



# Crash Course

Faisal Rahman

Faculty of Computer Science, Universitas Indonesia

[id.faisalrahman@gmail.com](mailto:id.faisalrahman@gmail.com)

# The Basics

# Kenapa VCS?

Dengan *Version Control System* (VCS) seperti Git, kita dapat menyimpan histori perubahan seluruh file pada project kita → Jika sewaktu-waktu ditemukan kesalahan, kita dapat melakukan rollback ke versi sebelumnya.

Dokumentasi proses development menjadi lebih baik karena setiap commit disertai deskripsi.

Git juga dapat bertindak sebagai backup di berbagai situasi.

Kontrol proses development menjadi lebih baik. Bayangkan jika tim Anda melakukan kolaborasi dengan shared folder, kemudian teman Anda mengubah file yang sedang Anda kerjakan, lalu Anda menyimpan file tersebut dan menimpa perubahan yang dilakukan teman Anda. Bencana?

# Git



# git

Adalah *version control system* terpopuler yang digunakan dalam pengembangan perangkat lunak.

Dikembangkan pada tahun 2005 oleh Linus Torvalds dan tim *developer* Linux Kernel.

42.9% *software developer* profesional menggunakan Git sebagai VCS utamanya [\[1\]](#).

# Instalasi

## Core application

Windows: <https://git-scm.com/downloads/win>

Linux Debian/Ubuntu: `$apt-get install git`

GUI (optional): <https://git-scm.com/downloads/guis>

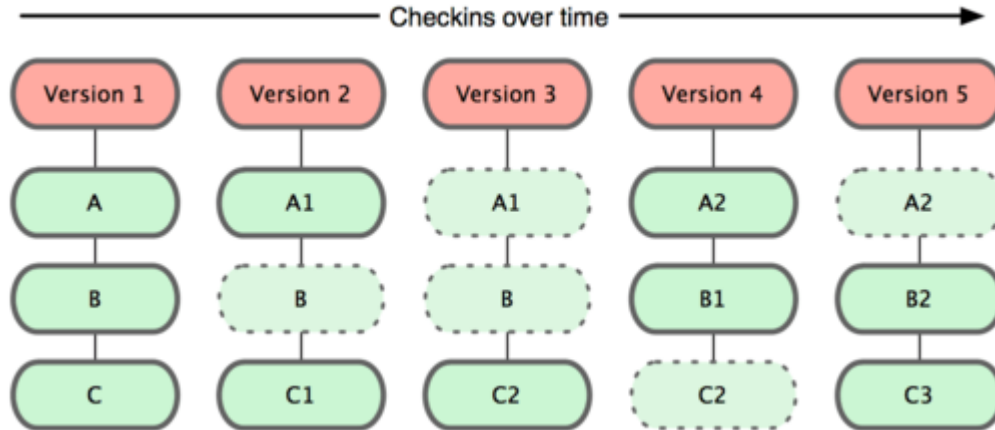
Untuk platform Windows, Anda bisa memilih untuk menggunakan Git Bash atau integrasi shell Git dengan Command Prompt.

Perhatikan bahwa saat tutorial ini ditulis, versi stable Git terbaru adalah **2.8.1**.

Terdapat beberapa perbedaan antara versi 1.x dan versi 2.x. Tutorial ini mengasumsikan Anda menggunakan versi 2.x.

# Bagaimana Git Bekerja? (1/4)

Git menyimpan data kita sebagai kumpulan *snapshot* dari file system kita. Setiap snapshot tersebut menyimpan keadaan dari seluruh file kita saat kita melakukan *commit*. Perhatikan juga bahwa Git tidak menyimpan keadaan dari file yang tidak berubah, hanya menyimpan *link* ke file pada keadaan sebelumnya.



# Bagaimana Git Bekerja? (2/4)

Bagaimana menjamin file kita tidak korup atau berubah saat diproses Git?

Git melakukan *checksum* dengan fungsi SHA-1 setiap kali Git menyimpan data. *Checksum* ini berfungsi sebagai “*fingerprint*” yang dapat membuktikan integritas setiap *snapshot*-nya. Ketika ada perbedaan SHA-1 hash, Git pasti dapat mendeteksinya. Perhatikan SHA-1 hash yang berupa 40 karakter heksadesimal di bawah ini:

```
icalrn@Faisal-Asuspro ~/web/yii2 $ git log
commit 34d999f467080ed280c3a492048b06ba800bca47
Author: Faisal <id.faisalrahman@gmail.com>
Date:   Wed Apr 13 22:12:17 2016 +0700

    create user model
```

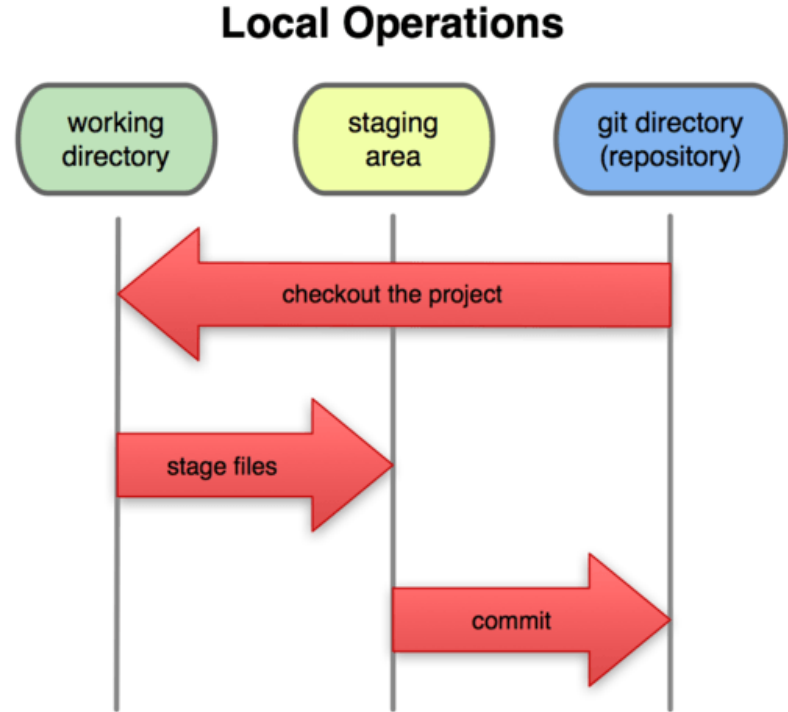
# Bagaimana Git Bekerja? (3/4)

## The Three States:

**Modified**, adalah data yang sudah berubah tetapi masih berada di working directory dan belum disimpan ke dalam repositori Git.

