**Rangkuman Belajar Git & GitHub**

**Kerja di lokal Git dan Update ke GitHub**

**git command (local) :**

**git add**

**git add .**

**git add -p (bisa pilih perbaris)**

**git branch**

**git branch features**

**git branch –merged**

**git branch -d about**

**git checkout**

**git checkout -b features**

**git checkout hash -- namafile**

**git commit -am "msg"**

**git commit -m "msg"**

**git commit –amend -am "msg"**

**git commit –amend -m "msg"**

**git config**

**git config --global alias.log1 "log --all --oneline --decorate --graph"**

**git clone**

**git clone** [**https://github.com/ariefcu/simple-landing-page.git**](https://github.com/ariefcu/simple-landing-page.git)**’**

**git diff**

**git fetch**

**git fetch sandhikagalih**

**git init**

**git help**

**git log**

**git merge**

**git merge sandhikagalih/master**

**git push -u origin master**

**git push**

**git pull**

**git rebase**

**git remote**

**git remote add origin https://github.co/ariefcu/ariefcu.githbub.io.git**

**git remote add sandhikagalih https://github.com/asandhikagalih/simple-landing-page**

**git remote -v**

**git remote set-url origin https://github.com/ariefcu/Laravel-10-User-Roles-and-Permissions-Buat-Dari-Awal.git**

**git restore**

**git restore --staged**

**git reset –soft**

**git reset –mixed (git reset)**

**git reset –hard**

**git revert**

**git status**

**git show**

[**https://medium.com/@gofrendiasgard/git-perjalanan-menuju-dunia-parallel-c4e74364cf84**](https://medium.com/@gofrendiasgard/git-perjalanan-menuju-dunia-parallel-c4e74364cf84)

**git revert <commit-bermasalah>**

**git reflog (mirip dengan : git log –all –oneline –decorate –graph)**

## EXAMPLES git log

git log --no-merges

Show the whole commit history, but skip any merges

git log v2.6.12.. include/scsi drivers/scsi

Show all commits since version v2.6.12 that changed any file in the include/scsi or drivers/scsi subdirectories

git log --since="2 weeks ago" -- gitk

Show the changes during the last two weeks to the file gitk. The -- is necessary to avoid confusion with the **branch** named gitk

git log --name-status release..test

Show the commits that are in the "test" branch but not yet in the "release" branch, along with the list of paths each commit modifies.

git log --follow builtin/rev-list.c

Shows the commits that changed builtin/rev-list.c, including those commits that occurred before the file was given its present name.

git log --branches --not --remotes=origin

Shows all commits that are in any of local branches but not in any of remote-tracking branches for origin (what you have that origin doesn’t).

git log master --not --remotes=\*/master

Shows all commits that are in local master but not in any remote repository master branches.

git log -p -m --first-parent

Shows the history including change diffs, but only from the “main branch” perspective, skipping commits that come from merged branches, and showing full diffs of changes introduced by the merges. This makes sense only when following a strict policy of merging all topic branches when staying on a single integration branch.

git log -L '/int main/',/^}/:main.c

Shows how the function main() in the file main.c evolved over time.

git log -3

Limits the number of commits to show to 3.

EXAMPLES git merge

* Merge branches fixes and enhancements on top of the current branch, making an octopus merge:

$ git merge fixes enhancements

* Merge branch obsolete into the current branch, using ours merge strategy:

$ git merge -s ours obsolete

* Merge branch maint into the current branch, but do not make a new commit automatically:

$ git merge --no-commit maint

This can be used when you want to include further changes to the merge, or want to write your own merge commit message.

You should refrain from abusing this option to sneak substantial changes into a merge commit. Small fixups like bumping release/version name would be acceptable.

## DESCRIPTION git rebase

If <branch> is specified, git rebase will perform an automatic git switch <branch> before doing anything else. Otherwise it remains on the current branch.

If <upstream> is not specified, the upstream configured in branch.<name>.remote and branch.<name>.merge options will be used (see [git-config(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-config.html) for details) and the --fork-point option is assumed. If you are currently not on any branch or if the current branch does not have a configured upstream, the rebase will abort.

All changes made by commits in the current branch but that are not in <upstream> are saved to a temporary area. This is the same set of commits that would be shown by git log <upstream>..HEAD; or by git log 'fork\_point'..HEAD, if --fork-point is active (see the description on --fork-point below); or by git log HEAD, if the --root option is specified.

The current branch is reset to <upstream> or <newbase> if the --onto option was supplied. This has the exact same effect as git reset --hard <upstream> (or <newbase>). ORIG\_HEAD is set to point at the tip of the branch before the reset.

The commits that were previously saved into the temporary area are then reapplied to the current branch, one by one, in order. Note that any commits in HEAD which introduce the same textual changes as a commit in HEAD..<upstream> are omitted (i.e., a patch already accepted upstream with a different commit message or timestamp will be skipped).

It is possible that a merge failure will prevent this process from being completely automatic. You will have to resolve any such merge failure and run git rebase --continue. Another option is to bypass the commit that caused the merge failure with git rebase --skip. To check out the original <branch> and remove the .git/rebase-apply working files, use the command git rebase --abort instead.

Assume the following history exists and the current branch is "topic":

A---B---C topic

/

D---E---F---G master

From this point, the result of either of the following commands:

git rebase master

git rebase master topic

would be:

A'--B'--C' topic

/

D---E---F---G master

**NOTE:** The latter form is just a short-hand of git checkout topic followed by git rebase master. When rebase exits topic will remain the checked-out branch.

If the upstream branch already contains a change you have made (e.g., because you mailed a patch which was applied upstream), then that commit will be skipped and warnings will be issued (if the merge backend is used). For example, running git rebase master on the following history (in which A' and A introduce the same set of changes, but have different committer information):

A---B---C topic

/

D---E---A'---F master

will result in:

B'---C' topic

/

D---E---A'---F master

Here is how you would transplant a topic branch based on one branch to another, to pretend that you forked the topic branch from the latter branch, using rebase --onto.

First let’s assume your topic is based on branch next. For example, a feature developed in topic depends on some functionality which is found in next.

o---o---o---o---o master

\

o---o---o---o---o next

\

o---o---o topic

We want to make topic forked from branch master; for example, because the functionality on which topic depends was merged into the more stable master branch. We want our tree to look like this:

o---o---o---o---o master

| \

| o'--o'--o' topic

\

o---o---o---o---o next

We can get this using the following command:

git rebase --onto master next topic

Another example of --onto option is to rebase part of a branch. If we have the following situation:

H---I---J topicB

/

E---F---G topicA

/

A---B---C---D master

then the command

git rebase --onto master topicA topicB

would result in:

H'--I'--J' topicB

/

| E---F---G topicA

|/

A---B---C---D master

This is useful when topicB does not depend on topicA.

A range of commits could also be removed with rebase. If we have the following situation:

E---F---G---H---I---J topicA

then the command

git rebase --onto topicA~5 topicA~3 topicA

would result in the removal of commits F and G:

E---H'---I'---J' topicA

This is useful if F and G were flawed in some way, or should not be part of topicA. Note that the argument to --onto and the <upstream> parameter can be any valid commit-ish.

In case of conflict, git rebase will stop at the first problematic commit and leave conflict markers in the tree. You can use git diff to locate the markers (<<<<<<) and make edits to resolve the conflict. For each file you edit, you need to tell Git that the conflict has been resolved, typically this would be done with

git add <filename>

After resolving the conflict manually and updating the index with the desired resolution, you can continue the rebasing process with

git rebase --continue

Alternatively, you can undo the git rebase with

git rebase --abort

# **git-reset Manual Page**

## NAME

git-reset - Reset current HEAD to the specified state

## SYNOPSIS

git reset [-q] [<tree-ish>] [--] <pathspec>…​

git reset [-q] [--pathspec-from-file=<file> [--pathspec-file-nul]] [<tree-ish>]

git reset (--patch | -p) [<tree-ish>] [--] [<pathspec>…​]

git reset [--soft | --mixed [-N] | --hard | --merge | --keep] [-q] [<commit>]

DEPRECATED: git reset [-q] [--stdin [-z]] [<tree-ish>]

## DESCRIPTION

In the first three forms, copy entries from <tree-ish> to the index. In the last form, set the current branch head (HEAD) to <commit>, optionally modifying index and working tree to match. The <tree-ish>/<commit> defaults to HEAD in all forms.

**git reset [-q] [<tree-ish>] [--] <pathspec>…​**

**git reset [-q] [--pathspec-from-file=<file> [--pathspec-file-nul]] [<tree-ish>]**

These forms reset the index entries for all paths that match the <pathspec> to their state at <tree-ish>. (It does not affect the working tree or the current branch.)

This means that git reset <pathspec> is the opposite of git add <pathspec>. This command is equivalent to git restore [--source=<tree-ish>] --staged <pathspec>....

