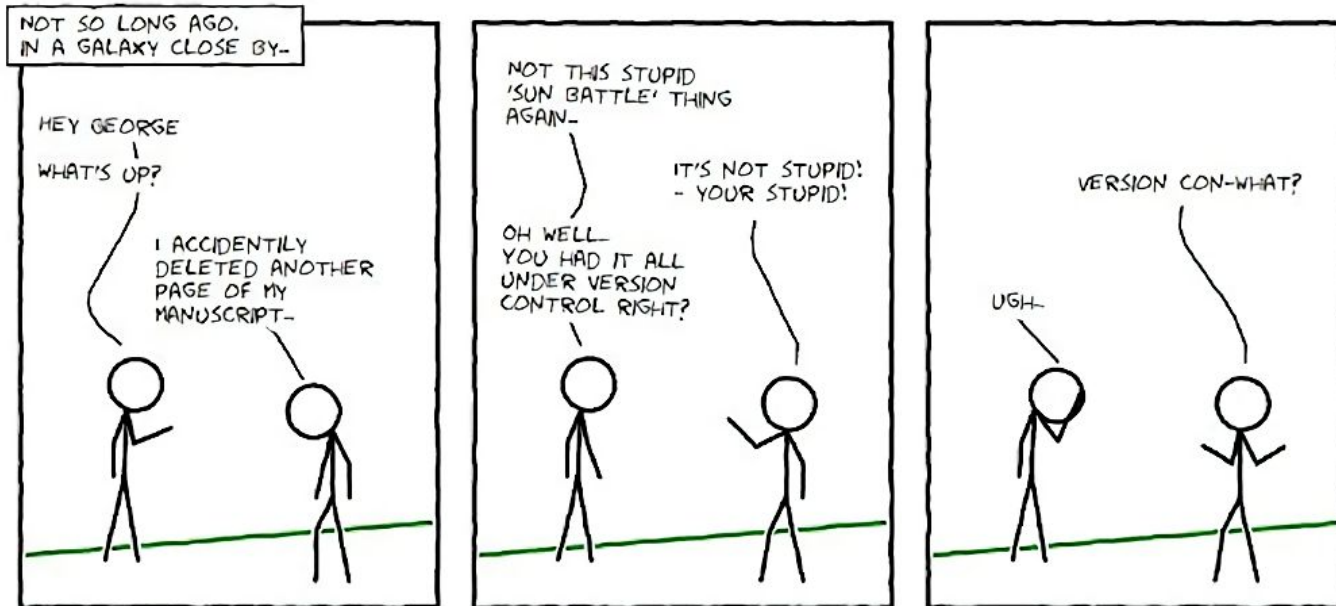


Buat nemenin kegiatan kamu hari ini,
materi pertama bakal ngenalin kamu
sama **Version Control**.

Yuk kita cari tau apa aja isinya~



Sambil pemanasan, kita coba perhatikan gambar di bawah, ya!



Penjelasan gambarnya, nih.

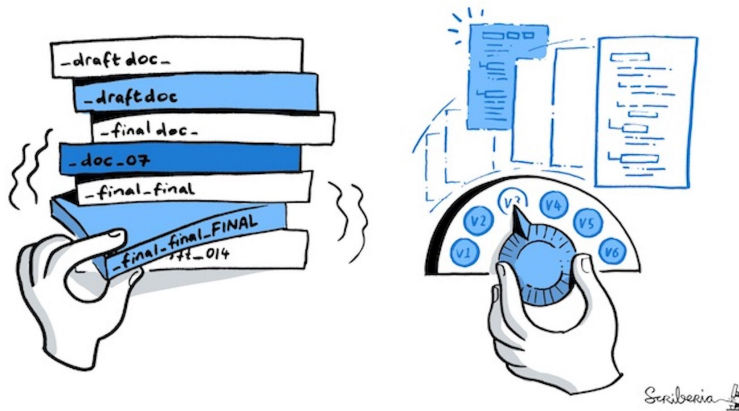
Gambar sebelumnya menceritakan tentang seorang developer yang nggak sengaja menghapus codenya.

Sebenarnya tuh hal kayak begini bukan menjadi permasalahan yang besar, **kalau** developer tersebut menuliskan codenya di version control.

Sayangnya, doi nggak melakukannya. Jadi hilanglah semua code yang dia tulis dengan susah payah 😞



TRACK PROJECT HISTORY



Dapet spoiler dikit, sebetulnya Version Control itu apa sih?

Version Control adalah sistem yang merekam semua perubahan dari suatu file ataupun sekumpulan file dalam suatu waktu.

Dari situ, perubahan sekecil apapun bakal diakumulasi menjadi sebuah version.

Bahkan sampai perubahan spasi sekalipun dicatat baik-baik sama system ini. Gokil!