**Staged**, adalah data yang akan dimasukkan ke langkah *commit* selanjutnya dan direkam dalam *staging area*.

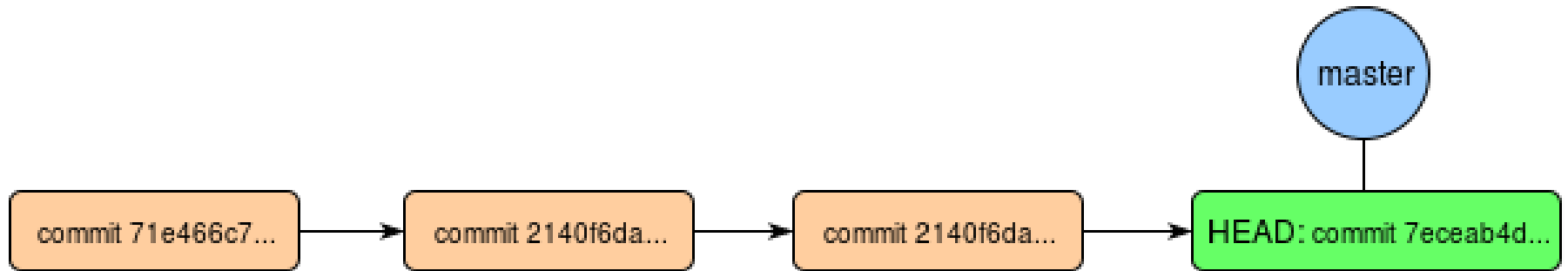
**Committed**, yaitu data sudah disimpan dengan aman di repositori Git.





# Bagaimana Git Bekerja? (4/4)

Rantai *snapshots* dengan **HEAD** pointer yang merepresentasikan keadaan repositori saat ini.



# Proses Development Bersama Git

Inisialisasi direktori project → Set-up remote repository (jika perlu) → *Happy Coding!*  
→ Update Index → Commit → Push (jika perlu)

# **Workflow Example**

# Contoh Workflow Git

Saya membuat direktori **test** yang akan dijadikan *local repository*.

```
icalrn@Faisal-Asuspro ~ $ mkdir test  
icalrn@Faisal-Asuspro ~ $ cd test/  
icalrn@Faisal-Asuspro ~/test $
```

Kemudian lakukan inisialisasi repositori git dengan command **git init**.

```
icalrn@Faisal-Asuspro ~/test $ git init  
Initialized empty Git repository in /home/icalrn/test/.git/  
icalrn@Faisal-Asuspro ~/test $
```

Setelah inisialisasi, direktori **test** ini menjadi sebuah repositori Git. Artinya, Git akan melacak perubahan pada isi direktori tersebut dan kita dapat melakukan operasi-operasi Git di dalamnya.

# Contoh Workflow Git

Kemudian saya menambahkan sebuah file bernama **file1** di dalam direktori **test** tersebut.

```
icalrn@Faisal-Asuspro ~/test $ touch file1
icalrn@Faisal-Asuspro ~/test $ ls
file1
icalrn@Faisal-Asuspro ~/test $
```

Mari kita lihat apakah Git menyadari adanya penambahan file tadi dengan command **git status**.

```
icalrn@Faisal-Asuspro ~/test $ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        file1

nothing added to commit but untracked files present (use "git add" to track)
```

# Contoh Workflow Git

**File1** tercatat sebagai *untracked file*, artinya file tersebut belum dilacak oleh Git.

Ketika sudah siap untuk disimpan, masukkan ke *staging area* dengan command **git add**, kemudian periksa keadaan *staging area* dengan command **git status**.

```
icalrn@Faisal-Asuspro ~/test $ git add file1
icalrn@Faisal-Asuspro ~/test $ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   file1
```

# Contoh Workflow Git

Terlihat bahwa **file1** tercatat sebagai *changes to be committed*, artinya file tersebut sudah berada di *staging area*.

Kemudian kita buat file baru lagi bernama **file2**, seperti yang kita duga, ia akan

```
icalrn@Faisal-Asuspro ~/test $ touch file2
icalrn@Faisal-Asuspro ~/test $ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   file1

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    file2
```

# Contoh Workflow Git

Pada poin ini, ketika kita melakukan **commit** maka yang akan tersimpan dalam repositori hanyalah **file1** karena **file2** belum masuk *staging area*.

Dapat kita lihat dari kasus ini bagaimana *staging area* memberikan fleksibilitas dalam *workflow kita*. Kita memiliki kontrol untuk menentukan perubahan apa yang akan disimpan pada commit yang akan dilakukan.

Mari kita lakukan commit. Jangan lupa untuk menyertakan ***commit message*** dalam setiap commit kita, hal ini sangat penting untuk mendokumentasikan perubahan yang kita lakukan di setiap commit.



# Contoh Workflow Git

Lakukan commit dengan command **git commit**. Sertakan opsi **-m** untuk menambahkan pesan commit.

```
icalrn@Faisal-Asuspro ~/test $ git commit -m "membuat file1"
[master (root-commit) fcfcfbc] membuat file1
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file1
```

Kemudian kita bisa melihat histori commit yang sudah dilakukan dengan **git log**.

```
icalrn@Faisal-Asuspro ~/test $ git log
commit fcfcfbc00a3780c46bad05bae45653254ca010f2
Author: Faisal <id.faisalrahman@gmail.com>
Date:   Wed Apr 13 23:40:22 2016 +0700

    membuat file1
```

# The Mantra...

**Commit *early*, and commit *often*!**

Karena Git hanya bertanggungjawab terhadap data yang sudah Anda commit. Juga jangan lupa tuliskan *commit message* se-deskriptif mungkin!

