Git Tutorial

Mavlarn
2014/02/19

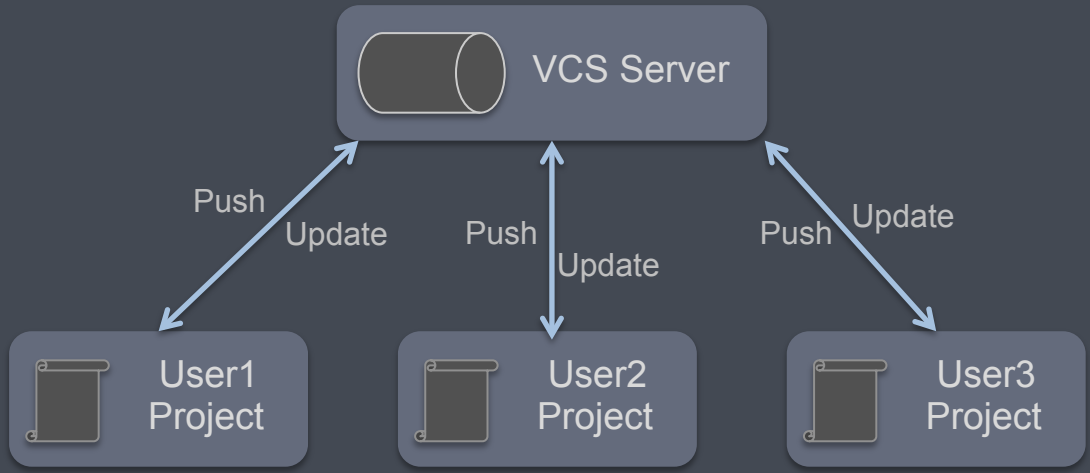
Contents

- VCS and DVCS
- Git basic
- Git workflow
- Working with remote repositories

Introduction

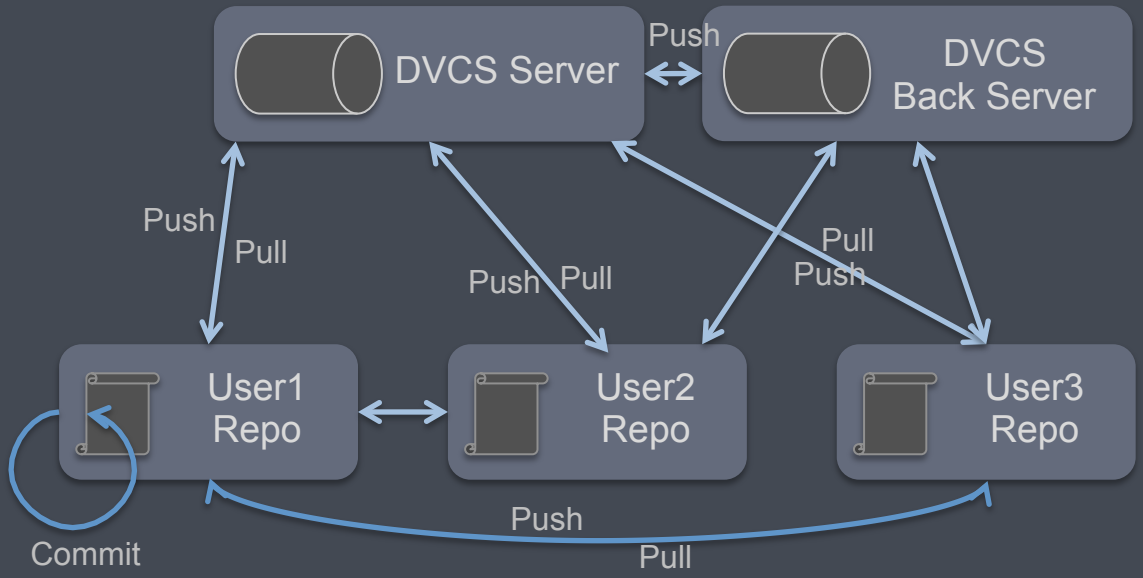
Version Control System

Subversion
CVS
.....



Distributed Version Control System

Git
Mercurial
.....



Distributed Version Control System

Git is:

Free and open source dvcs system
Super fast and easy to use
Able to use in local offline

Online Git Service:

https://github.com
https://bitbucket.org
https://code.csdn.net
http://git.oschina.net/

Open Source Git Management Application:

gitlab

Git Basic

$ git init

$ git clone

Create git repository is super easy

```
# Create a git repository in current directory.
$ git init your-project-name

# Clone from a remote git repository
$ git clone remote-repo-url
```

Git supports many protocols

```
http protocol:
        https://github.com/Mavlarn/git-tutorial.git
Git protocol:
        git@github.com:Mavlarn/git-tutorial.git
ssh protocol:
        ssh://[user@]host.xz[:port]/path/to/repo.git/
        [user@]host.xz:path/to/repo.git
```

$ git add

$ git rm

$ Git mv

Add files into Index to track version:

```
# Add changed file to index(staging area).
$ git add modified-file
# add all files
$ git add .
```

Remove or move files

```
# Remove files from index, also delete in working directory
$ git rm file-name
# remove the file from index and keep the change in working directory
$ git rm --cached file-name

# move file
$ git mv from-file-name  to-file-name
```

$ git status

Display the status change between working directory and current HEAD commit

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   test2.tmp
#
# 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:   git-tutorial.md
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       pic/
#       test.txt
```

$ git status

Git status includes change status between:

- the index file and the current HEAD commit
- the working tree and the index file
- the untracked files

$ git diff

Show changes between the working directory and the index.

```
$ git diff
diff --git a/git-tutorial.md b/git-tutorial.md
index 017fb35..252a464 100644
--- a/git-tutorial.md
+++ b/git-tutorial.md
@@ -78,3 +78,4 @@ Some changed content.
```

Show changes between 2 commits.

```
$ git diff 0e19dc5 16384ac
diff --git a/git-tutorial.md b/git-tutorial.md
index 017fb35..252a464 100644
--- a/git-tutorial.md
+++ b/git-tutorial.md
@@ -78,3 +78,4 @@ Some changed content.
Other changed before.
……
```

Show changes between cache(index) and HEAD(last commit).