Dengan direkamnya setiap perubahan yang sudah kamu buat, kamu bisa melihat dan mengembalikan file ke yang sebelumnya.

Bahkan bisa juga membandingkan file kodingan lama dengan file kodingan yang baru.



Version control ini dirancang supaya **para engineer bisa mengurangi resiko kegagalan di data-data yang udah dimodifikasi.**

Mirip-mirip deh kayak kamu yang lagi curiga sama pacar. Semua dicari tau, hihi~



Karena udah kenalan sama Version Control, selanjutnya kita bakal terjun lebih dalam tentang **GIT dan Version Control lain.**

Udah siap? Yuk kita mulai~

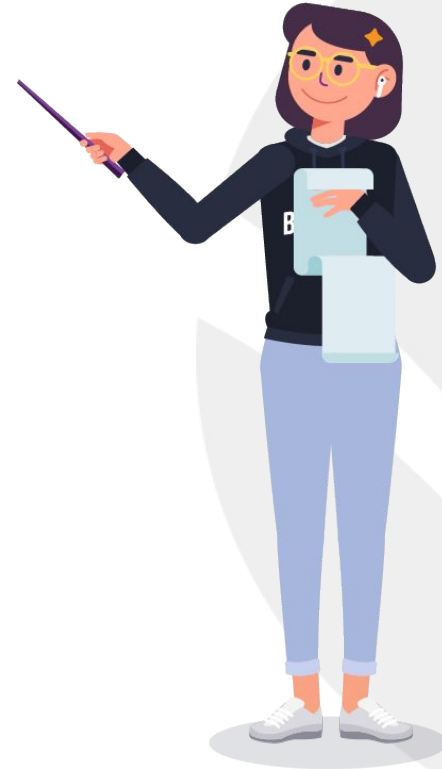


Bukan cuma kendaraan aja yang punya beberapa tipe, Version Control juga sama, lho!

Version control punya tiga tipe, Gengs.

1. **Local Version Control System**
2. **Centralized Version Control System**
3. **Distributed Version Control System**

Penjelasan lebih lanjut, mari kita bahas pada halaman berikutnya!



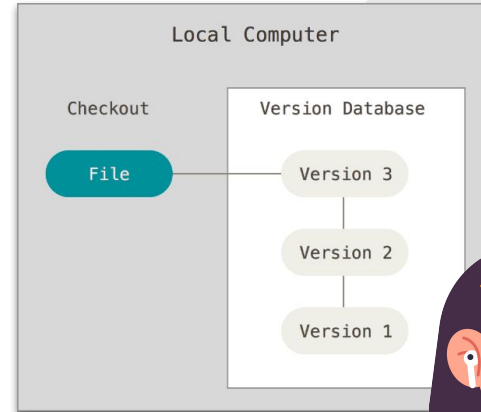
1. Local Version Control

Tipe version control yang pertama adalah Local Version.

Local Version Control disebut juga sebagai Local VCS. Cara kerjanya yaitu dengan meng-copy file ke directory lain berdasarkan timestamp atau penanda waktunya.

“Contohnya gimana?”

Contoh toolnya yaitu berupa RCS (Revision Control System)

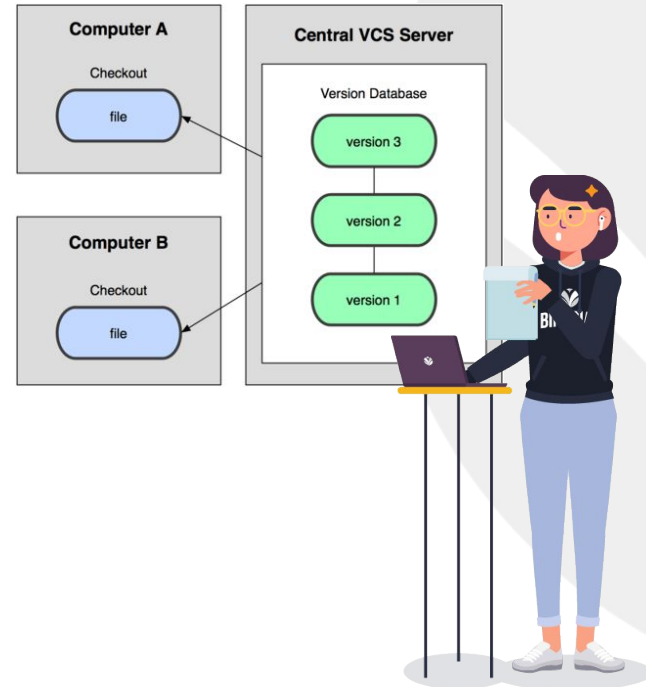


2. Centralized Version Control

Lanjut, ya~

Tipe version control yang kedua adalah Centralized Version Control atau disebut juga sebagai CVCS. Cara kerjanya adalah dengan berkolaborasi dengan developer di system lain.