# **Collaborating with Git**

# Remote Repository

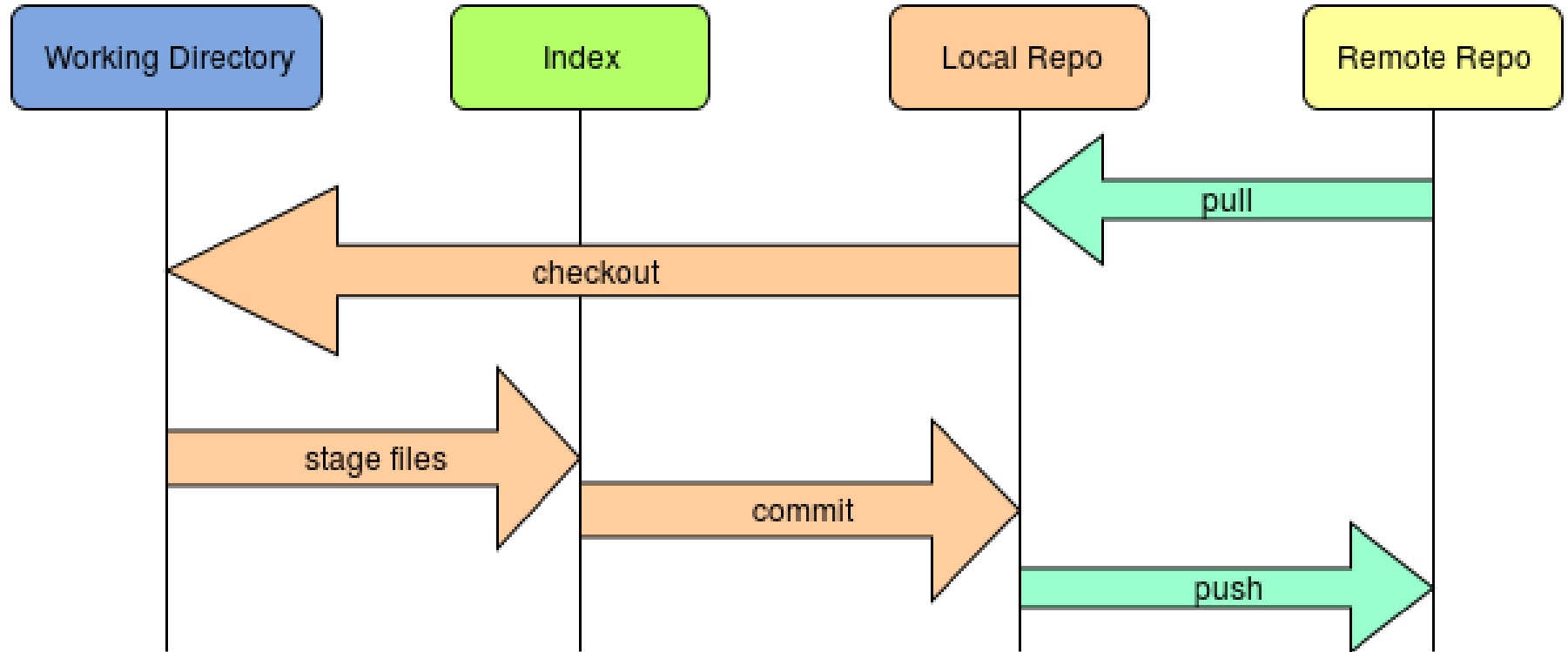
Kolaborasi pengembangan aplikasi dengan Git dilakukan dengan menggunakan **repositori *remote* yang tersentralisasi**.

Setiap anggota tim mengakses repositori *remote* tersebut untuk mengambil versi terbaru repositori dan meletakkannya di repositori lokal masing-masing.

Kemudian modifikasi kode dilakukan di repositori lokal. Lakukan perubahan, commit, kemudian kirim kembali ke repositori *remote* agar dapat diakses anggota tim lain.

Aktivitas mengambil data repositori *remote* disebut **pull**, sementara aktivitas mengirim data ke repositori *remote* disebut **push**.

# Kolaborasi Dengan Remote Repository



# Git Repository Hosting Service

Remote repository dapat berupa repositori git pada satu server yang dapat diakses seluruh tim, atau menggunakan *Git repository hosting service* di internet.

Anda dapat menemukan banyak *git hosting service* di internet, seperti [Github](#), [GitLab](#), [BitBucket](#), [Codebase](#), dan lainnya. Pada tutorial ini saya akan menggunakan Git



codebase

Atlassian



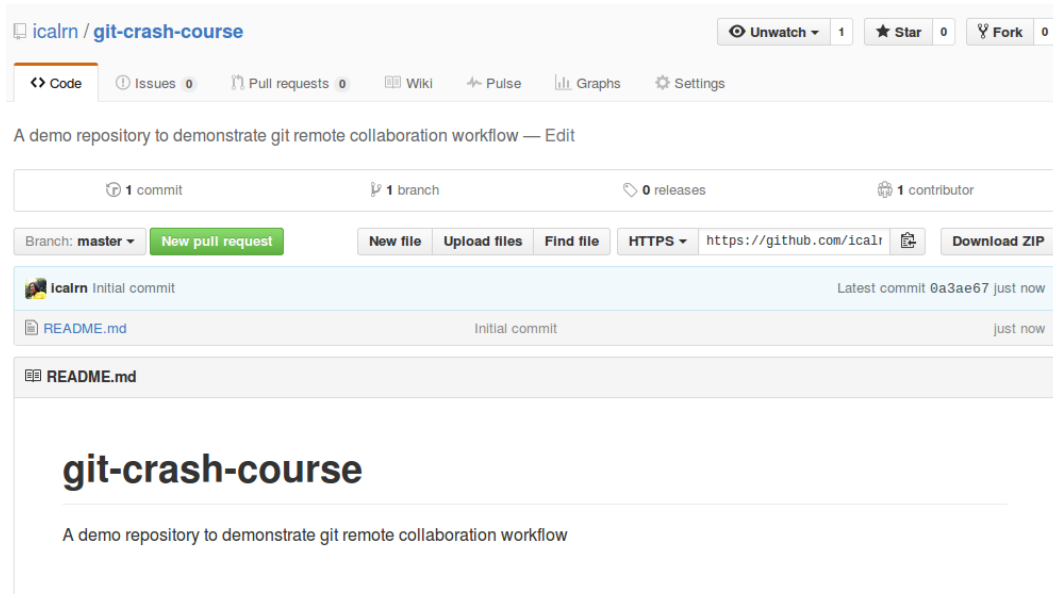
Bitbucket



GitLab

# Bekerja Dengan Remote Repo

Pertama-tama, saya akan membuat *remote repository* di github pada alamat <http://github.com/icalrn/git-crash-course>.