After running git reset <pathspec> to update the index entry, you can use [git-restore(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-restore.html) to check the contents out of the index to the working tree. Alternatively, using [git-restore(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-restore.html) and specifying a commit with --source, you can copy the contents of a path out of a commit to the index and to the working tree in one go.

**git reset (--patch | -p) [<tree-ish>] [--] [<pathspec>…​]**

Interactively select hunks in the difference between the index and <tree-ish> (defaults to HEAD). The chosen hunks are applied in reverse to the index.

This means that git reset -p is the opposite of git add -p, i.e. you can use it to selectively reset hunks. See the “Interactive Mode” section of [git-add(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-add.html) to learn how to operate the --patch mode.

**git reset [<mode>] [<commit>]**

This form resets the current branch head to <commit> and possibly updates the index (resetting it to the tree of <commit>) and the working tree depending on <mode>. If <mode> is omitted, defaults to --mixed. The <mode> must be one of the following:

**--soft**

Does not touch the index file or the working tree at all (but resets the head to <commit>, just like all modes do). This leaves all your changed files "Changes to be committed", as git status would put it.

**--mixed**

Resets the index but not the working tree (i.e., the changed files are preserved but not marked for commit) and reports what has not been updated. This is the default action.

If -N is specified, removed paths are marked as intent-to-add (see [git-add(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-add.html)).

**--hard**

Resets the index and working tree. Any changes to tracked files in the working tree since <commit> are discarded. Any untracked files or directories in the way of writing any tracked files are simply deleted.

**--merge**

Resets the index and updates the files in the working tree that are different between <commit> and HEAD, but keeps those which are different between the index and working tree (i.e. which have changes which have not been added). If a file that is different between <commit> and the index has unstaged changes, reset is aborted.

In other words, --merge does something like a git read-tree -u -m <commit>, but carries forward unmerged index entries.

**--keep**

Resets index entries and updates files in the working tree that are different between <commit> and HEAD. If a file that is different between <commit> and HEAD has local changes, reset is aborted.

**--[no-]recurse-submodules**

When the working tree is updated, using --recurse-submodules will also recursively reset the working tree of all active submodules according to the commit recorded in the superproject, also setting the submodules' HEAD to be detached at that commit.

See "Reset, restore and revert" in [git(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git.html) for the differences between the three commands.

# **git-revert Manual Page**

## NAME

git-revert - Revert some existing commits

## SYNOPSIS

git revert [--[no-]edit] [-n] [-m parent-number] [-s] [-S[<keyid>]] <commit>…​

git revert (--continue | --skip | --abort | --quit)

## DESCRIPTION

Given one or more existing commits, revert the changes that the related patches introduce, and record some new commits that record them. This requires your working tree to be clean (no modifications from the HEAD commit).

Note: git revert is used to record some new commits to reverse the effect of some earlier commits (often only a faulty one). If you want to throw away all uncommitted changes in your working directory, you should see [git-reset(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-reset.html), particularly the --hard option. If you want to extract specific files as they were in another commit, you should see [git-restore(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-restore.html), specifically the --source option. Take care with these alternatives as both will discard uncommitted changes in your working directory.

See "Reset, restore and revert" in [git(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git.html) for the differences between the three commands.

## EXAMPLES

git revert HEAD~3

Revert the changes specified by the fourth last commit in HEAD and create a new commit with the reverted changes.

git revert -n master~5..master~2

Revert the changes done by commits from the fifth last commit in master (included) to the third last commit in master (included), but do not create any commit with the reverted changes. The revert only modifies the working tree and the index.

# **git-cherry-pick Manual Page**

## NAME

git-cherry-pick - Apply the changes introduced by some existing commits

## SYNOPSIS

git cherry-pick [--edit] [-n] [-m <parent-number>] [-s] [-x] [--ff]

[-S[<keyid>]] <commit>…​

git cherry-pick (--continue | --skip | --abort | --quit)

## DESCRIPTION

Given one or more existing commits, apply the change each one introduces, recording a new commit for each. This requires your working tree to be clean (no modifications from the HEAD commit).

When it is not obvious how to apply a change, the following happens:

1. The current branch and HEAD pointer stay at the last commit successfully made.
2. The CHERRY\_PICK\_HEAD ref is set to point at the commit that introduced the change that is difficult to apply.
3. Paths in which the change applied cleanly are updated both in the index file and in your working tree.
4. For conflicting paths, the index file records up to three versions, as described in the "TRUE MERGE" section of [git-merge(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-merge.html). The working tree files will include a description of the conflict bracketed by the usual conflict markers <<<<<<< and >>>>>>>.
5. No other modifications are made.

See [git-merge(1)](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-merge.html) for some hints on resolving such conflicts.

EXAMPLES

git cherry-pick master

Apply the change introduced by the commit at the tip of the master branch and create a new commit with this change.

git cherry-pick ..master

git cherry-pick ^HEAD master

Apply the changes introduced by all commits that are ancestors of master but not of HEAD to produce new commits.

git cherry-pick maint next ^master

git cherry-pick maint master..next

Apply the changes introduced by all commits that are ancestors of maint or next, but not master or any of its ancestors. Note that the latter does not mean maint and everything between master and next; specifically, maint will not be used if it is included in master.

git cherry-pick master~4 master~2

Apply the changes introduced by the fifth and third last commits pointed to by master and create 2 new commits with these changes.

git cherry-pick -n master~1 next

Apply to the working tree and the index the changes introduced by the second last commit pointed to by master and by the last commit pointed to by next, but do not create any commit with these changes.

git cherry-pick --ff ..next

If history is linear and HEAD is an ancestor of next, update the working tree and advance the HEAD pointer to match next. Otherwise, apply the changes introduced by those commits that are in next but not HEAD to the current branch, creating a new commit for each new change.

git rev-list --reverse master -- README | git cherry-pick -n --stdin

Apply the changes introduced by all commits on the master branch that touched README to the working tree and index, so the result can be inspected and made into a single new commit if suitable.

The following sequence attempts to backport a patch, bails out because the code the patch applies to has changed too much, and then tries again, this time exercising more care about matching up context lines.

$ git cherry-pick topic^ **(1)**

$ git diff **(2)**

$ git reset --merge ORIG\_HEAD **(3)**

$ git cherry-pick -Xpatience topic^ **(4)**

1. apply the change that would be shown by git show topic^. In this example, the patch does not apply cleanly, so information about the conflict is written to the index and working tree and no new commit results.
2. summarize changes to be reconciled
3. cancel the cherry-pick. In other words, return to the pre-cherry-pick state, preserving any local modifications you had in the working tree.
4. try to apply the change introduced by topic^ again, spending extra time to avoid mistakes based on incorrectly matching context lines.

# **git-checkout Manual Page**

## NAME

git-checkout - Switch branches or restore working tree files

## SYNOPSIS

git checkout [-q] [-f] [-m] [<branch>]

git checkout [-q] [-f] [-m] --detach [<branch>]

git checkout [-q] [-f] [-m] [--detach] <commit>

git checkout [-q] [-f] [-m] [[-b|-B|--orphan] <new-branch>] [<start-point>]

git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [<tree-ish>] [--] <pathspec>…​

git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [<tree-ish>] --pathspec-from-file=<file> [--pathspec-file-nul]

git checkout (-p|--patch) [<tree-ish>] [--] [<pathspec>…​]

## DESCRIPTION

Updates files in the working tree to match the version in the index or the specified tree. If no pathspec was given, git checkout will also update HEAD to set the specified branch as the current branch.

**git checkout [<branch>]**

To prepare for working on <branch>, switch to it by updating the index and the files in the working tree, and by pointing HEAD at the branch. Local modifications to the files in the working tree are kept, so that they can be committed to the <branch>.

If <branch> is not found but there does exist a tracking branch in exactly one remote (call it <remote>) with a matching name and --no-guess is not specified, treat as equivalent to

$ git checkout -b <branch> --track <remote>/<branch>

You could omit <branch>, in which case the command degenerates to "check out the current branch", which is a glorified no-op with rather expensive side-effects to show only the tracking information, if exists, for the current branch.

**git checkout -b|-B <new-branch> [<start-point>]**

Specifying -b causes a new branch to be created as if [git-branch(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-branch.html) were called and then checked out. In this case you can use the --track or --no-track options, which will be passed to git branch. As a convenience, --track without -b implies branch creation; see the description of --track below.

If -B is given, <new-branch> is created if it doesn’t exist; otherwise, it is reset. This is the transactional equivalent of

$ git branch -f <branch> [<start-point>]

$ git checkout <branch>

that is to say, the branch is not reset/created unless "git checkout" is successful.

**git checkout --detach [<branch>]**

**git checkout [--detach] <commit>**

Prepare to work on top of <commit>, by detaching HEAD at it (see "DETACHED HEAD" section), and updating the index and the files in the working tree. Local modifications to the files in the working tree are kept, so that the resulting working tree will be the state recorded in the commit plus the local modifications.