Contoh tools-nya adalah CVS, SVN dan Perforce.

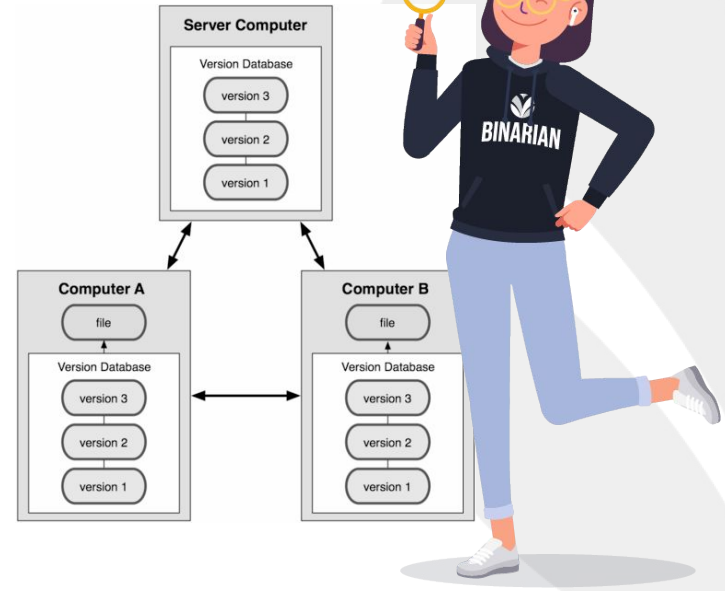


3. Distributed Version Control

Okey, tipe version control yang ketiga adalah Distributed Version Control atau disebut juga sebagai DVCS.

DVCS mengandung banyak repository. Sistem ini mendukung cara kerja yang membuat pengguna punya repository dan copy dari pekerjaan mereka sendiri.

Contoh toolnya adalah Perforce Git, Mercurial, Bazaar dan Darcs.



Setelah membahas tipe Version Control, kita lanjut nih buat bahas tentang Git!

Apa itu Git?

Git adalah version control system yang gratis dan bersifat open source (source code atau kode sumbernya tersedia bebas dan dapat diakses atau dimodifikasi oleh siapa pun).



Git dirancang untuk menangani berbagai project, mulai dari project kecil sampai project yang sangaattttt besar dengan kecepatan dan efisiensi yang Git miliki.

Mantul, kan?



Nggak pas rasanya kalau nggak bahas kelebihanannya juga~

“Lalu, apa sih kelebihan dari Git?”

- Git mudah dipelajari dan punya footprint kecil dengan kinerja secepat kilat. Wushh~
- Jika dibandingkan sama version control system lain seperti Subversion, CVS, Perforce dan ClearCase, Git lebih unggul. Mantap!
- Git juga punya fitur seperti **local branching** yang murah, **staging area** yang nyaman serta punya multiple workflow. Keren!



PERFORCE





Kamu pernah dengar istilah Github?

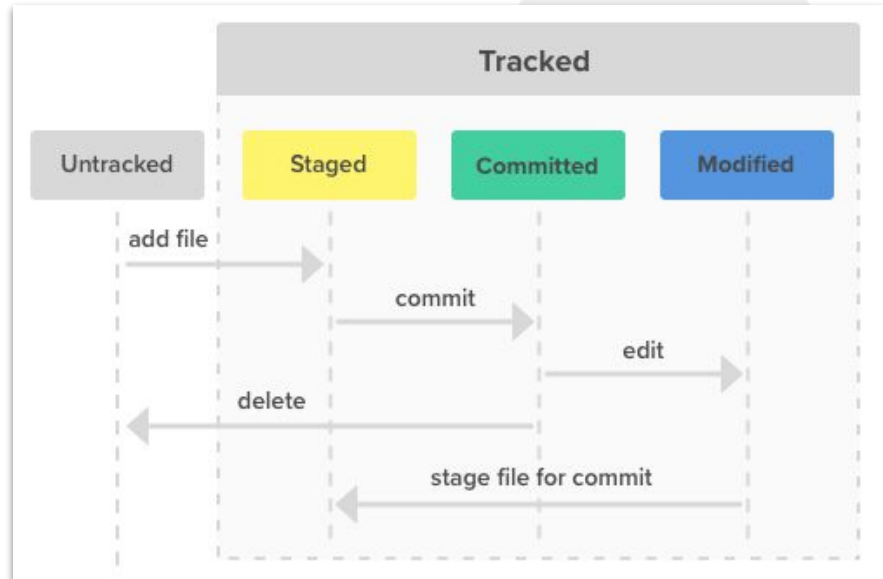
Nah, agak mirip ya penyebutannya dengan Git. Tapi, Git itu bukan Github walaupun kata depannya sama. Yes, betul mereka berbeda.