The screenshot shows the GitHub interface for the repository 'icalrn / git-crash-course'. At the top, there are buttons for 'Unwatch', 'Star' (0), and 'Fork' (0). Below this is a navigation bar with links for 'Code', 'Issues' (0), 'Pull requests' (0), 'Wiki', 'Pulse', 'Graphs', and 'Settings'. The main content area starts with the text 'A demo repository to demonstrate git remote collaboration workflow — Edit'. Below this is a summary bar showing '1 commit', '1 branch', '0 releases', and '1 contributor'. A secondary bar contains a 'Branch: master' dropdown, a 'New pull request' button, and buttons for 'New file', 'Upload files', 'Find file', 'HTTPS' dropdown, the repository URL 'https://github.com/icalr', and a 'Download ZIP' button. The commit history section shows 'icalrn Initial commit' as the latest commit (0a3ae67) 'just now'. Below this, the 'README.md' file is displayed, showing the title 'git-crash-course' and the description 'A demo repository to demonstrate git remote collaboration workflow'.

# Bekerja Dengan Remote Repo

Kemudian buat local repository yang terhubung ke remote repository, terdapat dua cara untuk ini:

Lakukan **git clone** dari remote repository untuk membuat direktori local repository baru

```
icalrn@Faisal-Asuspro ~/web $ git clone https://icalrn.github.com/icalrn/git-crash-course
Cloning into 'git-crash-course'...
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
icalrn@Faisal-Asuspro ~/web $ ls git-crash-course/
README.md
```

Tambahkan penunjuk ke remote repository dengan **git remote add** pada direktori local repository Anda.



# Bekerja Dengan Remote Repo

Tambahkan penunjuk ke remote repository dengan **git remote add** pada direktori local repository Anda kemudian lakukan **pull** dari branch master.

```
icalrn@Faisal-Asuspro ~/web $ mkdir git-crash-course
icalrn@Faisal-Asuspro ~/web $ cd git-crash-course/
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git init
Initialized empty Git repository in /home/icalrn/web/git-crash-course/.git/
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git remote add origin https://icalrn.github.com/icalrn/git-crash-course
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/icalrn/git-crash-course
* branch          master      -> FETCH_HEAD
* [new branch]    master      -> origin/master
```

# Bekerja Dengan Remote Repo

Lakukan pekerjaan seperti biasa di local repository Anda (i.e. *modify* → *add to index* → *commit*).

```
icalrn@Faisal-Asuspro ~/web/git-crash-course $ touch file1.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ echo "this is a test file" > file1.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    file1.txt

nothing added to commit but untracked files present (use "git add" to track)
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git add .
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git commit -m "added file1.txt"
[master f8a174a] added file1.txt
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt
```

# Bekerja Dengan Remote Repo

Pada poin ini, perubahan yang Anda lakukan baru tersimpan di local repository Anda dan belum tersimpan di server remote repository.

```
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git log
commit f8a174a00627c8d0561de9ba0c8b83f6b653c8dc
Author: Faisal Rahman <id.faisalrahman@gmail.com>
Date: Thu Apr 14 12:02:43 2016 +0700
```

added file1.txt

```
commit 0a3ae677e132dd431dbb61361aaf18aa62d0f151
Author: Faisal Rahman <id.faisalrahman@gmail.com>
Date: Thu Apr 14 11:48:10 2016 +0700
```

Initial commit

Branch: master ▾

Commits on Apr 14, 2016



**Initial commit**

icalrn committed 18 minutes ago



0a3ae67



# Bekerja Dengan Remote Repo

Untuk mengirimkan commit kita ke remote repository, lakukan **git push**.

```
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git push origin master
Password for 'https://icalrn@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 301 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://icalrn@github.com:icalrn/git-crash-course
    0a3ae67..f8a174a  master -> master
```

Branch: **master** ▼

Commits on Apr 14, 2016



**added file1.txt**

icalrn committed 8 minutes ago



f8a174a



**Initial commit**

icalrn committed 23 minutes ago



0a3ae67



# What If...

Apa yang akan terjadi jika kita mengubah file1.txt di local repository kita, namun saat akan kita push ke remote repository, ternyata sudah ada yang mengubah file1.txt di remote repository?

```
icalrn@Faisal-Asuspro ~/web/git-crash-course $ echo "what will happen if we modify this?" > file1.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ 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 checkout -- <file>..." to discard changes in working directory)

        modified:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git add file1.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git commit -m "changed file1.txt and wondering"
[master fde2dd0] changed file1.txt and wondering
1 file changed, 1 insertion(+), 1 deletion(-)
```

# What If...


Sebelum kita sempat push ke remote repository, ternyata ada yang mengubah file1.txt...