When the <commit> argument is a branch name, the --detach option can be used to detach HEAD at the tip of the branch (git checkout <branch> would check out that branch without detaching HEAD).

Omitting <branch> detaches HEAD at the tip of the current branch.

**git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [<tree-ish>] [--] <pathspec>…​**

**git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [<tree-ish>] --pathspec-from-file=<file> [--pathspec-file-nul]**

Overwrite the contents of the files that match the pathspec. When the <tree-ish> (most often a commit) is not given, overwrite working tree with the contents in the index. When the <tree-ish> is given, overwrite both the index and the working tree with the contents at the <tree-ish>.

The index may contain unmerged entries because of a previous failed merge. By default, if you try to check out such an entry from the index, the checkout operation will fail and nothing will be checked out. Using -f will ignore these unmerged entries. The contents from a specific side of the merge can be checked out of the index by using --ours or --theirs. With -m, changes made to the working tree file can be discarded to re-create the original conflicted merge result.

**git checkout (-p|--patch) [<tree-ish>] [--] [<pathspec>…​]**

This is similar to the previous mode, but lets you use the interactive interface to show the "diff" output and choose which hunks to use in the result. See below for the description of --patch option.

## EXAMPLES

1. The following sequence checks out the master branch, reverts the Makefile to two revisions back, deletes hello.c by mistake, and gets it back from the index.
2. $ git checkout master **(1)**
3. $ git checkout master~2 Makefile **(2)**
4. $ rm -f hello.c

$ git checkout hello.c **(3)**

* 1. switch branch
  2. take a file out of another commit
  3. restore hello.c from the index

If you want to check out all C source files out of the index, you can say

$ git checkout -- '\*.c'

Note the quotes around \*.c. The file hello.c will also be checked out, even though it is no longer in the working tree, because the file globbing is used to match entries in the index (not in the working tree by the shell).

If you have an unfortunate branch that is named hello.c, this step would be confused as an instruction to switch to that branch. You should instead write:

$ git checkout -- hello.c

1. After working in the wrong branch, switching to the correct branch would be done using:

$ git checkout mytopic

However, your "wrong" branch and correct mytopic branch may differ in files that you have modified locally, in which case the above checkout would fail like this:

$ git checkout mytopic

error: You have local changes to 'frotz'; not switching branches.

You can give the -m flag to the command, which would try a three-way merge:

$ git checkout -m mytopic

Auto-merging frotz

After this three-way merge, the local modifications are not registered in your index file, so git diff would show you what changes you made since the tip of the new branch.

1. When a merge conflict happens during switching branches with the -m option, you would see something like this:
2. $ git checkout -m mytopic
3. Auto-merging frotz
4. ERROR: Merge conflict in frotz

fatal: merge program failed

At this point, git diff shows the changes cleanly merged as in the previous example, as well as the changes in the conflicted files. Edit and resolve the conflict and mark it resolved with git add as usual:

$ edit frotz

$ git add frotz

# **git-commit Manual Page**

## NAME

git-commit - Record changes to the repository

## SYNOPSIS

git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [--amend]

[--dry-run] [(-c | -C | --squash) <commit> | --fixup [(amend|reword):]<commit>)]

[-F <file> | -m <msg>] [--reset-author] [--allow-empty]

[--allow-empty-message] [--no-verify] [-e] [--author=<author>]

[--date=<date>] [--cleanup=<mode>] [--[no-]status]

[-i | -o] [--pathspec-from-file=<file> [--pathspec-file-nul]]

[(--trailer <token>[(=|:)<value>])…​] [-S[<keyid>]]

[--] [<pathspec>…​]

## DESCRIPTION

Create a new commit containing the current contents of the index and the given log message describing the changes. The new commit is a direct child of HEAD, usually the tip of the current branch, and the branch is updated to point to it (unless no branch is associated with the working tree, in which case HEAD is "detached" as described in [git-checkout(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-checkout.html)).

The content to be committed can be specified in several ways:

1. by using [git-add(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-add.html) to incrementally "add" changes to the index before using the commit command (Note: even modified files must be "added");
2. by using [git-rm(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-rm.html) to remove files from the working tree and the index, again before using the commit command;
3. by listing files as arguments to the commit command (without --interactive or --patch switch), in which case the commit will ignore changes staged in the index, and instead record the current content of the listed files (which must already be known to Git);
4. by using the -a switch with the commit command to automatically "add" changes from all known files (i.e. all files that are already listed in the index) and to automatically "rm" files in the index that have been removed from the working tree, and then perform the actual commit;
5. by using the --interactive or --patch switches with the commit command to decide one by one which files or hunks should be part of the commit in addition to contents in the index, before finalizing the operation. See the “Interactive Mode” section of [git-add(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-add.html) to learn how to operate these modes.

The --dry-run option can be used to obtain a summary of what is included by any of the above for the next commit by giving the same set of parameters (options and paths).

If you make a commit and then find a mistake immediately after that, you can recover from it with git reset.

# **git-reflog Manual Page**

## NAME

git-reflog - Manage reflog information

## SYNOPSIS

git reflog <subcommand> <options>

## DESCRIPTION

The command takes various subcommands, and different options depending on the subcommand:

git reflog [show] [<log-options>] [<ref>]

git reflog expire [--expire=<time>] [--expire-unreachable=<time>]

[--rewrite] [--updateref] [--stale-fix]

[--dry-run | -n] [--verbose] [--all [--single-worktree] | <refs>…​]

git reflog delete [--rewrite] [--updateref]

[--dry-run | -n] [--verbose] <ref>@{<specifier>}…​

git reflog exists <ref>

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. See [gitrevisions(7)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\gitrevisions.html) for more details.

This command manages the information recorded in the reflogs.

The "show" subcommand (which is also the default, in the absence of any subcommands) shows the log of the reference provided in the command-line (or HEAD, by default). The reflog covers all recent actions, and in addition the HEAD reflog records branch switching. git reflog show is an alias for git log -g --abbrev-commit --pretty=oneline; see [git-log(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-log.html) for more information.

The "expire" subcommand prunes older reflog entries. Entries older than expire time, or entries older than expire-unreachable time and not reachable from the current tip, are removed from the reflog. This is typically not used directly by end users — instead, see [git-gc(1)](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-gc.html).

The "delete" subcommand deletes single entries from the reflog. Its argument must be an exact entry (e.g. "git reflog delete master@{2}"). This subcommand is also typically not used directly by end users.

The "exists" subcommand checks whether a ref has a reflog. It exits with zero status if the reflog exists, and non-zero status if it does not.

**Git : #1 Apa Itu Git & GitHub**

**Git : #2 Bekerja Dengan GitHub**

**Git : #3 GitHub Branch**

**Git : #4 GitHub Fork**

**Git : #5 Bekerja Dengan Git**

**Git : #6 Git Branch & Merge**

**Git : #7 Git Merge Conflict**

**Git : #8 Git Remote (Ini Paling Penting !!!)**

**Git : #9 GitHub Pages, Web Hosting Gratis dengan GitHub Pages**

**Git : #10 Multiple Remote**

**Git : #11 Remote Branch**

**Git : #12 .gitignore**

**Git : #13 Rebase**

**Git : #14 Git & Web Hosting**

[**https://medium.com/@gofrendiasgard/git-perjalanan-menuju-dunia-parallel-c4e74364cf84**](https://medium.com/@gofrendiasgard/git-perjalanan-menuju-dunia-parallel-c4e74364cf84)

**Git : #1 Apa Itu Git & GitHub**

**Konsep** : Git, Version Control System (VCS) atau Source Code Management (SCM)

**Defisi** : Sistem yang mengelola perubahan dari sebuat dokumen, program kompute, website dan kumpulan informasi lain.

**Kenapa kita harus memakai VCS :**

1. Bisa melacak versi atau history dari perubahan yang terjadi pada software kita
2. Kolaborasi
3. Sharring, membagikan atau memamerkan program yang sudah kita buat

**Git & GitHub merupakan 2 hal berbeda**, bisa bekerja dengan Git tanpa harus bekerja dengan GitHub, dan sebaliknya bisa bekerja dengan GitHub tanpa harus menginstall Git.

**Version Control System :**

* Sebuah sistem yang menyimpan “rekaman/snapshot” perubahan pada source code
* Memungkinkan bekerja berkolaborasi dengan lebih baik
* Mengetahui siapa yang melakukan dan kapan sebuat perubahan terjadi
* Memungkinkan kita untk kembali ke keadaan sebelum perubahan (checkout)

**Definisi Git :** Sebuah VCS terdistribusi untuk mengelola perubahan file di dalam folder

**Repository/Repo :** Folder pada sebuat aplikasi

Riwayat perubahan file disimpan menggunakan serangkaian **commit**

**Branch :** cabang di Git

**Merge** **:** gabung

**GitHub :** Layanan cloud untuk menyimpan & mengelola project/repo git. Instagramnya para programmer. Seperti layaknya Git, tapi ini online.