- Git adalah Version Control System,
- Sedangkan **GitHub adalah collaboration tool.**



Ada tiga status file pada Git, apa aja ya?

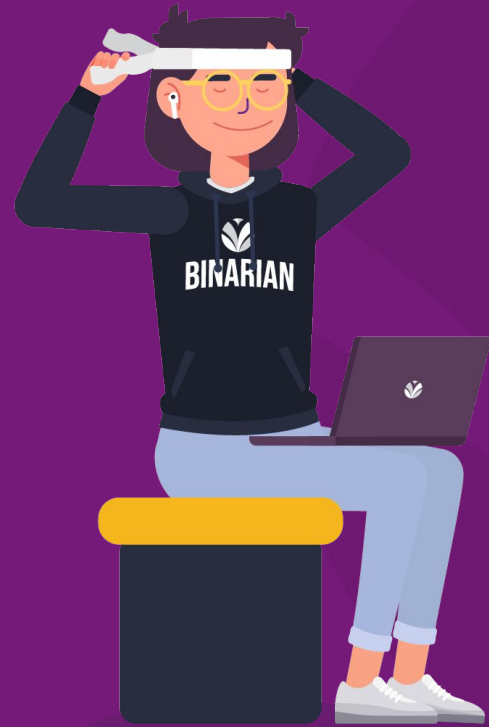
1. **Modified**, yaitu kamu udah melakukan perubahan file yang bisa di track sama Git, tapi perubahan tersebut belum tersimpan.
2. **Staged**, yaitu file yang sebelumnya bersifat untracked berubah jadi file yang bisa ditrack sama Git, biar selanjutnya bisa langsung di commit deh.
3. **Committed**, yaitu file udah disimpan dengan aman di local repository.



Kalau koki lagi mau masak, hal yang perlu disiapkan adalah bahan-bahannya, kan?

Nah, kalau Back End Engineer, salah satu yang perlu disiapkan adalah Git.

Oleh karena itu, selamat **Menyiapkan Git.**



Masak aja ada caranya, masa belajar Git enggak?!

Kalau belajar Git, cara terbaiknya adalah dengan menggunakan command line.

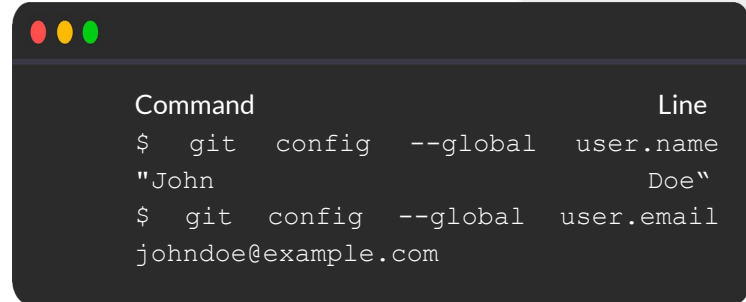
Dalam real project, kita bisa pake berbagai tools.

Makanya, jangan lupa Install [Git Bash](#) buat belajar Git ya!



Untuk melakukan setup Git, ikuti instruksi berikut ya!

1. Setup identitas kamu
2. Selalu lakukan kayak gambar disamping ini saat setup Git client untuk pertama kali. Don't forget, Gan!



```
Command                                     Line
$ git config --global user.name           "John"                                     Doe"
$ git config --global user.email          johndoe@example.com
```

Command clone ini bakal melakukan duplikasi di local dari remote repository.

Command:

```
$ git clone [RepositoryLink]
```

e.g

```
$ git clone http://gitlab.binar.com/student.git
```

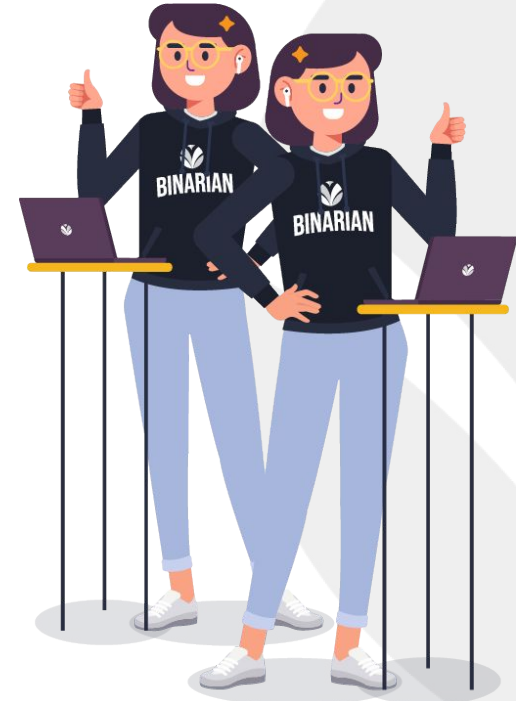
Dengan command di atas, folder student bakal dibuat dan bakal jadi working directory.