Branch: **master** ▾

**git-crash-course** / **file1.txt**

Find file

Copy path

 **icalrn** change the contents of file1.txt

d8c3e49 just now


1 contributor


3 lines (2 sloc) | 68 Bytes

Raw

Blame

History





1	this is a test file
2	let's add this line and see what happens *grin*

# What If...

Wah! Kita tidak boleh melakukan **push** karena local repository kita tidak up to date dengan remote repository...

```
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git push
To https://icalrn@github.com:icalrn/git-crash-course
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://icalrn@github.com:icalrn/git-crash-course'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Ini berarti telah ada commit yang terekam di remote repository yang tidak terekam di local repository kita sehingga kita harus lakukan **pull** terlebih dahulu dari remote repository.

# What If...

Whoa! Ternyata terjadi konflik! Artinya ada file yang sama-sama diubah oleh dua commit yang mereferensikan commit yang sama sebelumnya.

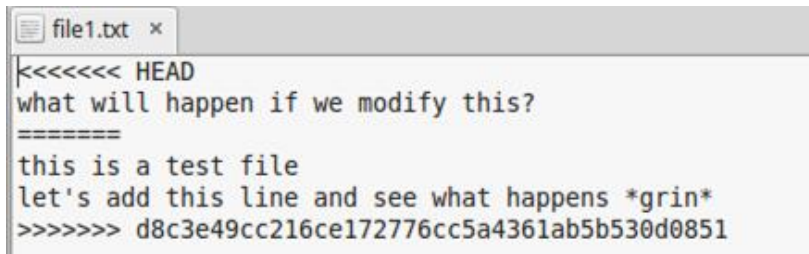
```
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/icalrn/git-crash-course
   f8a174a..d8c3e49  master    -> origin/master
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Git dapat mengatasi konflik secara otomatis sampai batasan tertentu, namun ternyata konflik yang terjadi pada kasus kita membutuhkan pengecekan secara manual.



# What If...

Mari kita lihat file yang bermasalah tersebut...

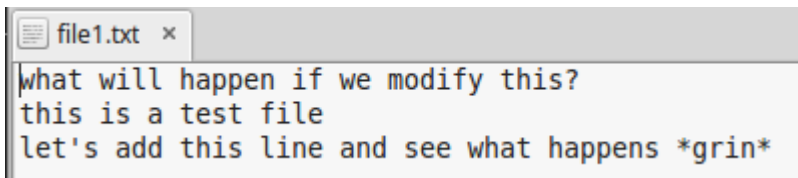


```
file1.txt x
<<<<<< HEAD
what will happen if we modify this?
=====
this is a test file
let's add this line and see what happens *grin*
>>>>>> d8c3e49cc216ce172776cc5a4361ab5b530d0851
```

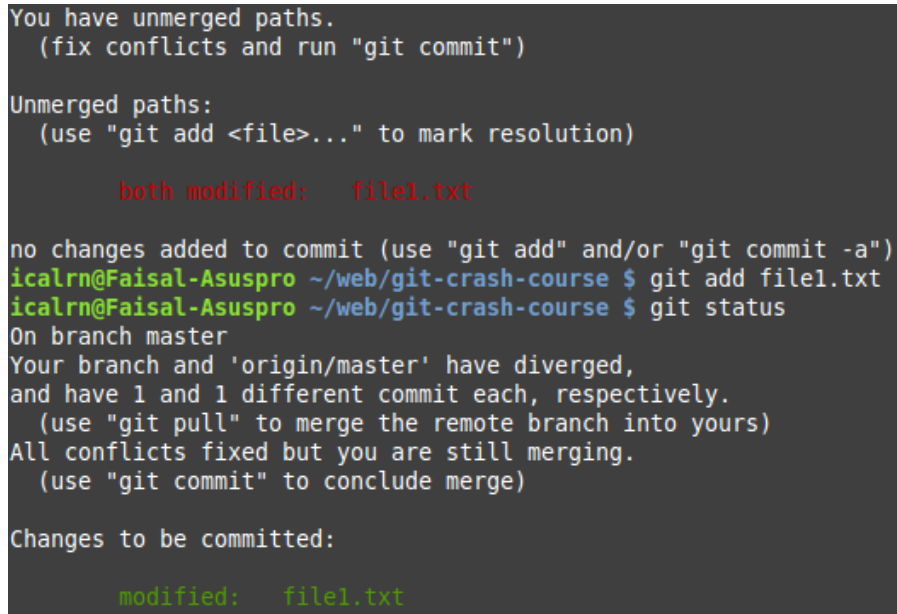
Git menandai dengan pembatas “<<<<<< **HEAD**” sampai “=====” yang artinya semua baris di antara kedua pembatas tersebut adalah baris yang terekam oleh pointer HEAD kita (commit terakhir di local repo), sementara baris selanjutnya sampai “>>>>>> **d8c3e4...**” adalah baris yang terekam pada commit **d8c3e4...** yang berada di remote repository.

# What If...

Kemudian kita resolve konflik tersebut dengan menuliskan perubahan yang ingin disimpan dan menghapus pembatas tersebut. Setelahnya, masukkan kembali file tersebut ke dalam index.



```
file1.txt x
what will happen if we modify this?
this is a test file
let's add this line and see what happens *grin*
```



```
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git add file1.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commit each, respectively.
  (use "git pull" to merge the remote branch into yours)
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

        modified:   file1.txt
```

# What If...

Setelahnya, commit index tersebut.

```
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git commit -m "resolved conflicts on file1.txt"
[master 21b86c5] resolved conflicts on file1.txt
```

Ingat bahwa kita belum melakukan **push** karena terlibat konflik tadi. Maka dari itu, mari kita lakukan **push**.
















```
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git push
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 705 bytes | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://icalrn@github.com:icalrn/git-crash-course
d8c3e49..21b86c5 master -> master
```

# What If...

Setelah itu, mari periksa keadaan di remote repository.