**Akan lebih terasa jika kita menggabungkan keduanya, Git & GitHub.**

**Kita bisa mengirimkan Source Code kita atau Project kita ke Git Hub (Push) atau sebaliknya kita bisa mengambil Source Code kita atau Project kita dari GitHub (Pull). Yang di Push dan di Pull adalah commit-nya.**

**Syaratnya adalah : GitHub dijadikan Remote. Lalu Clone. Baru bisa lakukan Push dan Pull.**

Akan sangat bermanfaat jika bekerja dengan kolaborasi.

**Git Command/Istilah**

* repo : folder project kita
* commit : rekaman/snapshot dari repo kita
* hash : penanda unik pada sebuah commit
* checkout : berpindah ke sebuah commit
* branch : cabang bebas dari sebuah commit
* merge : menggabungkan branch
* remote : sumber yang memiliki repo
* clone : mengambil repo dari remot
* push : mengirimkan commit ke repo
* pull : mengambil commit dari repo

**Git : #2 Bekerja Dengan GitHub**

Create Account : ariefcu

Email Account : [ariefcu@gmail.com](mailto:ariefcu@gmail.com)

Nama Repo Private : coba-laravel

Nama Repo Public : wpu-resolusi

**Git : #3 GitHub Branch**

Create Account : ariefcu

Email Account : [ariefcu@gmail.com](mailto:ariefcu@gmail.com)

Nama Repo Private : coba-laravel

Nama Repo Public : wpu-resolusi

Checkout untuk pindah cabang

**Pull Request** untuk meminta pemilik dari reponya atau main branchnya untuk menarik data kita

**Merge** untuk menggabungkan 2 cabang

**Merge Conflict** karena 2 baris yang sama dirubah 2 cabang yang berbeda

**Git : #4 GitHub Fork**

**Fork/Forking :**

* Membuat “copy/duplicat” dari repo orang lain (beserta historynya)
* Jembatan antara repo original dan duplikatnya
* Modifikasi terhadap repo original
* Berkontribusi pada repo orang lain
* Fork != Clone
* Karena Clone hanya meduplicat dari GitHub ke komputer lokal kita

**Mencoba fork dari webprogrammingunpas dan lakukan pull request.**

**Test kolaborasi.**

**Git : #5 Bekerja Dengan Git**

Install dari[**https://git-scm.com/**](https://git-scm.com/)

**Console**

**Git Client** (GUI – Graphical User Interface)

**Pro Git eBook :**

htpps://git-scm.com/book/en/v2 for english

htpps://git-scm.com/book/id/v2 for bahasa

Banyak yang pakai Git, antara lain : Google, facebook, Microsoft, Twitter dll

Git Command (local) :

* git init : inisialisasi repo git di komputer kita
* git add <file(s)> : buat nambahin file di staging area
* git status : penting banget, status repo kita
* git commit : untuk melakukan commit
* git config : untuk melakukan configurasi ke dalam gitnya
* git branch : untuk bikin branch
* git help : untuk mau tau perintah

3 area pada repo Git

* **Working Tree** : folder lokal tempat kita bekerja, tempat file-file project
* **Staging Area** : ngasih tau ke Git kita melakukan perubahan
* **History**  : setelah di commit masuk ke sini

Staging Area dan History akan masuk ke folder **.git**

**git add** untukmemindahkan dari Working Area ke Staging Area

Atau pakai **git add .** untuk add semua file

**git commit** untuk memindahkan dari Staging Area ke History

**git status** untuk mengecek kondisi sebelum **commit**

**git config** –global user.name “Ariefcu”

**git config** –global user.email [ariefcu@gmail.com](mailto:ariefcu@gmail.com)

**git commit -m** “commit pertama kali ke repo”

-m diatas untuk memberi message

**git log** untuk melihat commit2 yang sudah pernah dilakukan

**git log -3** untuk commit2 3 terakhir

**git log – style.css** untuk melihat perubahan yang terkair file style.css

**git checkout 5digitawaldarihash – style.css** untuk kembali ke posisi tertentu terkait file style.css

kalau ada file yang di staging berarti harus di commit dahulu biar masuk ke history

**Git : #6 Git Branch & Merge**

**pwd : print working directory,** untuk melihat kita lagi di directory mana

**Tarik folder ke git bash** biar pindah direktorinya ke situ, atau

Buka folder yang kita mau repo lalu **klik kanan dengan buka Git Bash disini**

**git init**, maka foldernya akan berubah menjadi reposity

**git commit -am “menambahkan repo”** -a adalah untuk add jika filenya sudah pernah ditambahan

**head** menandakan branch sedang aktif di branch tersebut

**git branch** untuk melihat branch ada apa aja

**cabang** yang warna hijau adalah yang aktif

**git branch about** untuk membuat cabang about

**git branch post** untuk membuat cabang post

**git branch** maka muncul ada 3, about, master dan post, master tulisan hijau dan ada bintangnya

**git log –all –decorate –oneline –graph** untuk melihat visualisasi graphnya

**alias g=”git log –all –decorate –oneline –graph”** untuk membuat shortcut, hanya berlaku 1 sesi

**git checkout <nama\_branch>** untuk pindah headnya ke nama\_branch

**git checkout about** maka head akan pindah ke about

**merge ada 2 jenis :**

* **Fast Forward Merge**, yang direct path
* **Three-way Merge**

1. Menggabungkan master dengan about

**git checkout master**

**git branch**, memastikan pindah ke master

**git merge about**, Fast-forward

**git branch –merged**, untuk mengetahui branch mana yang sudah di merged

**git branch -d about**, untuk mendelete branch about

**git branch -d post**, error: **The Branch ‘post’ is not fully merged**, if you are sure you want to delete it, run **‘git branch -D post**’

1. Menggabungkan master dengan post

merge antara master dan post tidak bisa langsung di merge, karena **tidak direct path**

**git branch** untuk memastikan di master

**git merge post**, merge made by the ‘ort’ strategy

**Git : #7 Git Merge Conflict**

**Karena 2 branch mengerjakan baris yang sama dalam satu repo.**

**Harus manual untuk solusinya.**

**Sama seperti saat merge, hanya saja ada yang harus diselesaikan secara manual untu solusinya, tidak bisa menggunakan auto merge**

**Git : #8 Git Remote (Paling Penting !!!)**

**Kasus #1 Di Komputer Belum Ada, Mau Mengambil Dari GitHub**

**Git Remote bisa ke GitHub, GitLab atau Bucket**

**Remote intinya adalah duplikat dari punya kita**

Kali ini kita akan pakai **GitHub sebagai remote kita**

Buat repository baru di GitHub, misal test-remote-2

Lalu copy alamat HTTPSnya

Masuk ke folder d:\applications

Lalu **git clone** [**https://github.com/ariefcu/test-remote-2.git**](https://github.com/ariefcu/test-remote-2.git)

**Maka sudah terhubung secara remote**

Cd test-remote-2

**git remote**, hasilnya origin

**git remote -v**,

git remote -v

origin https://github.com/ariefcu/test-remote-2.git (fetch)

origin https://github.com/ariefcu/test-remote-2.git (push)

**Git Status**, on branch master, **your branch is up to date with ‘origin/master’**

**alias g=”git log –all –decorate –oneline –graph”** untuk membuat shortcut, hanya berlalu 1 sesi

**graph**

di vscode buat file baru index.html, lalu save

kemudian di git bash ketik **git bash**

lalu **git add .**

lalu **git commit -m “menambah file index.html”**

**git status**,

On branch main

Your branch is ahead of 'origin/main' by 1 commit.

(use "git push" to publish your local commits)

nothing to commit, working tree clean

**graph**,

\* 90197dd (HEAD -> main) menambahkan file index.html

\* 12b9e23 (origin/main, origin/HEAD) Initial commit

**git push**

**Masukkan password jika diminta**

Enumerating objects: 4, done.

Counting objects: 100% (4/4), done.