Terusss..

Setelah melakukan Git Clone, kita bakal punya repository sendiri pada local kita. Semua operations seperti commit, branching semua dilakukan secara local.

Ingat, ya. **Git merupakan Distributed Version Control System.**

Jadi, cuma bisa dilakukan satu kali saat di awal bikin local repository.

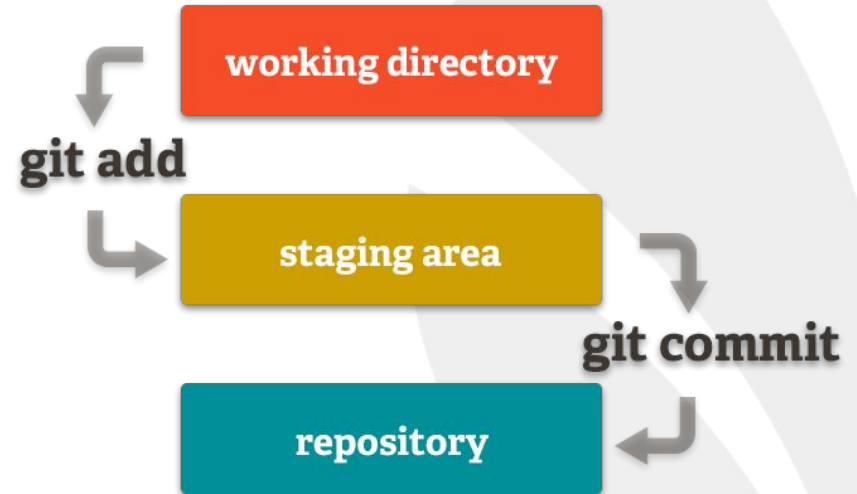


Kita juga bisa ngelakuin beberapa hal, lho!

Karena merupakan Distributed Version Control System, kita bisa ngelakuin berbagai operations di local seperti commit, stash, branch, view history, dll.

Commit untuk menambahkan update file dan komentar. Jadi, setiap kontributor bisa memberi konfirmasi update file di proyek yang lagi dikerjakan.

Keren banget, kan?



Apa aja sih perintah yang ada di Git?

- Git add – menambahkan file dari working ke staged state.
`$ git add MerlinPOS/Forms/frmProducts.vb`
- Git reset – Reset file dari Staged ke Working state.
`$ git reset HEAD
MerlinPOS/Forms/frmProducts.vb`
- Git checkout – menghapus changes pada file yang ada di Working state atau switch branches.



Ada yang lainnya juga nih, bosque~

- Menghapus changes pada file yang ada di Working state.

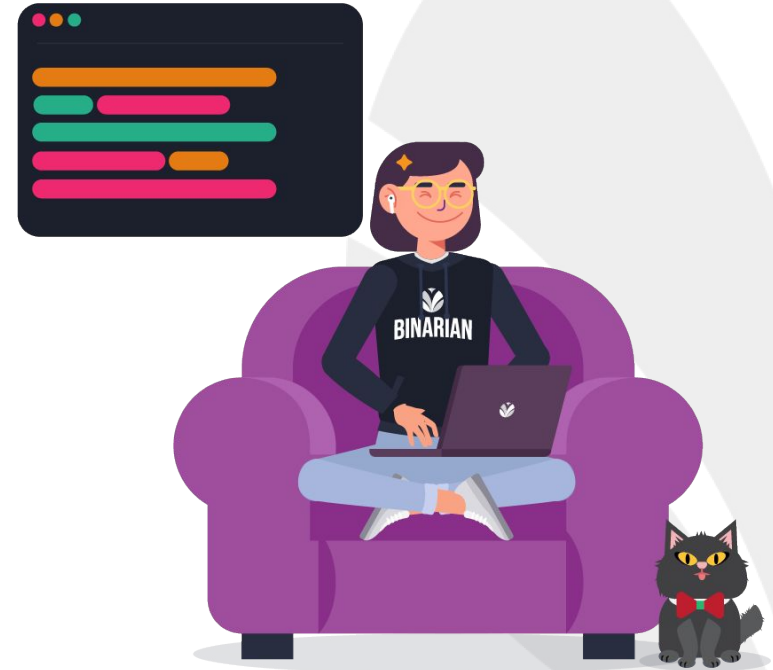
```
$ git checkout  
MerlinPOS/Forms/frmProducts.vb
```

- Switch branches.

```
$ git checkout master
```

- Melakukan commit terhadap file yang berada pada staged state.

```
$ git commit -m "your message is here"
```



Di mana saja, kapan saja, Commit selalu bisa~

Kayak yang udah disebutin sebelumnya, kalau kita melakukan commit tanpa melakukan push, kita cuma bisa melakukan commit di local kita aja.