Branch: **master** ▾

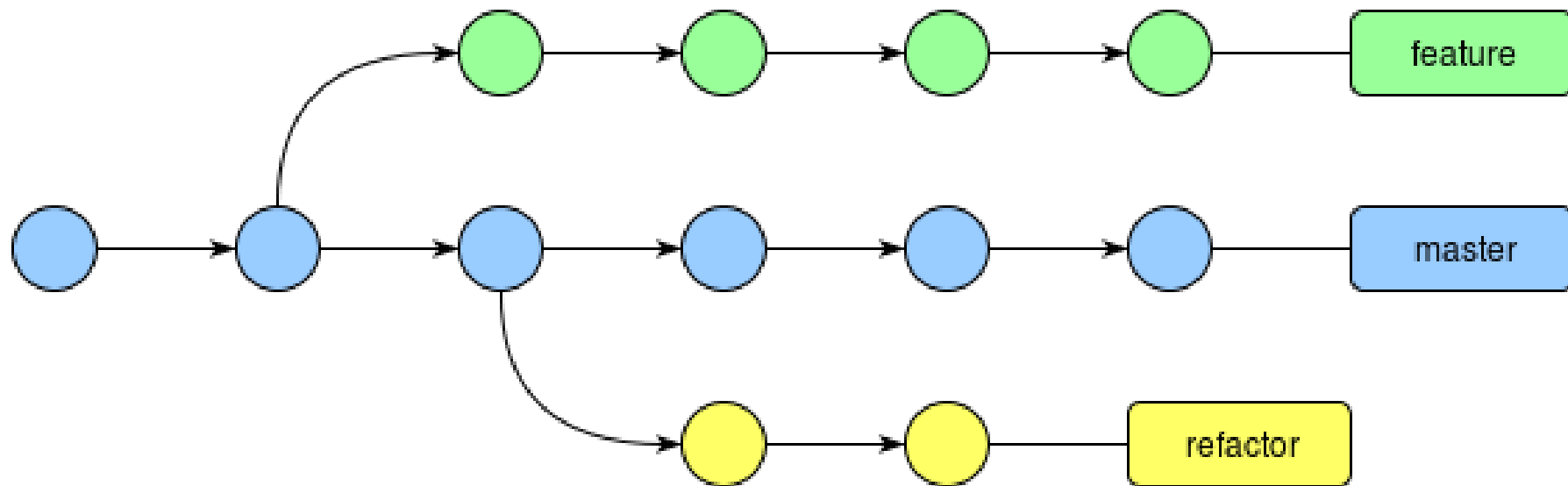
Commits on Apr 14, 2016

 <b>resolved conflicts on file1.txt</b> lcalrn committed 3 minutes ago	 <b>21b86c5</b> 
 <b>change the contents of file1.txt</b> lcalrn committed 21 minutes ago	 <b>d8c3e49</b> 
 <b>changed file1.txt and wondering</b> lcalrn committed 22 minutes ago	 <b>fde2dd0</b> 
 <b>added file1.txt</b> lcalrn committed 39 minutes ago	 <b>f8a174a</b> 
 <b>Initial commit</b> lcalrn committed an hour ago	 <b>0a3ae67</b> 

Untuk meminimalisasi risiko konflik seperti yang kita alami tadi, *best practice* yang dilakukan adalah dengan menggunakan fitur ***branching and merging***.

# **Basic Branching and Merging**

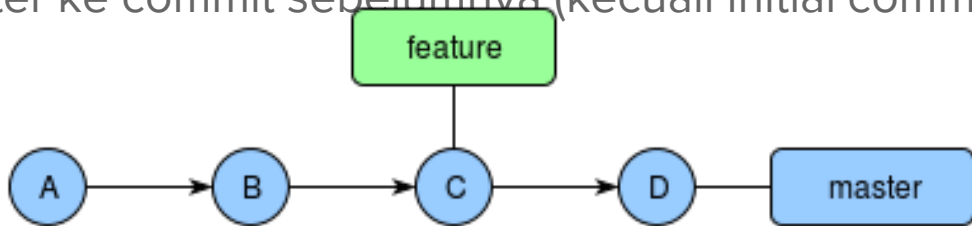
# Apa Itu Branch?



# Apa Itu Branch?

Pada dasarnya, perintah branch dalam Git hanya membuat **pointer** ke suatu commit.

Ingat bahwa esensi Git adalah rangkaian commit, jadi setiap commit pasti memiliki pointer ke commit sebelumnya (kecuali initial commit).



Pada contoh di atas, branch **master** menunjuk ke commit D. Sementara itu, branch **feature** menunjuk ke commit C.

# Kenapa Melakukan Branching?

Enkapsulasi kode yang masih dalam keadaan *unstable* agar tidak mengontaminasi basis kode utama (master).

Menghindari terjadinya konflik akibat banyak orang bekerja pada satu branch tertentu (seperti yang sudah dicontohkan di bagian sebelumnya).

Pada prinsipnya, branch master sebaiknya hanya berisi keadaan yang *stable*, pengembangan fitur sebaiknya dilakukan di branch lain lalu kemudian disatukan (*merge*) ke branch master setelah fitur tersebut lolos testing.



# Melakukan Branching

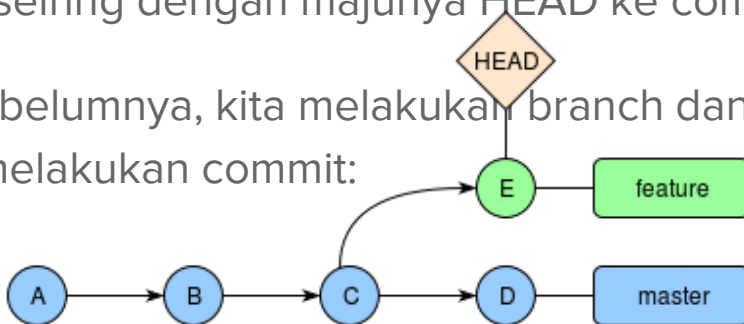
Membuat branch dapat dilakukan dengan dua langkah:

**git branch <nama branch>** akan membuat branch baru dengan nama yang disertakan.

**git checkout <nama branch>** akan memindahkan HEAD ke branch tersebut dan menandai branch yang aktif.

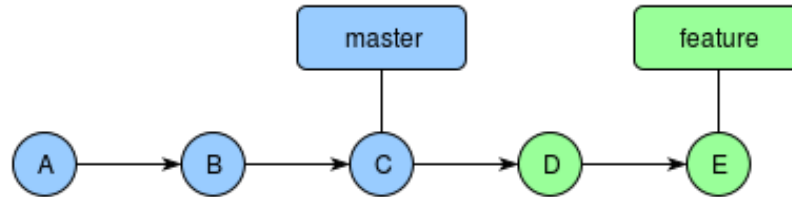
Ketika kita sudah melakukan **checkout** ke suatu branch, berarti pointer branch tersebut akan maju seiring dengan majunya HEAD ke commit selanjutnya.