Delta compression using up to 8 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 477 bytes | 477.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

To https://github.com/ariefcu/test-remote-2.git

12b9e23..90197dd main -> main

**git status**

On branch main

Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

**graph**

\* 90197dd (HEAD -> main, origin/main, origin/HEAD) menambahkan file index.html

\* 12b9e23 Initial commit

**git config –global user.name** “Ariefcu”

**git config –global user.email** “[ariefcu@gmail.com](mailto:ariefcu@gmail.com)”

**Menit 18:00 Kasus #2 di Lokal sudah ada, di GitHub belum ada, kirim dari Lokal ke GitHub**

Keluar dari repository dengan **CD ..**

**mkdir** test-remote-3

**cd** test-remote-3

**git init**

Initialized empty Git repository in D:/applications/test-remote-3/.git/

Di **vscode** buat file index.html

**ls** untuk melihat file yang ada

**git status**

**git add .**

**git commit -m “menambahkan file index.html”**

**graph**

**clear**

Di **vscode** update file index.html

**git status**

**git commit -am “mengupdate file index.html”**

**graph**

\* 0b3bbea (HEAD -> master) mengupdate file index.html

\* 58e2474 menambahkan file index.html

**Ceritanya mau manambahkan repository ini ke account GitHub kita**

Di **GitHub create repository baru** dengan **nama yang sama test-remote-3**

**Initialize this repository with a README jangan dicentang**

**git remote**, masih kosong

**Copy dari lokal ke GitHub :**

**git remote add origin https://github.com/ariefcu/test-remote-3.git**

**git remote**

origin

**git remote -v**

origin https://github.com/ariefcu/test-remote-3.git (fetch)

origin https://github.com/ariefcu/test-remote-3.git (push)

**git push -u origin master**

Enumerating objects: 6, done.

Counting objects: 100% (6/6), done.

Delta compression using up to 8 threads

Compressing objects: 100% (4/4), done.

Writing objects: 100% (6/6), 671 bytes | 671.00 KiB/s, done.

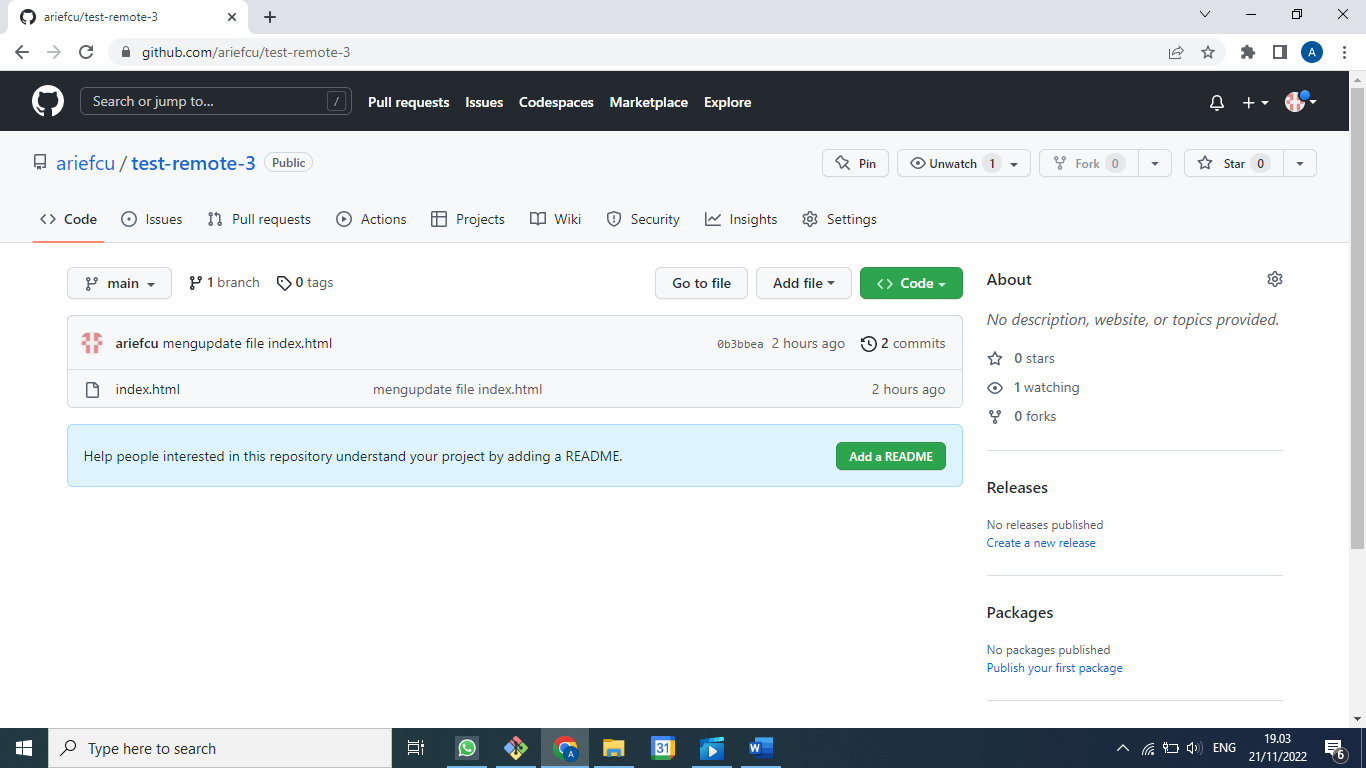
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (1/1), done.

To https://github.com/ariefcu/test-remote-3.git

\* [new branch] main -> main

branch 'main' set up to track 'origin/main'.



**Di vscode rubah index.html**

**Save**

**Lalu di Git Bash**

**graph**

**git Add .**

**git Commit -am “mengupdate index.html lagi”**

**git push**

**git graph**

**Kasus #3 Di Komputer Sudah Ada, Di GitHub Sudah Ada, Tapi Ada Perubahan Di Masing2 File Index.html Di Baris Yang Sama**

**Ubah file index.html di komputer di baris yang sama**

**Ubah file index.html di GitHub di baris yang sama lalu di commit**

**Di Git Bash**

**clear**

**git add .**

**git commit -m “mengubah file index.html”**

**git push**

To https://github.com/ariefcu/test-remote-3.git

! [rejected] main -> main (fetch first)

error: failed to push some refs to 'https://github.com/ariefcu/test-remote-3.git'

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.

**clear**

**git status**

On branch main

Your branch is ahead of 'origin/main' by 1 commit.

(use "git push" to publish your local commits)

nothing to commit, working tree clean

**clear**

**git fetch**, dia akan ngecek repo yang ada di remote sudah sampai mana commitnya, beda ga sama yang kita punya

**git status**

On branch main

Your branch is ahead of 'origin/main' by 1 commit.

(use "git push" to publish your local commits)

nothing to commit, working tree clean

**graph**

\* 28cff3a (HEAD -> main) mengubah file index.html

| \* 2a4bdcb (origin/main) Update index.html

|/

\* 4bdd337 mengupdate index.html lagi

\* 0b3bbea mengupdate file index.html

\* 58e2474 menambahkan file index.html

**git pull**

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

**Di vscode :**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Test Git Remote</title>

</head>

<body>

<<<<<<< HEAD

    <h1>Tes Remote Merge Conflict</h1>

=======

    <h1>Ayo Semangat Pelajari Remote Merge Conflict</h1>

>>>>>>> 2a4bdcb33a59f495fd815b8d86350f9e7ec822d4

    <p>Paragraph X</p>

</body>

</html>

Solusinya jadi seperti ini :

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Test Git Remote</title>

</head>

<body>

    <h1>Tes Remote Merge Conflict Ayo Semangat Pelajari Remote Merge Conflict</h1>

    <p>Paragraph X</p>

</body>

</html>

**git status**

**clear**

**git add .**

**git commit -m “tahapan merge conflict”**

**git status**

**git push**

**git status**

\* 9dd0d1e (HEAD -> main, origin/main) tahapan merge conflic

|\

| \* 2a4bdcb Update index.html

\* | 28cff3a mengubah file index.html

|/

\* 4bdd337 mengupdate index.html lagi

\* 0b3bbea mengupdate file index.html

\* 58e2474 menambahkan file index.html

**Git : #9 GitHub Pages, Web Hosting Gratis dengan GitHub Pages**

Membuat salah satu repo menjadi GitHub Pages

Harus punya repo website statis

Github.com/sandhikagalih, website 3 awal bisa digunakan

1. Bostrap v3
2. Boostrap v4, ngobar 4
3. Company Profile, Materilize

Yang pertama disiapkan adalah websitenya dulu

Bikin dulu repo dulu

New repo, namanya ga boleh sembarang :

ariefcu.github.io

di explorer masuk ke folder company profile, git bash here

**git init**

**git remote add origin** [**https://github.co/ariefcu/ariefcu.githbub.io.git**](https://github.co/ariefcu/ariefcu.githbub.io.git)

**git remote**

**clear**

**git add .**

**git commit -m “inisialisasi commit”**

**git status**

**git push -u origin master**

**git status**

di github sudah terupload

di browser : ariefcu.github.io

kalau mau update tinggal di lokal lalu di push

kalau mau .com tinggal arahkan di githubnya

**Kasus #2 Bikin halaman website untuk masing2 repo**

Alamat urlnya tetap di bawah alamat ariefcu.github.io

**cd ..**

**ls**

**cd wpu-landing**

**git init**

**git status**

**git add .**

**git commit -m “inisial”**

**git status**

Di github repo new namanya bebas wpu-landing

Di github

**git remore add origin copyan dari github**

**git push -u origin master**

**git status**

Di github sudah ada website siap tampil, pastikan index.html harus ada

Scroll sampai github pages, di source, branch yang mana yang mau dijadikan default

Pilih yang master branch

Setelah loading balik lagi ke github pages

Your site is readu to be published at <https://ariefcu.github.io/wapu-landing/>

Kalau diklik otomatis website kalian sudah tampil

Jadi website gratisan, bisa diakses oleh siapapun

Kalau untuk ganti ke .com bisa custome domain

Kalau untuk PHP pakai theme

**Git : #10 Multiple Remote**

Kita bisa terhubung ke beberapa remote sekaligus

Kenapa? Mungkin aja kita ingin singkron dengan beberapa remote sekaligus

Fork punya repo orang lain, misal github/sandhikagalih/simple-landing-page

Lalu copy ke lokal

Tentukan dulu di mana mau copynya, d:\applications

**git clone** [**https://github.com/ariefcu/simple-landing-page.git**](https://github.com/ariefcu/simple-landing-page.git)

**ls**

**cd simple-landing-page**

**git remote**

**git remote -v**

**git remote add sandhikagalih** [**https://github.com/asandhikagalih/simple-landing-page**](https://github.com/asandhikagalih/simple-landing-page.)