```
$ git diff --cache
```

$ git commit

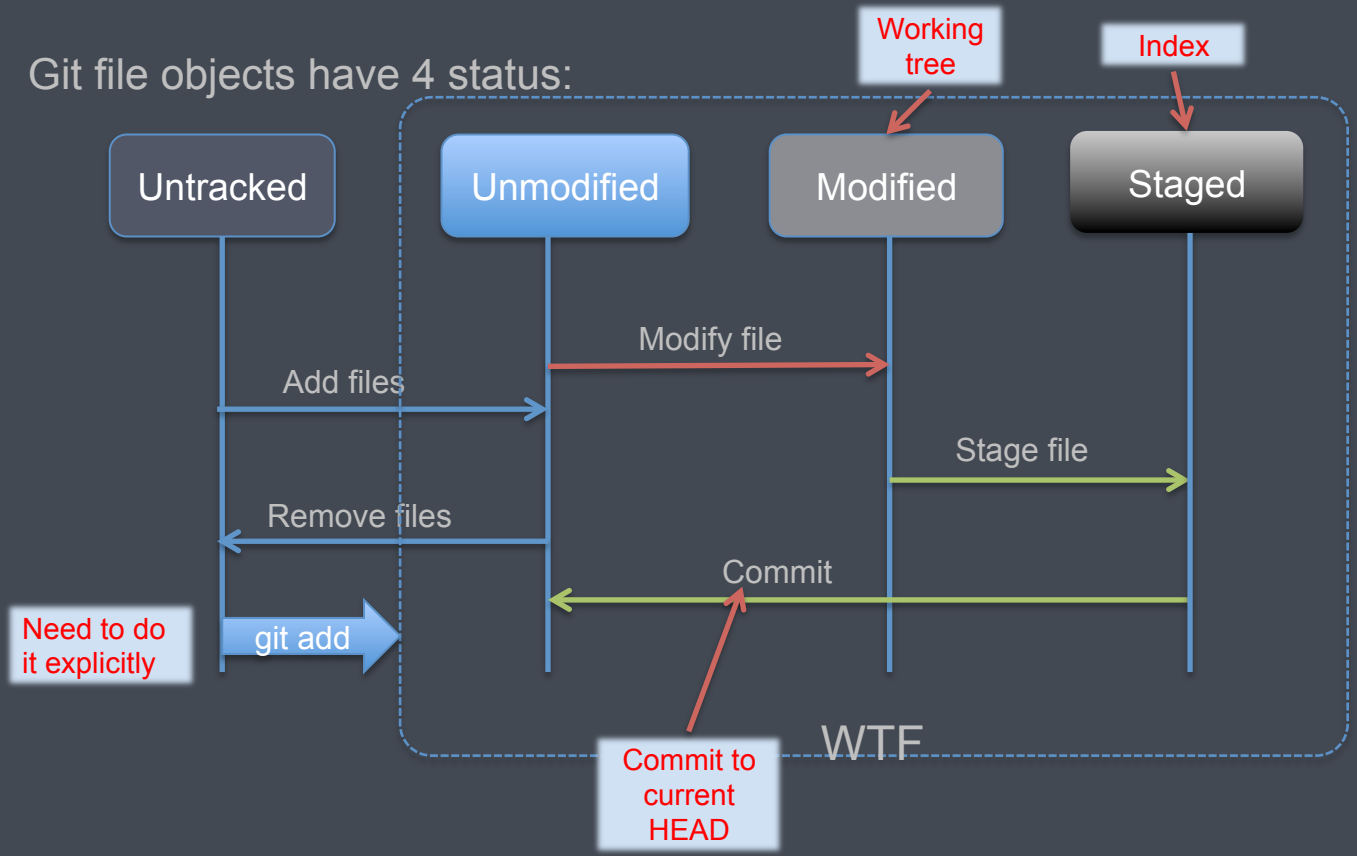
Commit the change in working directory into local repository.

```
$ git commit -m “commit message”
[master 288ea24] commit message
 1 file changed, 1 insertion(+)
Mavlarn-Mac:git-tutorial mavlarn$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       pic/
#       test.txt
nothing added to commit but untracked files present (use "git add" to
track)
```

The files not in stage will not be committed. Need to add first using ‘git add …’

Git File Status

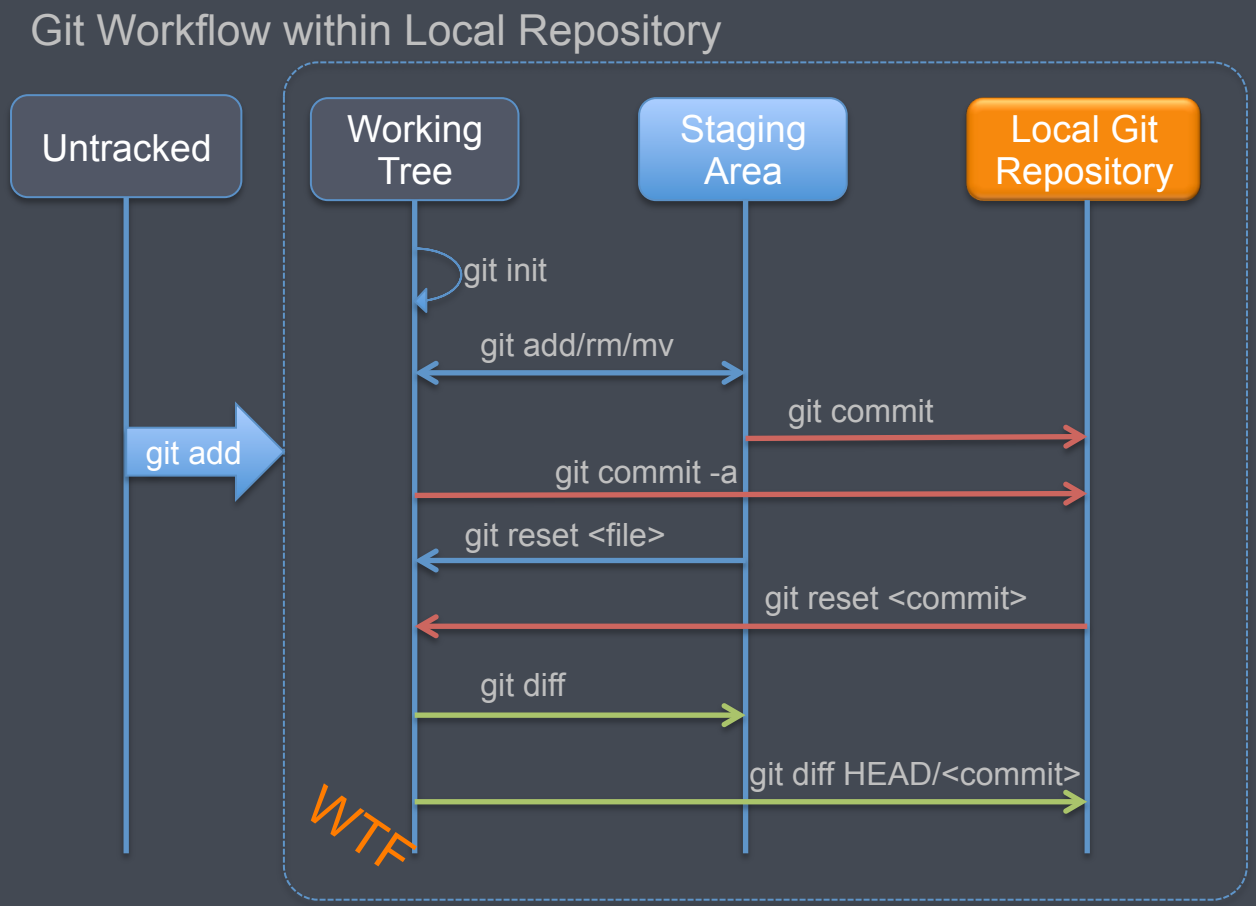
Git file objects have 4 status:



For a modified file in working tree, you need to add it into stage and then commit. Or do it in one step with -a