Padahal kita juga bisa melakukan commit kapan saja di local tanpa orang lain mendapat perubahannya. Iho.

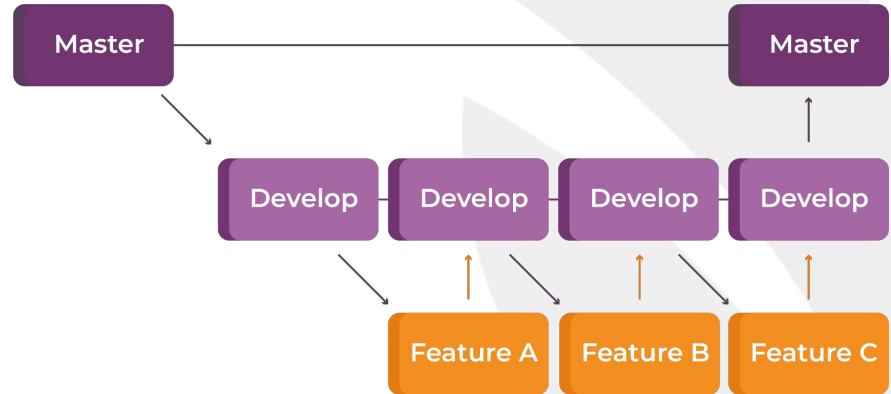
Biar orang lain bisa cek perubahannya, kamu harus melakukan push terlebih dahulu. Tapi ingat ya, jangan pernah melakukan commit di master branch. Nggak boleh, pokoknya!



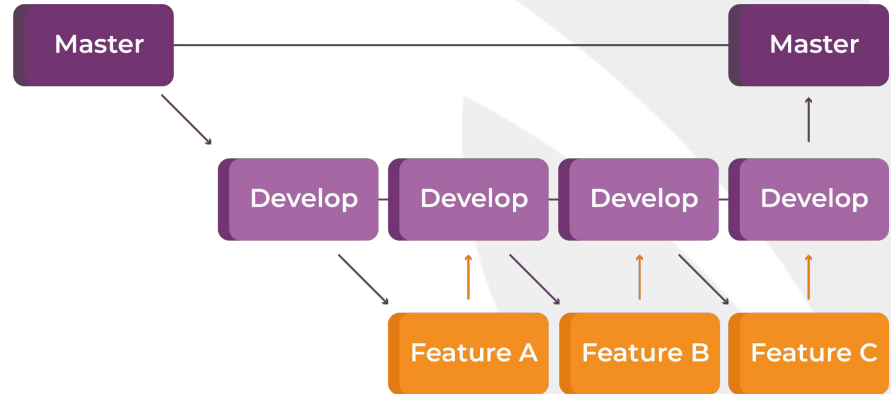
Dari tadi nyebut branch, sebetulnya branch itu apa sih?

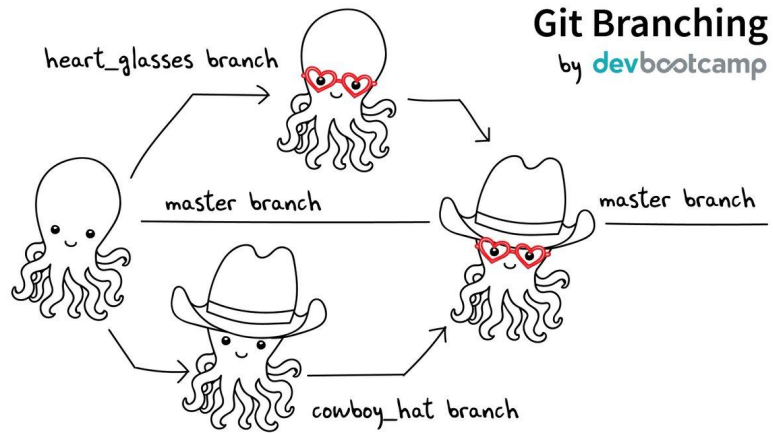
Wah, sabar-sabar, kita bahas pelan-pelan~

Branch merupakan bagian dari main repository master branch. Ketika kamu berkolaborasi dengan banyak orang, kamu harus membuat branch Git kamu sendiri.



Lho, emangnya kenapa? Agar developer lain yang sedang mengerjakan fitur tertentu bebas melakukan apapun di branchnya tanpa merusak atau membuat bugs pada main branch (master) yang udah stable.