**clear**

**graph**

**git fetch sandhikagalih**

**graph**

**ganti index.html di sandhikagalih**

**git status**

**git add .**

**git commit -m “mengubah warna tombol”**

**git push -u origin master**

**git status**

Kalau buka github reponya sudah tambah commit

Kembali ke ariefcu

**git status**

Katanya up to date aja, jadi harus feth

**git fetch sandhikagalih**

**git status**

**graph**

**git merge sandhikagalih/master**

**graph**

Lokal kita sudah sama dengan github sandhikagalih tapi github kita belum sama

Maka untuk menyamakan github kita dengan lokal dan sandhikagalih, caranya :

**git push -u origin master**

**clear**

**graph**

Sekarang ketiganya sudah sama lagi

Sudah menskronkan 3 repo kita

Lokal dan 2 remote yang berbeda

**Git : #11 Remote Branch**

Yaitu bagaimana saat kita membuat branch di repo kita, lalu branch tersebut kita usulkan perubahannya ke repo sumbernya atau repo aslinya, di dalamnya kita akan belajar pull request, bagaimana perubahan kita agar disetujui diterima oleh pemilik repo aslinya

Di lokal bikin branch dulu, agar master selalu sama dengan repo aslinya

**git branch**, Cuma ada master

**git branch features** setelah itu kalian checkout untuk pindah ke branchnya

Atau bisa pakai ini

**git checkout -b features**

Kalau ini kalau branchnya belum ada dibikn dulu dan langsung pindah

Kalau mau satu2 :

**it branch features**

**git branch**

Sudah ada,

Agar pindah ke features

**git checkout features**

Kalau kalian lihat lagi di vscodenya paling kiri bawah sudah berada di branch di fatures

Kembali rubah sedikit di index.html

**git status**

**git add .**

**git commit -m “rubah dikit index.html”**

**gti status**

**graph**

Branch features kita maju 1 commit lainnya masih di belakang

**git push origin features**

**clear**

**graph**

Sudah sama dengan github kita

Sekarang kita usulkan pull request di github kita karena pull request Cuma ada di github

Klik compare & pull request

Klik create pull request

Sampai sini tugas kita sudah selesai, tinggal menuggu sandhika galih untuk menyetujui dan menerima request kita

Di github sandhikagalih, lihat pull request dari kita, lihat detilnya files changed

Kalau sudah ok, bisa review changes untuk diskusi atau komentar terhadap pull request

Merge pull requetsnya, confirm merge

Di master sekarang commitnya nambah 2, merge dan perubahan yang dilakukan oleh orang yang pull request

Di github ariefcu kalau sudah disetujui

Berarti tinggal menyakan lagi yang punya sandhikagalih dengan punya kita

**git fetch sandhikagalih**

**graph**

**git branch**

**git checkout master**

**git merge sandhikagalih/master**

**graph**

Lokal kita sudah sama dengan sandhikagalih/master

Yang belum sama adalah di github punya kita

**git push origin master**

**graph**

Sudah sama semua

Tinggal kalau kita sudah tidak butuh branch features maka bisa kita delete, karena tugasnya adalah untuk pull request saja

**git branch -d features**

**git branch**

Sudah hilang

**graph**

Ternyata di github kita masih ada branch features, kalau mau dihilangkan juga

**git push origin ---d features**

Branchnya sudah terhapus

**Graph**

Sudah tidak ada branch featuresnya

Kalau memastikan bisa dilihat di github kita

**Git : #12 .gitignore**

Sebuah file yang bisa kita simpan di dalam repository git kita agar kedepannya pada saat kita melakukan add dan commit ada file yang tidak ikut terbawa ke dalam addnya

Code . untuk buka vscode

Bisa menyimpan di folder .gitignore :

1. Nama file, contoh : config.txt
2. Folder, contoh : data/
3. Pola, contoh : \*.exe

Sebagai contoh lain adalah config untuk lokal, ga usah dibawa commit

Git push, yang kedua tidak perlu pakai -u karena upstreamnya sudah dianggap seperti yang kita set sebelumnya

Untuk detilnya bisa dilihat : github/gitignore

Atau bisa mudah dengan website gitignore.io dibuatkan rekomendarinya tinggal dimsukkan ke file .ignore, sebagai contoh parameternya adalah windows +, visualcodestudio + laravel, hasilnya adalah sbb :

# Created by https://www.toptal.com/developers/gitignore/api/windows,visualstudiocode,laravel

# Edit at https://www.toptal.com/developers/gitignore?templates=windows,visualstudiocode,laravel

### Laravel ###

/vendor/

node\_modules/

npm-debug.log

yarn-error.log

# Laravel 4 specific

bootstrap/compiled.php

app/storage/

# Laravel 5 & Lumen specific

public/storage

public/hot

# Laravel 5 & Lumen specific with changed public path

public\_html/storage

public\_html/hot

storage/\*.key

.env

Homestead.yaml

Homestead.json

/.vagrant

.phpunit.result.cache

### VisualStudioCode ###

.vscode/\*

!.vscode/settings.json

!.vscode/tasks.json

!.vscode/launch.json

!.vscode/extensions.json

!.vscode/\*.code-snippets

# Local History for Visual Studio Code

.history/

# Built Visual Studio Code Extensions

\*.vsix

### VisualStudioCode Patch ###

# Ignore all local history of files

.history

.ionide

### Windows ###

# Windows thumbnail cache files

Thumbs.db

Thumbs.db:encryptable

ehthumbs.db

ehthumbs\_vista.db

# Dump file

\*.stackdump

# Folder config file

[Dd]esktop.ini

# Recycle Bin used on file shares

$RECYCLE.BIN/

# Windows Installer files

\*.cab

\*.msi

\*.msix

\*.msm

\*.msp

# Windows shortcuts

\*.lnk

# End of https://www.toptal.com/developers/gitignore/api/windows,visualstudiocode,laravel

**Git : #13 Rebase**

Rebase adalah salah satu cara untuk memilih git workflow, sebelumnya ktia selalu pakai merging.

Di github kita,

pilih setting,

pilih collaborators,

masukkan siapa yang mau dimasukkan

accept dulu oleh orang yang dimasukkan

di gituhub yang kita undang, accept

jadi bisa rubah program tanpa harus pull request lagi

lalu di git bash orang yang kita undang

**git clone** [**https://github.com/ariefcu/coba-laravel.git**](https://github.com/ariefcu/coba-laravel.git)

cd coba-laravel

**clear**

**git status**

sudah up to date dengan masternya

biasakan untuk merubah sesuatu buat branch dulu

**git brach fitur\_keren**

**lalu git checkout fitur\_keren**

atau bisa langsung

**git checkout -b fitur\_keren**

langsung switch ke branch baru

**ubah index.html**

**git status**

**git commit -am “mengubah index.html lagi dan lagi hehe”**

**clear**

**graph**

ubah index.html

**git status**

**git commit -am “mengubah index.html lagi dan lagi”**

kembali ke ariefcu

ubah index.html

**git commit -am “mengubah index.html lagi dan lagi”**

**push -u origin master**

balik lagi ke yang diundang collaborats

sebelum melakukan merging masternya berubah ga

**git checkout master**

**clear**

**graph**

kalau kita merging maka akan merge commit, merging dengan menambahkan 1 commit

tapi kita tidak akan melakukan itu

**git bracnh**

**git checkout fitur\_keren**

**git rebase master**

**graph**

**git checkout master**

**graph**

**git rebase fitur\_keren**

**graph**

**git push -u origin master**

**graph**

**Git : #14 Git & Web Hosting**

Melakukan sinkronisasi antara lokal, github dan web hosting

[**https://medium.com/@gofrendiasgard/git-perjalanan-menuju-dunia-parallel-c4e74364cf84**](https://medium.com/@gofrendiasgard/git-perjalanan-menuju-dunia-parallel-c4e74364cf84)

**Git Repository: Alam Semesta dalam Genggaman Kita**

Sebelum mengenal git, mungkin teman-teman sudah terlebih dahulu mengenal github, gitlab, atau bitbucket. Ketiganya adalah penyedia layanan untuk menyimpan repository git yang kita miliki. Pada kenyataannya, kita bisa membuat git-server sendiri, atau bahkan menggunakan git tanpa git-server.