```
$ git commit -am “Add and commit in one step”
```

Git Local workflow

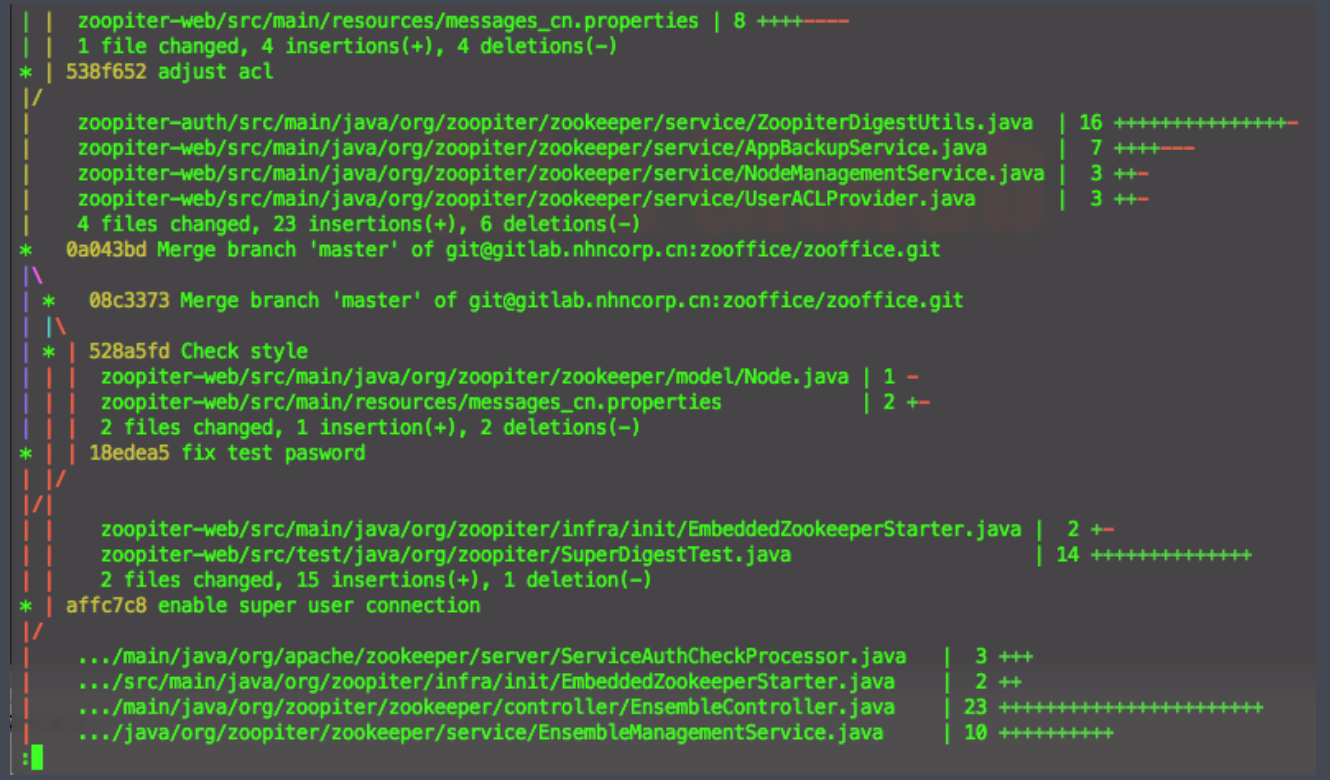


Git History

$ git log

Get git log

```
$ git log --graph --decorate --oneline --stat -n 5
```



$ git log -S

You can search something from log.

```
# -p used to print the actual content, but not difference.
# use -S search the changed content, it is not commit message.
$ git log -p -S”change_content”
$ git log -p -S"change_content" --since="1 week ago”

# to search commit, use grep
$ git log -p --grep="Ticket #382"
```

$ git blame

Sometime you want to check who and when modified one file.

```
$ git blame test.txt

1d5d9b07 (Mavlarn 2014-02-18 11:45:32 +0800 1) <<<<<<< HEAD
5617e3df (Mavlarn 2014-02-18 11:43:53 +0800 2) master
1d5d9b07 (Mavlarn 2014-02-18 11:45:32 +0800 3) =======
965e8bb3 (Mavlarn 2014-02-18 11:45:00 +0800 4) devel
1d5d9b07 (Mavlarn 2014-02-18 11:45:32 +0800 5) >>>>>>> devel
```

$ git log changelog

Git log can also be used to generate change log.
Use .. Between 2 commits, branches or tags.

```
$ git log --no-merges --format="%an: %s" f3174cf..1b78002
$ git log --no-merges --format="%an: %s" version-tag1.. version-tag2

JunHo Yoon: [NGRINDER-719] Trim the user name and id before saving into
db.
junoyoon: [NGRINDER-703] Fix the messages keys of configurations
junoyoon: [NGRINDER-703] Fix configuration key typo error
junoyoon: [NGRINDER-703] Fix test error
junoyoon: [NGRINDER-690] Simplify impl
junoyoon: [NGRINDER-690] Provide log apis
junoyoon: [NGRINDER-690] Provide log apis
junoyoon: [NGRINDER-703] Fix configuration key to be more consistent
junoyoon: [NGRINDER-718] Fix message to be explicit
```

Commit message is important. Make it meaningful and standard.

$ git help

Git provides a very convenient and detailed manual.

```
$ git help merge
```

```
NAME
       git-merge - Join two or more development histories together

SYNOPSIS
       git merge [-n] [--stat] [--no-commit] [--squash] [--[no-]edit]
               [-s <strategy>] [-X <strategy-option>]
               [--[no-]rerere-autoupdate] [-m <msg>] [<commit>...]
       git merge <msg> HEAD <commit>...
       git merge --abort

DESCRIPTION
       Incorporates changes from the named commits (since the time their histories diverged from the current
       branch) into the current branch. This command is used by git pull to incorporate changes from another
       repository and can be used by hand to merge changes from one branch into another.

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

                     A---B---C topic
                    /
               D---E---F---G master

       Then "git merge topic" will replay the changes made on the topic branch since it diverged from master
       (i.e., E) until its current commit (C) on top of master, and record the result in a new commit along
       with the names of the two parent commits and a log message from the user describing the changes.

                     A---B---C topic
                    /         \
               D---E---F---G---H master
```

Git Undo

$ git commit --amend

There are 4 kinds of “Undo” operation for git:
Commit --amend reset revert checkout

git commit --amend can be used to fix the last commit.

```
$ git log --graph --decorate --oneline –n 2
* a65766b (HEAD, master) Init update.
* ff02a9e update

$ git add forgotten_file

$ git commit --amend --m “amend message” # Commit the change to last commit
$ git log --graph --decorate --oneline -n 2
* 02625f4 (HEAD, master)  amend message  # Amended to latest commit
* ff02a9e update  # Previous commit

$ git commit --amend -m "amend update message” # commit without any change
$ git log --graph --decorate --oneline –n 2
* 20fe216 (HEAD, master) amend update message # Just commit message changed
* ff02a9e update # Previous commit
```

$ git reset <file>

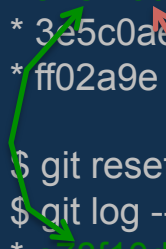
We can reset files from stage.

```
$ echo “Updated content” >> new-file.txt
$ git add new-file.txt
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   new-file.txt