Kita pake analogi deh biar kamu makin paham

Misalnya, ada dua tim ditugaskan untuk membuat gurita menggunakan topi cowboy dan kacamata bentuk love.

Tim fitur topi cowboy dan kacamata bentuk love menuliskan code dengan algoritma yang berbeda. Tapi karena mereka membuat branch Git masing-masing, ketika akan melakukan commit, si code sumber nggak bakalan berantakan. Mantap betul!

Membuat Branch di local

```
D:\codes\merlin\Taipan_git [master]> git checkout -b new-foo-branch
Switched to a new branch 'new-foo-branch'
D:\codes\merlin\Taipan_git [new-foo-branch]> git status
# On branch new-foo-branch
nothing to commit, working directory clean
D:\codes\merlin\Taipan_git [new-foo-branch]>
```

Branch dibuat berdasarkan current branch yang ada di working dir. Bikin branch baru bukan berarti orang lain udah bisa langsung akses branch baru tersebut.

Yang harus diingat, semua itu masih di local, supaya branch-nya bisa diakses orang lain, kamu harus melakukan push terlebih dahulu. Jangan lupa!

Membuat Branch di local

```
D:\codes\merlin\Taipan_git [master]> git checkout -b new-foo-branch
Switched to a new branch 'new-foo-branch'
D:\codes\merlin\Taipan_git [new-foo-branch]> git status
# On branch new-foo-branch
nothing to commit, working directory clean
D:\codes\merlin\Taipan_git [new-foo-branch]>
```

Branching ini gampang banget di Git. Kita bisa bikin ribuan branch tanpa harus mengorbankan performance pengoperasian dari Git.

Mantulita banget kan, bund!

Kita juga bisa pindah (switch) branch di working dir.

Selanjutnya, untuk melihat current branch, dan kalau ada file dengan state working atau staged, kita bisa pakai git status.

```
D:\codes\merlin\Taipan_git [master]>
D:\codes\merlin\Taipan_git [master]> git status
# On branch master
nothing to commit, working directory clean
D:\codes\merlin\Taipan_git [master]> git checkout new-foo-branch
Switched to branch 'new-foo-branch'
D:\codes\merlin\Taipan_git [new-foo-branch]> git status
# On branch new-foo-branch
nothing to commit, working directory clean
D:\codes\merlin\Taipan_git [new-foo-branch]>
```

Tapi, kalau ada file dengan state working dan staged, kita nggak bisa pindah branch. Kalau lagi di kondisi tersebut, Git stash bisa jadi solusi yang tepat.

Git Stash yang bisa diandalkan~

Git stash menyimpan segala perubahan dengan state Working dan Staged terpisah dari working dir.

- Untuk melakukan stash, seluruh file harus dalam keadaan staged dulu, terus baru deh kita bisa melakukan command berikut ini.
`$ git stash`
- Kalau mau lihat stash yang ada, pakai ini ya
`$ git stash list`
- Kalau ini buat mengembalikan stash yang paling terakhir.
`$ git stash pop`



- Sedangkan yang ini buat lihat command yang ada untuk melakukan stash.
`$ git stash help`
- Git log dipake buat lihat history dari semua commit yang pernah dilakukan.

```
$ git log -n10
```

- Git show dipake buat menampilkan perubahan secara rinci.

```
$ git show  
3e92768e9a45c45f2c0f7ff4fcd01e5171e7ee94
```



- Git diff dipakai buat melakukan perbandingan file perubahan yang sekarang dengan file sebelum dilakukan perubahan.

1. Compare file dengan keadaan working pada unmodified (sebelum diubah).

```
$ git diff  
MerlinControls/Controls/MerlinRibbo  
nForm.vb
```

2. Compare file dengan keadaan staged pada unmodified (sebelum diubah).

```
$ git diff --staged  
MerlinControls/Controls/MerlinRibbo  
nForm.vb
```



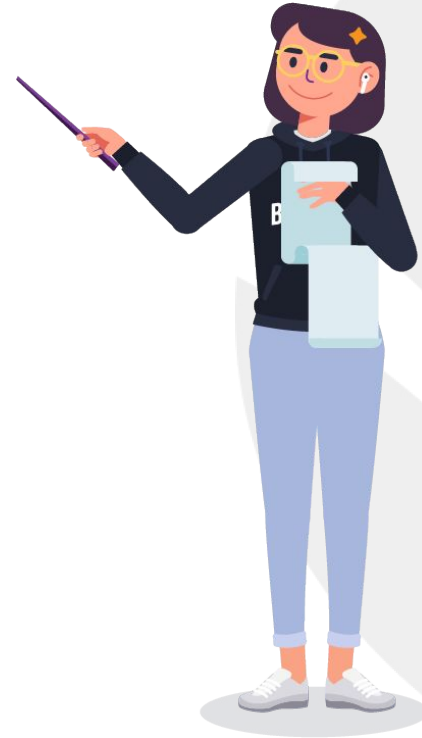
Kok ada Pull, sih?