Tiga baris berikut adalah perintah sederhana untuk menciptakan repository git folder git-workshop:

* mkdir git-workshop  
  Buat directory git-workshop
* cd git-workshop  
  Masuk ke directory tersebut
* git init  
  Jadikan directory tersebut sebuah git repository.

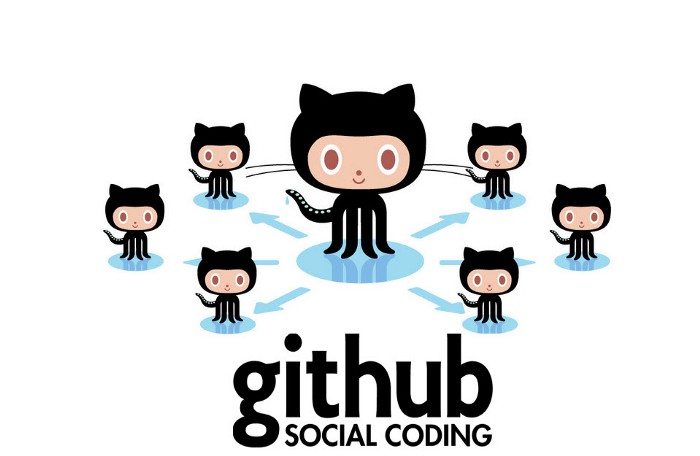
**Menambahkan File ke Git Repository**

Setelah kita memiliki git repository, kita bisa menambahkan sebuah file ke dalam git repository tersebut. Usai menambahkan file, kita perlu membawa file-file tersebut ke stage, sebelum akhirnya kita commit.

* echo "awal yang bagus" > catatan-sejarah.md  
  Di sini kita membuat sebuah file dengan nama catatan-sejarah.md dan mengisinya dengan sebaris kalimat awal yang bagus
* git add . -A  
  Lakukan staging pada semua file secara recursive (parameter -A) pada directory ini (tanda .)
* git commit -m "sebuah awal"  
  Commit semua file yang sudah di-stage dengan pesan sebuah awal.
* git status  
  Melihat status perubahan yang terjadi (ter-stage atau belum)
* git log  
  Melihat semua commit yang sudah terjadi sampai di titik ini

**Publikasi Git Repository ke Github**

Walaupun tidak wajib, namun ada baiknya kita simpan pekerjaan kita di github/bitbucket/gitlab. Selain memudahkan kolaborasi, ini juga dipakai untuk backup seandainya terjadi hal-hal yang tidak diinginkan dengan komputer/laptop kita.



Github: Social media yang minim bacotan netizen+62

Untuk melakukan itu, pertama-tama kalian harus punya account di github dan membuat repository di sana. Sebaiknya gunakan nama yang sama dengan repository lokal supaya tidak membingungkan.

Setelah itu, kalian dapat menjalankan perintah berikut:

* git remote add origin git@github.com:<user>/git-workshop.git  
  Dengan perintah ini, maka repository lokal kita akan mengenal repository github dengan alias origin.
* git push -u origin master  
  Sinkronisasi semua perubahan yang sudah kita lakukan di branch master ke github (dengan alias origin)
* git pull origin master  
  Sinkronisasi semua perubahan yang ada di github ke branch master di repository lokal.

Dengan mempublikasikan repository kita di github, maka kita membuka peluang kolaborasi dengan programmer lain di seluruh dunia.

Orang lain bisa bebas melakukan clone atau fork terhadap repository kita tersebut, melakukan perubahan seperlunya di branch mereka sendiri, dan memberikan pull-request.

Jika kalian masih belum ingin mempublikasikan repository kalian, maka kalian bisa mengatur setting repository menjadi private.

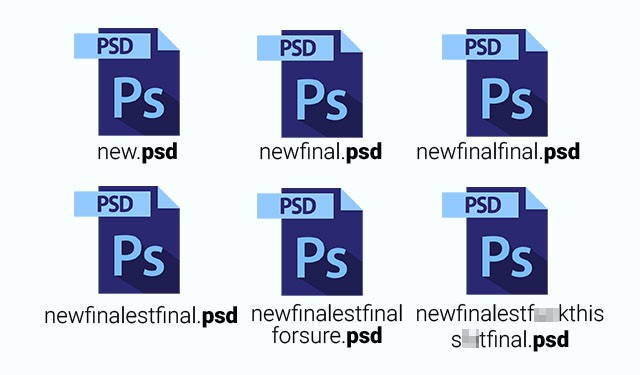
**Mengarungi Sang Waktu dengan Git: Unlimited Undo**

Kalian bisa melakukan beberapa commit yang lain, Lalu mencoba perintah berikut:

* git log  
  Melihat semua commit yang sudah terjadi sampai di titik ini.
* git rev-list --all  
  Melihat semua commit yang sudah terjadi, baik sebelum maupun setelah titik ini.
* git reflog  
  Melihat semua commit, termasuk checkout, yang sudah terjadi sampai di titik ini, sekaligus memberikan “referensi” seperti HEAD~0, HEAD~1, dan seterusnya.
* git checkout HEAD~1 atau git checkout <commit-hash>  
  Berpindah “satu langkah” ke belakang atau berpindah ke commit-hash tertentu
* git checkout master  
  Kembali ke commit terakhir di branch master.

Kalian lihat apa yang terjadi? Semua commit yang kalian lakukan tersimpan dengan baik, dan kalian bisa kembali ke titik manapun sesuka kalian !!!

Dengan adanya “unlimited undo” ini kalian tidak perlu lagi membuat file-file dengan nama konyol semacam ini:



Pekerjaan normies, tidak pakai git.

**Branch: Saat Kau Tak Harus Memilih**

Ada saatnya di mana kita harus berkolaborasi dengan orang lain untuk membuat fitur-fitur berbeda. Dalam kasus seperti ini akan lebih mudah seandainya kita bisa membuat “branch”.

Jadi ada versi source-code yang dikerjakan Si Anton, dan ada versi yang dikerjakan oleh Si Budi. Selama meng-coding, Anton dan Budi bisa bekerja secara terpisah. Lalu selanjutnya pekerjaan mereka bisa disatukan ke branch master.

Ini sangat menarik. Seolah-olah seperti kita menciptakan time-line baru di sebuah dunia parallel !!!

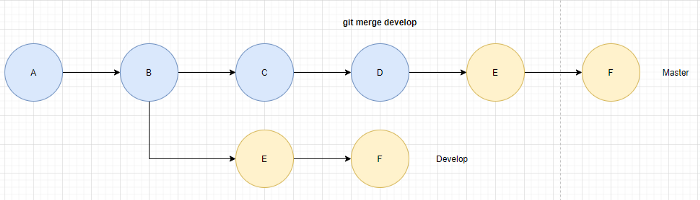


Yang bercabang dan menyatu: Rel kereta api, branch git, dan hubungan kita.

Untuk membuat branch baru kita bisa menggunakan perintah git checkout -b <nama-branch>

Berikut adalah perintah-perintah yang biasa dipakai untuk memanipulasi branch:

* git checkout -b <nama-branch>  
  Membuat branch baru dan berpindah ke branch tersebut
* git checkout <nama-branch>  
  Berpindah ke branch lain
* git merge <nama-branch>  
  Menggabungkan seluruh perubahan yang terjadi di branch nama-branch dengan perubahan yang kita lakukan di branch sekarang.