$ git reset new-file.txt
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       new-file.txt
```

$ git reset <commit>

We can also reset the commit.

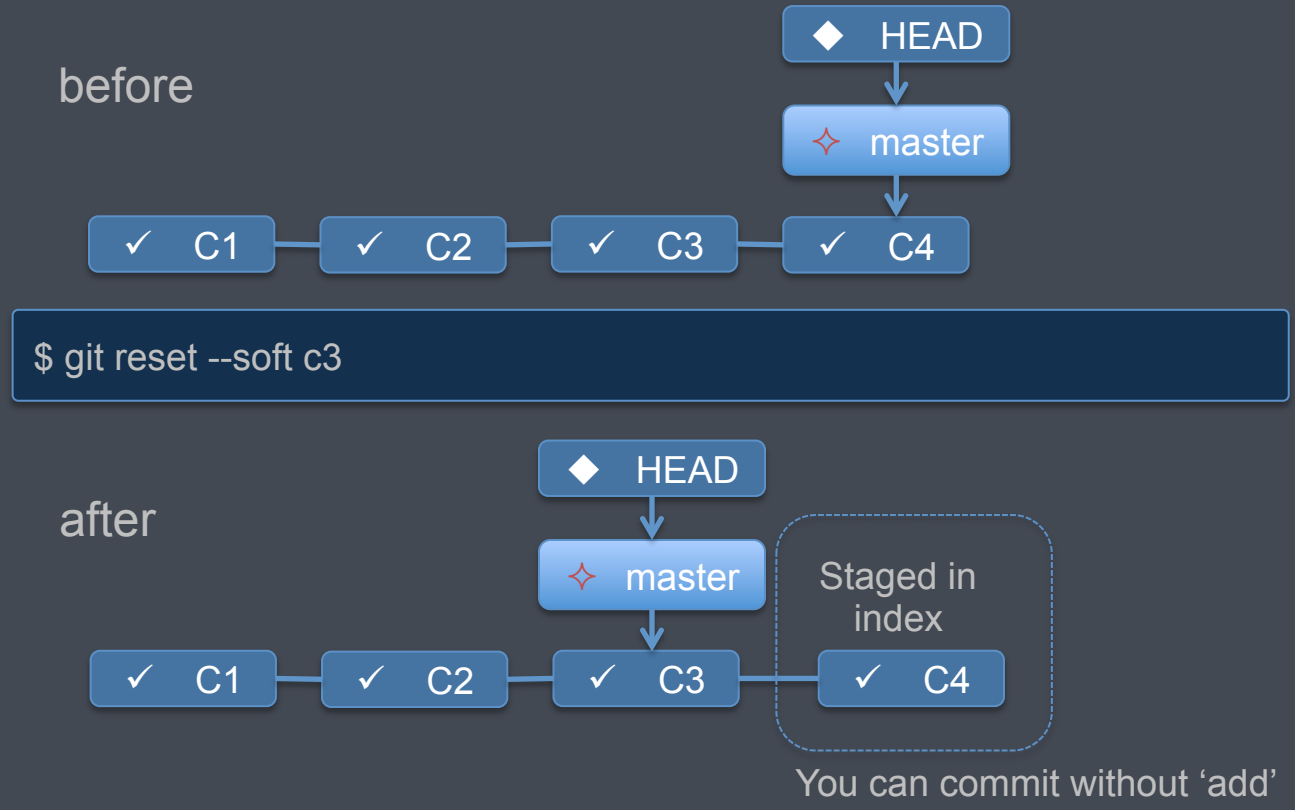
```
$ git log --graph --decorate --oneline –n 4
* 864d47d (HEAD, master) new change
* a78f10d test
* 3e5c0ae amend update message
* ff02a9e update