Iya dong. Pull dipakai untuk melakukan sinkronisasi perubahan yang ada di local, supaya sesuai kayak yang ada di bagian remote.

Command-nya berupa:

- pull dari branch master
`$ git pull`
- pull dari remote branch tertentu
`$ git pull origin foo-branch`

Kita harus selalu ingat sama pull dari master dulu ya kalau mau bikin branch baru supaya branch itu menjadi branch yang up to date.



Tadi Pull, sekarang Push. Eits, tapi bukan mau olahraga ya, gengs!

Jadi, push ini dipakai buat melakukan sinkronisasi perubahan yang ada di remote biar sesuai sama yang ada di local. Kebalikannya dari Pull, kan?

Command-nya berupa:

- Untuk melakukan push dari current branch ke remote branch tertentu (dalam contoh remote branchnya adalah foo-branch).

```
$ git push origin foo-branch
```



Kalau nggak bisa dipakai, gimana?

Nah, kalau misalnya ada suatu case dimana kita nggak bisa melakukan push. Biasanya terjadi karena adanya suatu conflict. Biar bisa dipakai, conflict ini harus diselesaikan terlebih dahulu.

Ketika terjadi suatu conflict, kita dilarang keras buat melakukan:

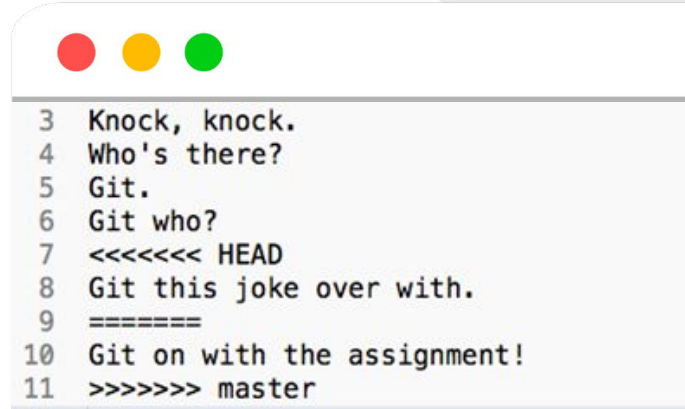
```
git push --force
```



Biasanya conflict terjadi kalau ada perubahan di bagian yang sama pada suatu file yang sama, gengs!

Di samping adalah tampilan kalau terjadi conflict.

Biar bisa melakukan operasi git yang lain, kita harus nyelesain conflict ini dengan editor, ya.



```
3 Knock, knock.  
4 Who's there?  
5 Git.  
6 Git who?  
7 <<<<<<< HEAD  
8 Git this joke over with.  
9 =====  
10 Git on with the assignment!  
11 >>>>>>> master
```

Berikut tampilan ketika terjadi conflict yang sudah di fixed~

```
1 Hello World!  
2  
3 Knock, knock.  
4 Who's there?  
5 Git.  
6 Git who?  
7 Git this joke over with and on with the assignment!
```

Pastikan udah nggak ada DIVIDER, LEFT dan RIGHT.

Sekarang kita lakukan git add dan lakukan commit buat finalisasi hasil fixing tersebut.

Selesai, deh!

Untuk selanjutnya, kita harus nyiapin Git lab, akun dan instalasi di local, ya.



Tadaa, ada penugasan, nih!

Kamu dipersilakan untuk membuat Branch yang di dalamnya terdapat fitur calculator di console (case pada Challenge Chapter 1)! Branch yang kamu buat terdiri dari:

- Master
- Development
- Branch fitur penjumlahan
- Branch fitur pengurangan
- Branch fitur perkalian
- Branch fitur pembagian
- Branch fitur modulus

Kalau tugasnya selesai, kamu boleh minta facilitator untuk mengecek penugasan yang kamu kerjakan di pertemuan selanjutnya. Selamat mengerjakan 😊



Kita udah berpetualang dari pengenalan Java, tools, data type, variable, clean code sampai dengan version control. Selamat! Kita udah tuntas belajar Chapter 1! 🙌

Dari semua topik yang udah dipelajari, ada topik yang kamu sulit pahami?

Kalau ada kira-kira bagian yang mana, sob? Coba kamu catat dan silakan pelajari kembali materinya ya!



Nah, selesai sudah pembahasan kita di **Chapter 1** ini. Luar biasa! 🎉

Selanjutnya, kita bakal siap-siap untuk move on ke chapter baru yaitu:

✨**Chapter 2.** ✨

Sampai jumpa pada chapter selanjutnya~