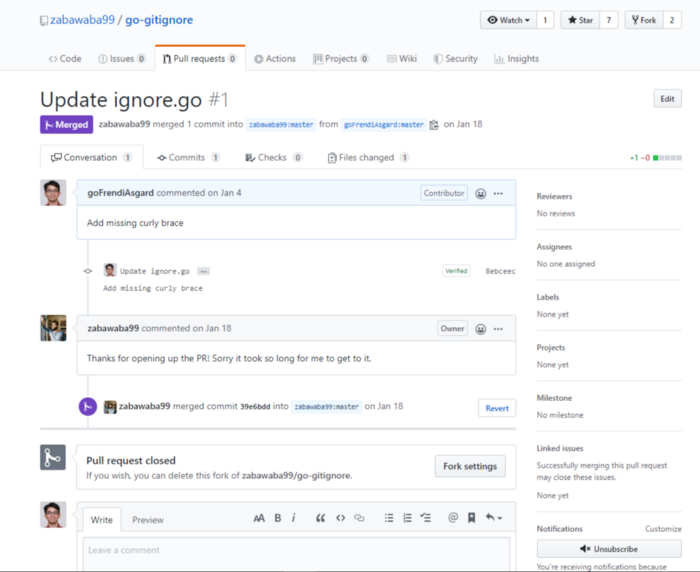
git merge

***PERINGATAN:****Saat bekerja bersama programmer lain, jangan pernah merge dan push langsung ke master. Pastikan semua perubahan yang kalian lakukan sudah melalui pull-request dan proses review.*

**Membuat Pull Request**

Setelah melakukan perubahan di branch kita, maka kita bisa meminta sang pemilik repository untuk menyatukan perubahan yang sudah kita buat ke branch master.

Github, bitbucket, dan gitlab sama-sama memiliki fitur pull-request, dan ini bisa dijadikan ajang untuk saling berkenalan serta mempererat tali silaturahmi sesama programmer.



Contoh Pull request (walaupun cuma nambahin kurung kurawal tutup)

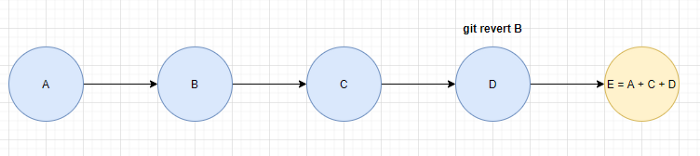
Kecuali kalian bekerja sendiri, maka ada baiknya kalian selalu melakukan pull request untuk setiap perubahan yang kalian lakukan. Di satu sisi ini membuat pekerjaan kalian “tercatat”. Di sisi lain, ini memungkinkan terjadinya code-review oleh rekan-rekan kalian.

**Mengubah Masa Lalu: Git revert**

Tak ada yang abadi. Fitur yang saat ini kita anggap bagus, mungkin beberapa bulan ke depan dianggap sampah. Sesuatu yang terlanjur berada di branch master pun, terkadang harus disesali.

Tak ada yang abadi. Bahkan band sekelas “Peterpan” pun tak abadi

Jika hanya ada sedikit commit yang bermasalah, maka cara terbaik untuk “memperbaiki masa lalu adalah dengan melakukan “revert”:



Git revert

git revert <commit-bermasalah>

Perintah revert memungkinkan kita membuat satu commit baru yang berisi semua commit sebelumnya kecuali commit yang di-revert. Untuk lebih jelasnya, coba lakukan serangkaian perintah berikut:

cd workshop-git-revert  
git init

echo 'alice' > alpha.html  
git add . -A && git commit -m "1st git commit: 1 file"

echo 'becky' > beta.html  
git add . -A && git commit -m "2nd git commit: 2 files"

echo 'callie' > charlie.html  
git add . -A && git commit -m "3rd git commit: 3 files"

echo 'diana' > delta.html  
git add . -A && git commit -m "4th git commit: 4 files"

echo 'ellen' > edison.html  
git add . -A && git commit -m "5th git commit: 5 files"

Sampai di sini, HEAD mengarah pada commit ke lima. Kita bisa me-refer commit-commit sebelumnya dengan “commit-hash” yang bisa didapatkan dengan perintah git log atau git rev-list --all. Namun supaya lebih mudah, kita bisa me-refer commit-commit sebelumnya dengan HEAD~1, HEAD~2 dan seterusnya.

ls  
git reflog

Semisal kita ingin membatalkan penambahan file beta.html (3 commit di belakang HEAD), maka kita bisa menjalankan perintah berikut:

git revert HEAD~3  
ls  
git reflog

Sukses !!! File beta.html sudah menghilang dari peredaran, seolah-olah tidak pernah ada.

Tapi tunggu dulu, pada kenyataannya, git revert bukan meniadakan sejarah, namun menambahkan satu commit baru yang berisi pembatalan satu commit.

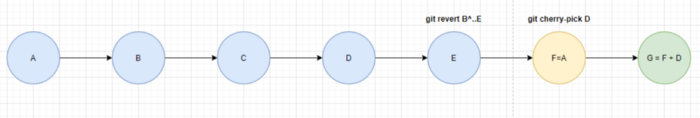
**Memilih commit terbaik: Git cherry-pick**

Kita sudah belajar bagaimana “membatalkan suatu commit”. Sekarang bagaimana jika kita hanya ingin membatalkan beberapa commit dan hanya ingin mengambil sebagian kecil commit saja?



cherry-pick

Kita bisa memanfaatkan fasilitas cherry-pick:



Git cherry-pick

cd workshop-git-cherry-pick  
git init

echo 'alice' > alpha.html  
git add . -A && git commit -m "1st git commit: 1 file"

echo 'becky' > beta.html  
git add . -A && git commit -m "2nd git commit: 2 files"

echo 'callie' > charlie.html  
git add . -A && git commit -m "3rd git commit: 3 files"

echo 'diana' > delta.html  
git add . -A && git commit -m "4th git commit: 4 files"

echo 'ellen' > edison.html  
git add . -A && git commit -m "5th git commit: 5 files"

ls  
git reflog

Sekarang katakan, kita ingin meniadakan perubahan kedua, ke tiga, dan ke lima (harapannya, nanti hanya ada file alpha.html dan delta.html saja). Dalam hal ini kita bisa melakukan perintah berikut:

git revert 84e72d2^..733776 (dengan asumsi 84e72d2 adalah hash commit perubahan beta.html dan 733776 adalah hash commit perubahan edison.html)  
git cherry-pick f0d522e (dengan asumsi f0d522e adalah hash commit perubahan delta.html)

Oke, sekarang repository kita hanya berisi alpha.html dan delta.html.

ls  
git reflog

**With Great Power Comes Great Responsibility**

Dalam penggunaan sehari-hari, umumnya kita hanya menggunakan perintah-perintah umum seperti checkout, add, commit, dan push.

Perintah revert dan cherry-pick pada dasarnya tidak menghilangkan commit-commit sebelumnya, melainkan membuat sebuah commit baru. Segala perubahan yang ada masih tetap tersimpan sebagai “sejarah” dan dapat diambil sewaktu-waktu.

Di balik segala keunggulannya, git menuntut penggunanya untuk bertanggung jawab atas setiap tindakan yang mereka lakukan. Menghapus “sejarah” sebenarnya dimungkinkan, namun memiliki resiko yang sangat tinggi, terutama jika semua perubahan kita telah ter-sinkronisasi ke github/bitbucket/gitlab. Oleh karena itu, para developer selalu melakukan hal berikut:

* Tidak meng-upload data-data sensitif (user/password, API key, dsb)
* Tidak meng-upload file-file berukuran besar dan rentan berubah (node\_modules, grafik atau apapun yang di-generate ulang oleh program, file database sqlite)
* Cuci tangan dengan sabun, tidur secukupnya, makan makanan bergizi. (Stay sane and healthy guys, kesehatanmu yang paling utama… he he he…)

# Cara Ubah Remote URL Repositori Git

* Kategori:
* [git](https://www.kaklabs.com/category/git)

https://www.kaklabs.com/2016/01/14/cara-ubah-remote-url-repositori-git.html

Jan 14, 2016 - Terakhir diubah Jan 27, 2022

Tulisan dibawah ini adalah penjelasan bagaimana mengganti remote repositori git, bukan menambakan remote repositori baru. Untuk menambahkan remote repositori dapat menggunakan perintah berikut

git remote add namarepo https://github.com/USERNAME/OTHERREPOSITORY.git

Terkait mengganti remote repositori, ada beberapa cara mengubah remote URL pada repositori git, yang pertama dengan perintah git remote set-url. Yang kedua adalah mengganti url pada file .git/config. Berikut adalah langkah-lang mengubah remote URL repositori git

## 1. Buka terminal dan pindah ke git repositori

cd your-git-repository

## 2. Lihat Remote URL yang Tersedia

git remote -v

# origin git@github.com:USERNAME/REPOSITORY.git (fetch)

# origin git@github.com:USERNAME/REPOSITORY.git (push)

## 3. Ganti Repositori git Dengan Perintah git remote set-url

git remote set-url origin https://github.com/USERNAME/OTHERREPOSITORY.git

Opsi lain bisa ubah URL dengan mengubah berkas .git/config pada suatu repositori.

git remote set-url origin https://github.com/USERNAME/OTHERREPOSITORY.git

[remote "origin"]

url = git@bitbucket.org:USERNAME/OTHERREPOSITORY.git

## 4. Verifikasi Remote URL Telah Berubah dan Selesai

git remote -v

# origin https://github.com/USERNAME/OTHERREPOSITORY.git (fetch)

# origin https://github.com/USERNAME/OTHERREPOSITORY.git (push)