Misalkan dari contoh sebelumnya, kita melakukan branch dan checkout ke branch *feature*, kemudian melakukan commit:

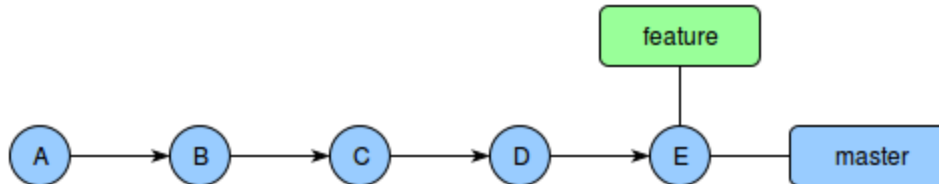


# Merging: Fast-Forward

Perhatikan skenario branch berikut:



Pada skenario tersebut, branch **master** adalah *direct ancestor* dari branch **feature**. Kondisi ini memungkinkan dilakukannya **fast-forward merge**. Ketika merge dilakukan, yang akan terjadi adalah:



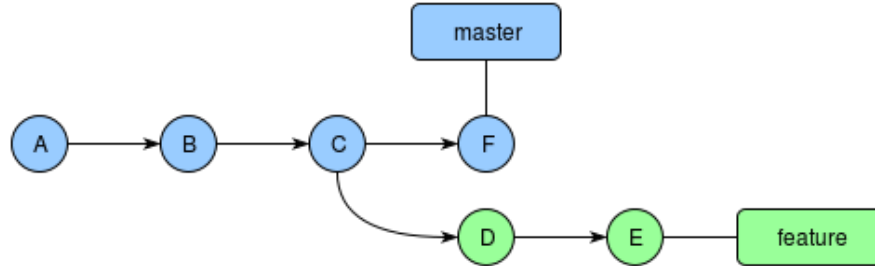
# Merging: Fast-Forward

Dengan **fast-forward merging**, git hanya perlu memindahkan pointer branch **master** ke commit E karena pointer branch **master** saat ini merupakan *direct ancestor* dari branch **feature**. Artinya, tidak ada perubahan pada **master** sejak **feature** dibuat.

Namun, apabila pointer sebuah branch bukan merupakan *direct ancestor* dari branch yang akan di-merge ke dalamnya, fast-forward merging tidak mungkin dilakukan. Ketika ini terjadi, git akan melakukan **three-way merging**.

# Merging: Three-Way

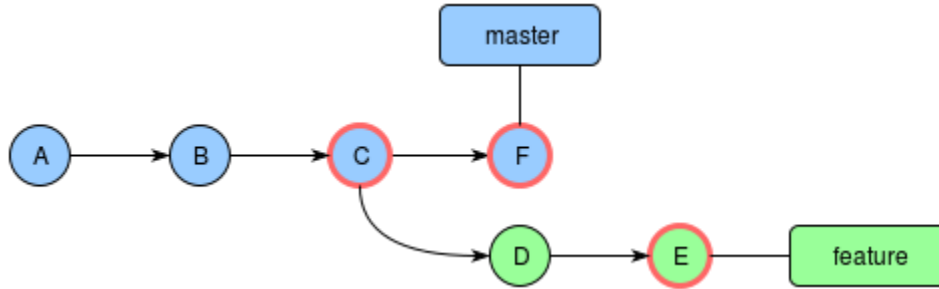
Perhatikan skenario branch berikut:



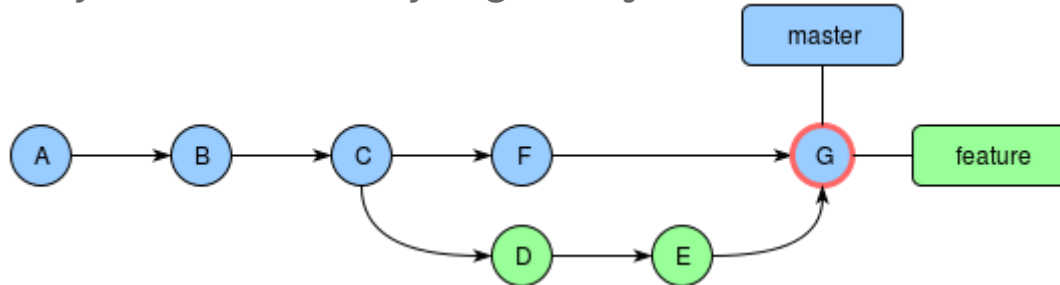
Ancestor yang dimiliki **feature** pada **master** adalah commit C yang sudah tertinggal satu commit dari pointer **master** → terdapat *common ancestor* commit C pada dua branch tersebut.

Ketika **feature** akan di-merge ke **master**, git akan melakukan merge tiga snapshot: ujung **master** (F), ujung **feature** (E), dan *common ancestor* (C).

# Merging: Three-Way



Setelahnya, git akan menyimpan hasil merge ketiga snapshot tersebut di snapshot baru yaitu **commit G** yang ditunjuk oleh kedua branch tersebut.



# Simulasi Fast-Forward Merge

```
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git checkout -b "feature_branch"
Switched to a new branch 'feature_branch'
icalrn@Faisal-Asuspro ~/web/git-crash-course $ touch file2.txt | echo "test for branch" >> file2.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git add .
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git commit -m "added file2.txt."
[feature_branch 7be07de] added file2.txt.
 1 file changed, 1 insertion(+)
 create mode 100644 file2.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git merge feature_branch
Updating 21b86c5..7be07de
Fast-forward
 file2.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 file2.txt
```

# Simulasi Three-Way Merge

```
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git checkout -b "feature2_branch"
Switched to a new branch 'feature2_branch'
icalrn@Faisal-Asuspro ~/web/git-crash-course $ touch file3.txt | echo "test for 3-way merge" >> file3.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git add .
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git commit -m "added file3.txt."
[feature2_branch e7f2caa] added file3.txt.
 1 file changed, 1 insertion(+)
 create mode 100644 file3.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
icalrn@Faisal-Asuspro ~/web/git-crash-course $ touch file4.txt | echo "test for 3-way merge" >> file4.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git add .
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git commit -m "added file4.txt."
[master 1e9ea5e] added file4.txt.
 1 file changed, 1 insertion(+)
 create mode 100644 file4.txt
icalrn@Faisal-Asuspro ~/web/git-crash-course $ git merge feature2_branch
Merge made by the 'recursive' strategy.
 file3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 file3.txt
```

# Branch dan Remote Repository

Branch yang kita buat di local repository hanya hidup di local repository kita saja. Untuk dapat menghubungkannya dengan remote repository, kita harus membuatnya menjadi **tracking branch**.