$ git reset a78f10d # reset to previous commit, commit 864d47d is removed
$ git log --graph --decorate --oneline –n 4
* a78f10d (HEAD, master) test  # Head is previous commit now.
* 3e5c0ae amend update message
* ff02a9e update
* a5f3384 update
```



You should **NOT** reset a commit after you have push it to remote repository.

$ git reset <commit>

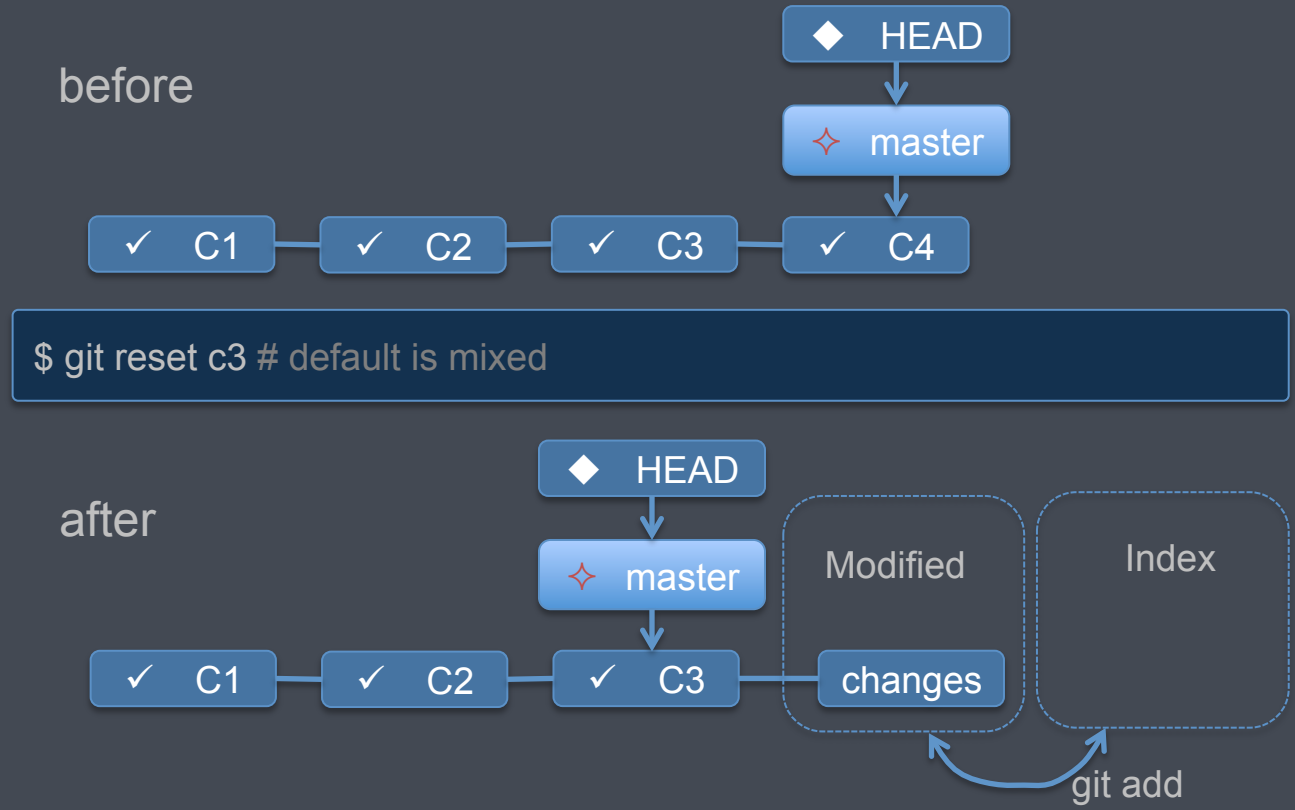
Reset mode:
--soft: Change of reset commit will be kept and still staged.



$ git reset <commit>

Reset mode:

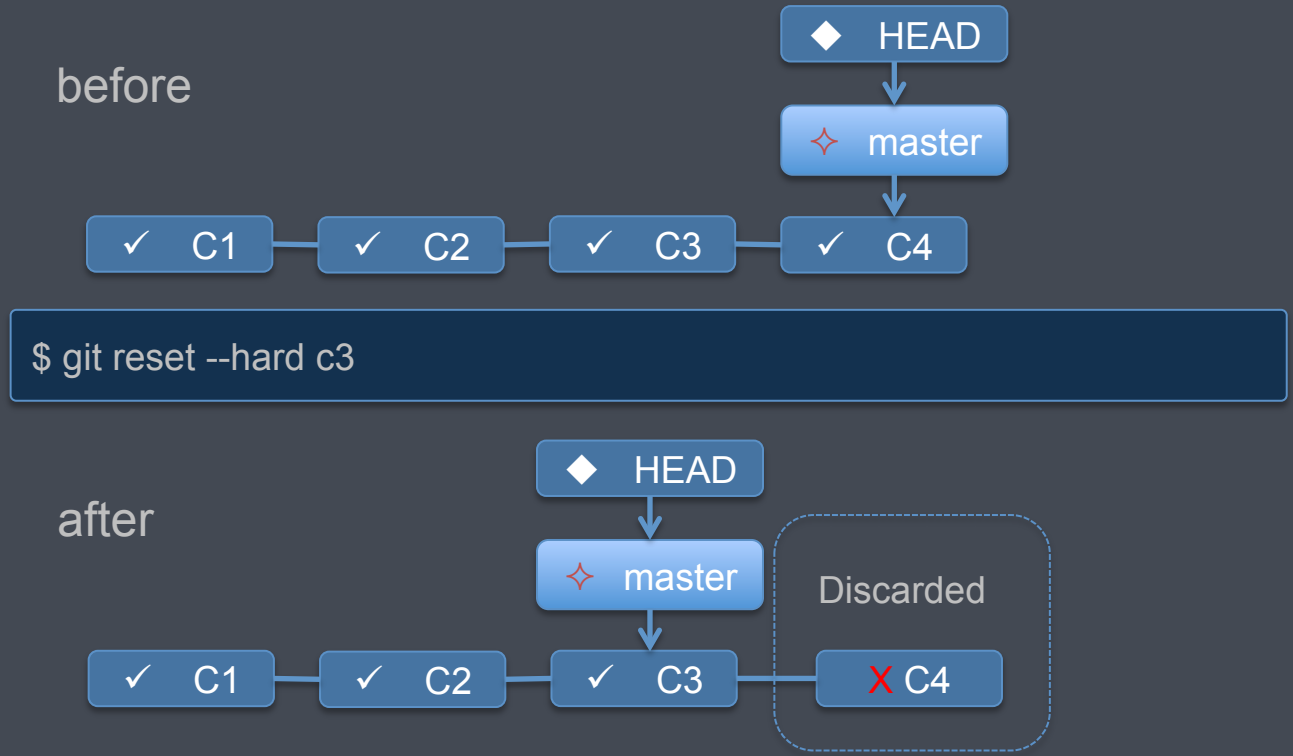
--mixed: Change of reset commit will be kept and become un-staged.



$ git reset <commit>

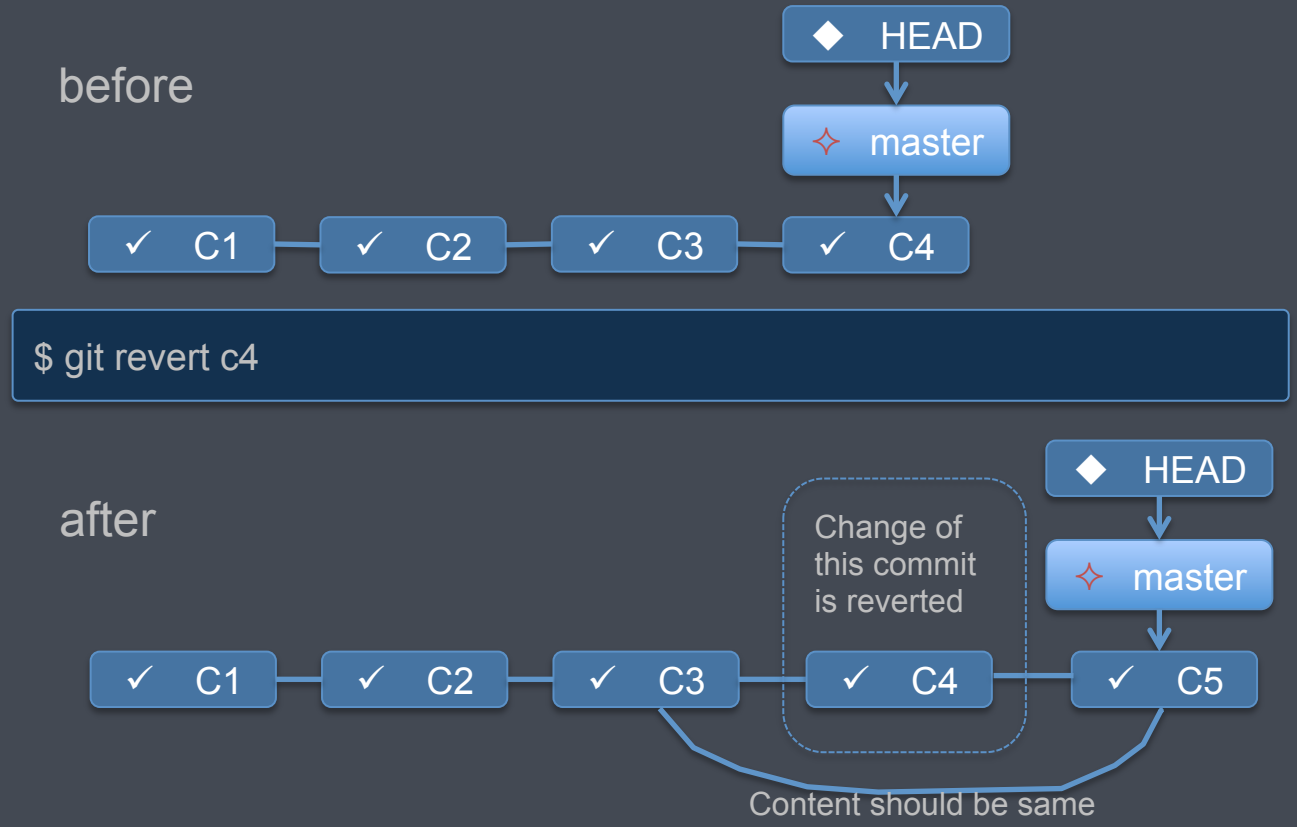
Reset mode:

--hard: The change of reset commit will be discarded.



$ git revert

Git revert is used to add new commit to reverse the effect of some earlier commits.



$ git checkout <file>/ <commit>

To ignore some change of one file, we can checkout it from commit.

```
# Discard some change to the file
$ echo “Update content” >> test1.txt
$ git status # check the change need to be committed
$ git checkout -- test1.txt # discard the change
```

Or we can checkout all files of one commit.

```
# checkout one commit
$ git log --graph --decorate --oneline -n 4
* 878b4d7 (HEAD, master) 555
* 3e5c0ae amend update message
* ff02a9e update
* a5f3384 update
$ git checkout HEAD^^ # checkout the 2 step previous commit.
$ git status
# HEAD detached at ff02a9e
```

Git Branch

$ git branch

To work with git branch:

```
$ git branch new-branch-name  # create new branch
$ git branch -d branch-name # delete branch
$ git branch -a # list all branch
```

Create and switch to new branch:

```
$ git checkout -b new-branch-name

# or do it like this:
$ git branch new-branch-name
$ git checkout new-branch-name
```

$ git tag

To work with git branch:

```
# create new tag, based on HEAD commit
$ git tag new-tag-name [-m message]
$ git tag -d tag-name # delete tag
$ git tag -l # list all tags

# create a tag based on one commit
$ git tag new-tag-name cd7dd94
```

Don’t forget to push the tags into remote repository.
Without ‘--tags’ options, tags info will be be pushed.

```
$ git push origin --tags
```

$ git merge <branch>

To merge branch:

```
# create new branch
$ git merge branch-name # merge this branch to current working branch.
```

But you will not be always lucky to merge successfully. There will be conflicts sometimes.

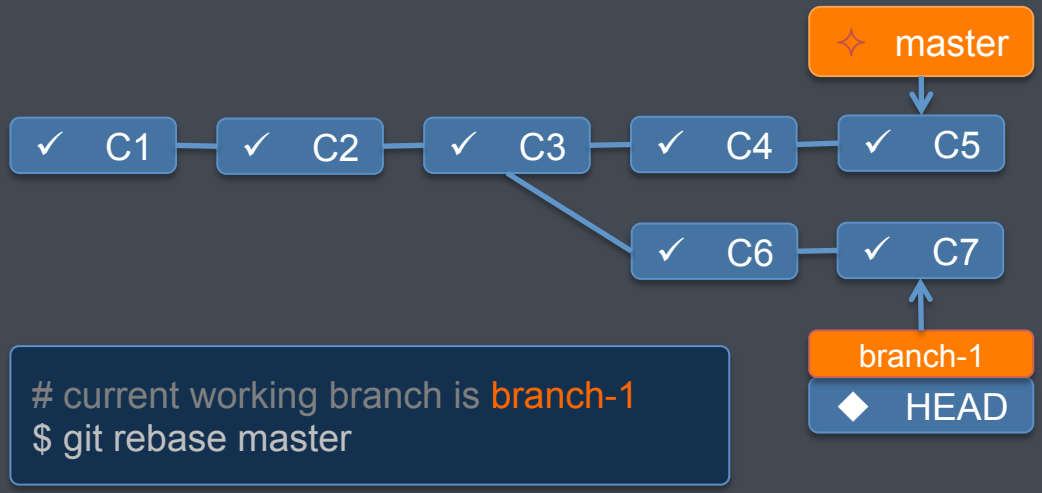
```
<<<<<<< HEAD:index.html                    Content of current branch
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer"> please contact us at support@github.com </div>
>>>>>>> iss53:index.html                  Content of branch to merge
```

After you fixed the conflicts, use ‘git add’ to mark it as ‘resolved’:

```
$ git add conflict-file
$ git commit -m “merged”
```

$ git rebase <branch>

Rebase will change the HEAD of current branch.

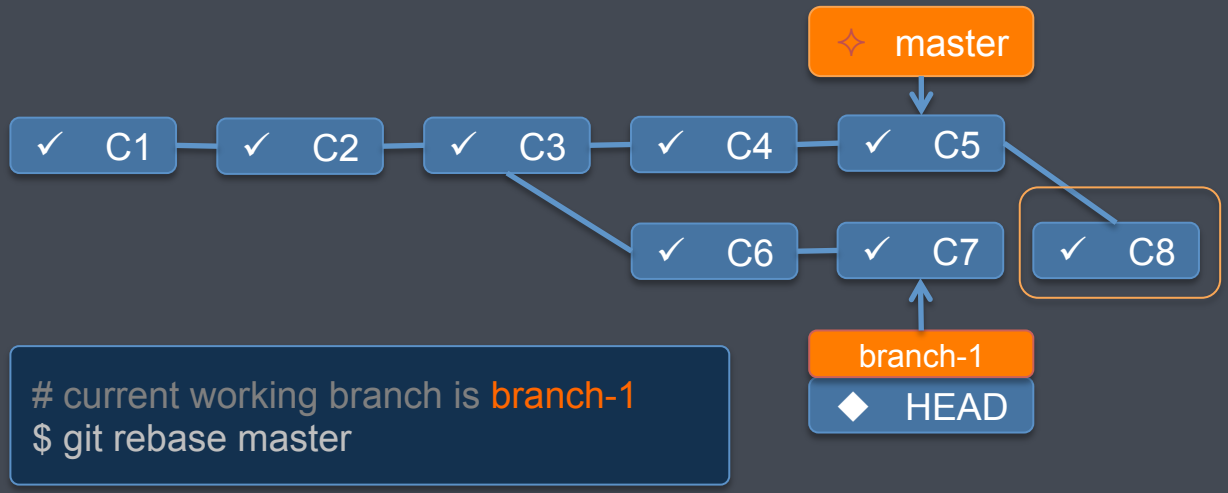



```
# current working branch is branch-1
$ git rebase master
```

- HEAD of branch-1 will be re-based on C5 (current master HEAD)

$ git rebase <branch>

Rebase will change the HEAD of current branch.

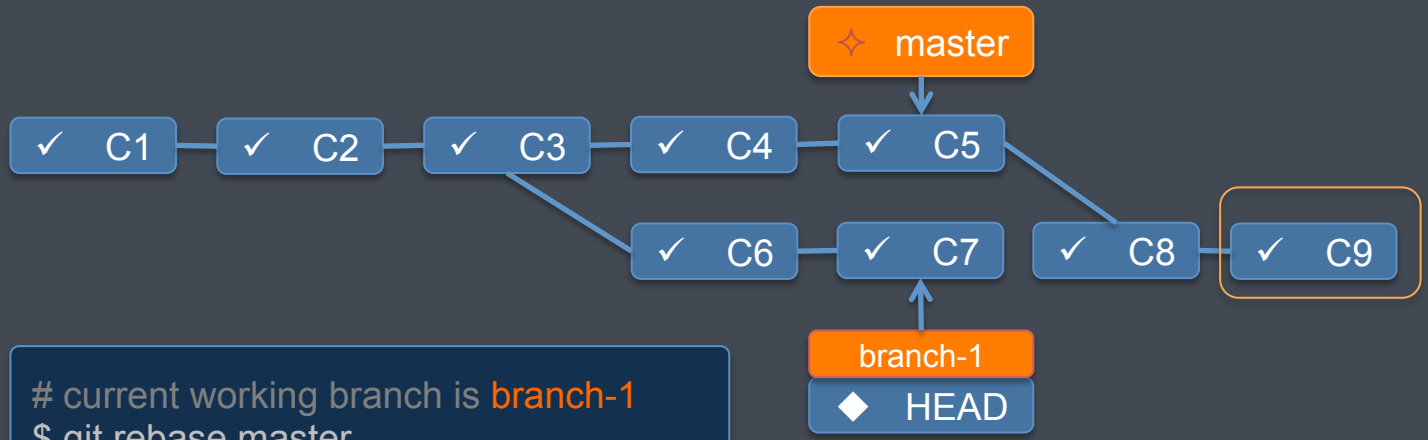



```
# current working branch is branch-1
$ git rebase master
```

- HEAD of branch-1 will be re-based on C5 (current master HEAD)
- Patch the change of C6 to C5, commit as C8

$ git rebase <branch>

Rebase will change the HEAD of current branch.

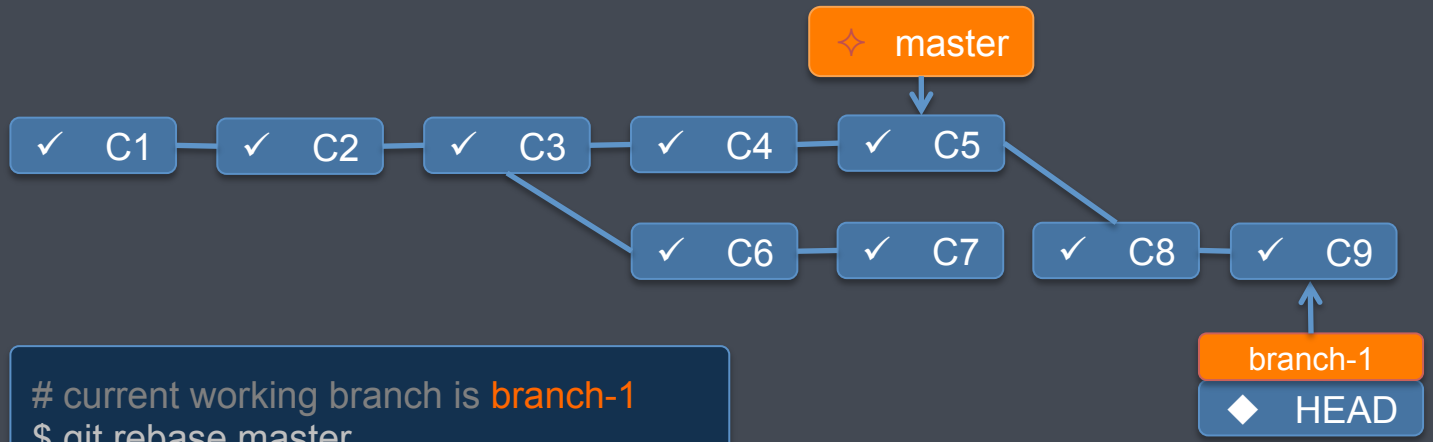