**Tracking branch** adalah branch pada local repo yang mereferensi ke branch di remote repo, yang kemudian disebut **upstream branch**. Dengan begitu, branch pada local repo bisa melakukan **pull**, **push** dan **merge** ke branch tersebut.

Untuk menjadikan suatu branch menjadi **tracking branch**, gunakan perintah **git branch -u <nama remote repo> <nama branch di remote repo>**

Jika branch baru dibuat di local repository, lakukan **git push -u <nama remote repo> <nama branch di remote repo>** untuk satu kali saja. Dengan begitu,



# **Commands**

## **Basics**

# Git add

Digunakan untuk menambahkan file ke **staging area** sebelum melakukan commit.

Variasi penggunaan:

`git add <path/to/file>` → menambahkan file spesifik

`git add .` | `git add -A` → menambahkan file baru (*untracked*), file yang berubah (*modified*), dan file yang dihapus (*removed*)

`git add -u` → menambahkan file yang berubah dan file yang dihapus saja

`git add --ignore-removal` → menambahkan file yang baru dan file yang berubah saja

# Git commit

Menyimpan seluruh perubahan pada index (*staging area*) dalam commit baru disertai dengan *commit message* yang mendeskripsikan commit tersebut.

Penggunaan:

`git commit -a` → melakukan *staging* otomatis terhadap file yang berubah dan file yang dihapus, kemudian melakukan commit

`git commit --interactive` → melakukan commit dengan *interactive mode* dimana kita dapat menentukan file apa saja yang akan di-commit, meningkatkan fleksibilitas

`git commit -m <message>` → melakukan commit dengan *message* yang diberikan

# Git log

Menampilkan log dari commit-commit yang telah dilakukan beserta commit message nya.

Penggunaan:

`git log`

`git log -p` → mencetak log bersama dengan *patch* yang akan lebih menjelaskan apa yang dilakukan pada setiap commit

# Git reset

Reset HEAD pada saat ini ke keadaan yang disebutkan. Bisa juga digunakan untuk mengosongkan index/*staging area* tanpa me-reset perubahan yang dilakukan terhadap file-file di index.

Penggunaan:

git reset → mengosongkan index

git reset --soft <HEAD pointer> → *undo* commit, namun tetap membiarkan perubahan file dan entri index seperti sebelum commit dilakukan. Biasanya digunakan apabila terjadi kesalahan penulisan commit message.

git reset --hard <HEAD pointer> → reset index dan perubahan file yang dilakukan. Biasanya dilakukan dalam kasus-kasus dimana kita perlu kembali ke keadaan bersih commit terakhir.

# Git checkout

Update file pada working tree agar sesuai dengan keadaan di index. Jika file yang disebutkan tidak berada dalam index, maka file tersebut akan di-reset menjadi keadaannya pada commit terakhir. Juga digunakan untuk memindahkan HEAD ke branch atau commit.

Penggunaan:

`git checkout <path/to/file>` → lakukan checkout pada file spesifik

`git checkout <branch/commit>` → memindahkan HEAD pointer ke HEAD branch/commit yang disebutkan

`git checkout -b <new branch>` → membuat branch baru kemudian pindahkan HEAD ke branch tersebut. Ekuivalen dengan `git branch <new branch> && git checkout <new branch>`

# Git push

Update remote repository berdasarkan keadaan local repository

## Penggunaan

`git push <nama remote repository>` → push perubahan pada tracking branch aktif ke branch upstreamnya pada remote repository yang disebutkan

`git push <nama remote repository> <nama branch pada remote repository>` → push perubahan pada branch aktif ke branch dan remote repository sesuai parameter yang disebutkan

`git push -u <nama branch pada remote repository>` → membuat branch dengan nama yang disebutkan ke remote repository, set branch tersebut menjadi upstream branch untuk branch aktif, dan push perubahan pada branch aktif ke branch tersebut.

# Git pull

Mengambil data perubahan dari remote repository dan mengintegrasikannya ke dalam branch aktif.

Penggunaan:

`git pull` → jika branch memiliki upstream branch (merupakan tracking branch), ambil data dari upstream branch dan merge ke dalam tracking branch

`git pull <nama remote repo> <nama branch>` → ambil data dari repository dan branch yang disebutkan, kemudian merge ke dalam branch aktif.



# Git branch

Melihat daftar branch yang ada, membuat, atau menghapus branch.

Penggunaan:

`git branch` → melihat daftar branch yang ada pada repository.

`git branch <nama branch>` → membuat branch baru dengan nama yang disebutkan.

`git branch -d <nama branch>` → menghapus branch yang disebutkan (harus sudah di-merge).

`git branch -D <nama branch>` → menghapus paksa branch yang disebutkan.

`git branch -u <nama remote repository> <nama branch di remote repository>` → mengatur upstream branch untuk branch aktif berdasarkan parameter yang disebutkan.

# Git merge

Menggabungkan dua atau lebih branch.

Penggunaan:

`git merge <nama branch>` → menggabungkan branch yang disebutkan ke dalam branch aktif.

`git merge --no-ff <nama branch>` → menggabungkan branch yang disebutkan ke dalam branch aktif tanpa melakukan fast-forward meskipun skenario memungkinkan.

`git merge <nama branch> -m <commit message>` → melakukan merge branch yang disebutkan ke branch aktif dengan commit message yang disebutkan (jika terjadi commit dalam proses merge).

# Hope this helps!

Untuk pertanyaan, diskusi ataupun koreksi silakan hubungi saya di **id.faisalrahman@gmail.com** atau LINE saya di **@icalrn**. Have a good day!