```
# current working branch is branch-1
$ git rebase master
```

- HEAD of branch-1 will be re-based on C5 (current master HEAD)
- Patch the change of C6 to C5, commit as C8
- Patch the change of C7 to C8, commit as C9

$ git rebase <branch>

Rebase will change the HEAD of current branch.

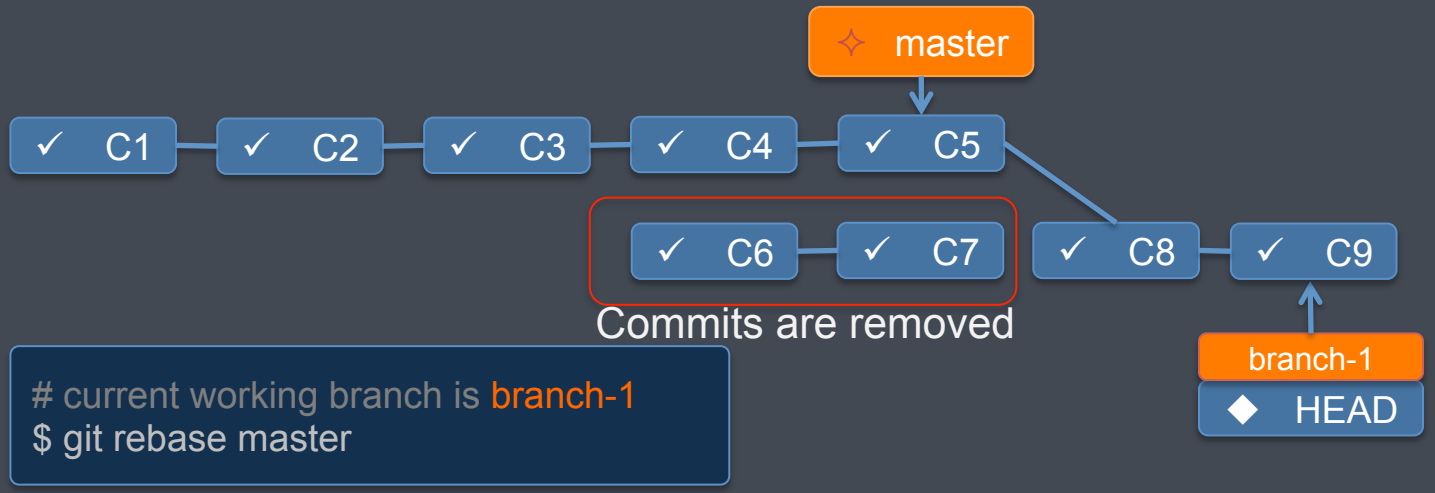



```
# current working branch is branch-1
$ git rebase master
```

- HEAD of branch-1 will be re-based on C5 (current master HEAD)
- Patch the change of C6 to C5, commit as C8
- Patch the change of C7 to C8, commit as C9
- Change current branch HEAD to C9

$ git rebase <branch>

Rebase will change the HEAD of current branch.




```
# current working branch is branch-1
$ git rebase master
```

- HEAD of branch-1 will be re-based on C5 (current master HEAD)
- Patch the change of C6 to C5, commit as C8
- Patch the change of C7 to C8, commit as C9
- Change current branch HEAD to C9
- C6 and C7 commit will be deleted

$ git rebase <branch>

After ‘rebase’, some commits will be removed.

If some others are working on that branch, they would kill you.

Never using ‘rebase’ on a shared branch.

Use ‘merge’ instead.

Git Configuration

$ git config

Configure global or repository level setting.

```
$ git config --global user.name “Mavlarn”
$ git config --global user.mail mavlarn@mail.com
$ git config --global color.ui true
$ git config –global core.excludesfile ~/.gitignore_global

# Repository configuration
$ git config core.autocrlf true

git config --get core.excludesfile
git config --unset core.gitproxy
```

Git config files priority:

```
$GIT_DIR/config
        Repository configuration file.
~/.gitconfig
        User git config, also called "global" configuration file.
/etc/gitconfig
         System-wide configuration file.
```

$ git ignore

My global ignore file:

```
*.iml
.idea/
.DS_Store
.settings
.project
.class
.classpath
target/
src/main/webapp/WEB-INF/classes
```

Git ignore files:

core.excludesfile configuration in gitconfig

$GIT_DIR/.gitignore

Git Remote

$ git remote

Time to work in **distributed**

```
# add a remote repo named as 'origin'
$ git remote add origin https://github.com/Mavlarn/git-tutorial.git

# change the url of remote repository
$ git remote set-url origin https://github.com/Mavlarn/git-tutorial-other.git

$ git remote show origin
$ git remote remove origin

$ git remote -v  # list the remote repositories list
```

Git remote config is saved in $GIT_DIR/config.
You need to set ssh key for remote git. Or add user/password in url.

```
$ git remote set-url origin https://mavlarn:mypassword@github.com/
Mavlarn/git-tutorial-other.git
```

$ git push

Push the local commits to remote repository

```
$ git push -u origin master
# add '-u' to set up-stream for every branches. Later you can fetch from 'origin master' without set them.

# push tags into remote repository
$ git push origin --tags
```

If remote repository is on the same branch with yours, you can not push into it. You need to add 'force':

```
# Assume you are working in 'master' branch.
# Working branch of some-repo is 'master' too
$ git push --force some-repo master
```

Normally, it's better to switch the central repository's current branch to a not-used branch, to avoid this.

$ git pull

Pull commits from remote repository to local

```
# Pull change commits from remote ‘origin’ and ‘master’ branch.
$ git pull origin master

# if you add ‘-u’ option during push, or set up-stream with:
$ git branch --set-upstream-to=origin/<master> master
# not need to set ‘where’ to pull commits
$ git pull
```

For ‘push’, from remote repository’s aspect, this ‘push’ operation is like:

- Get updated commits from ‘user’ repository
- Merge these commits into remote’s current repository

For ‘pull’, from user’s repository’s aspect, this ‘pull’ operation is like:

- Get updated commits from ‘remote’ repository
- Merge these commits into ‘current’ repository

$ git fetch

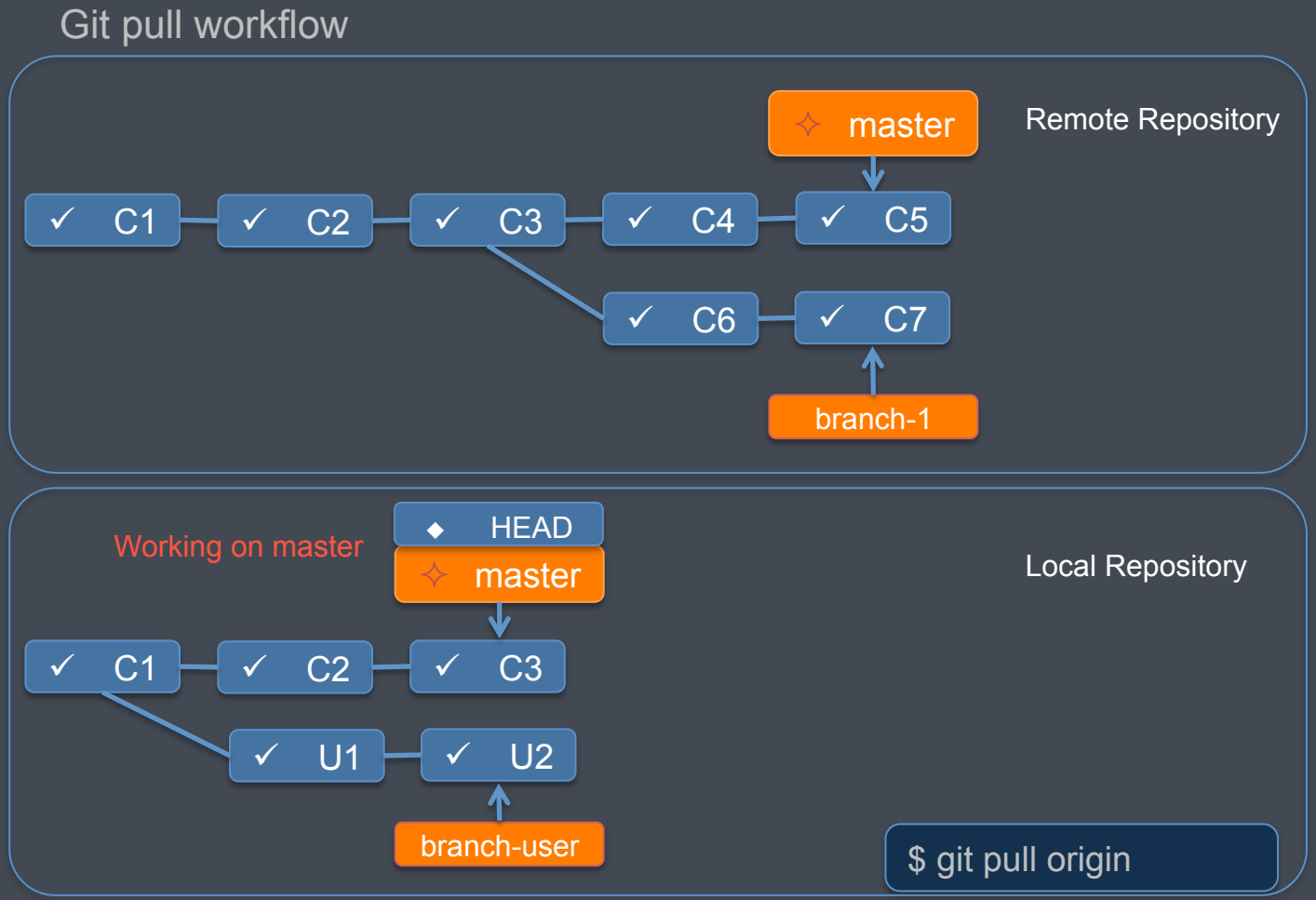
You can just get the commits from remote, but not merge.

```
# fetch from one remote
$ git fetch origin

# fetch all remote
$ git fetch --all

$ git fetch --tags # tags info will be fetched by default
```

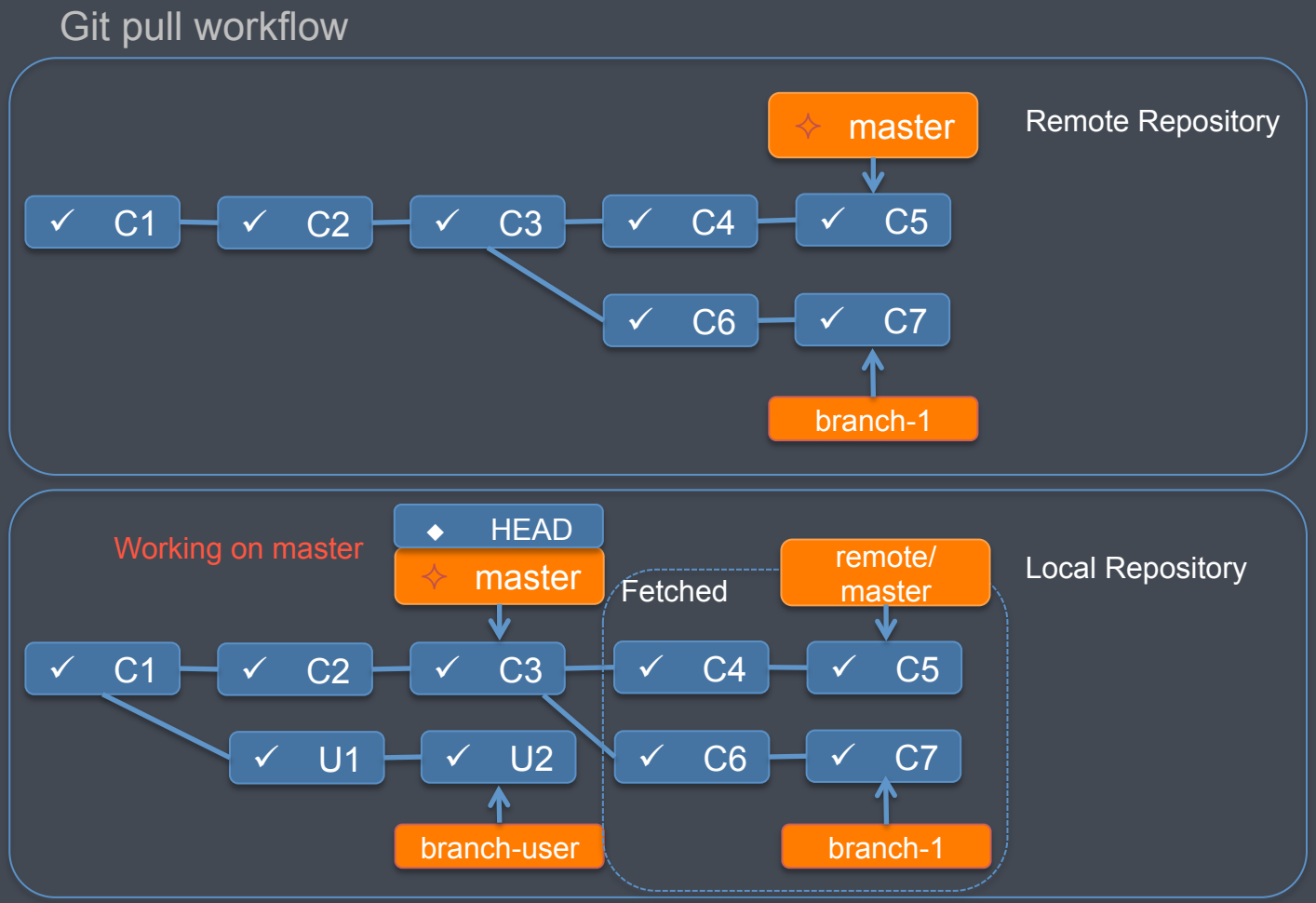
'pull' is same as fetch + merge, the fetched commit from remote is saved as '`FETCH_HEAD`'

```
$ git fetch origin

# fetched HEAD pointer is saved as FETCH_HEAD
$ git merge FETCH_HEAD
```

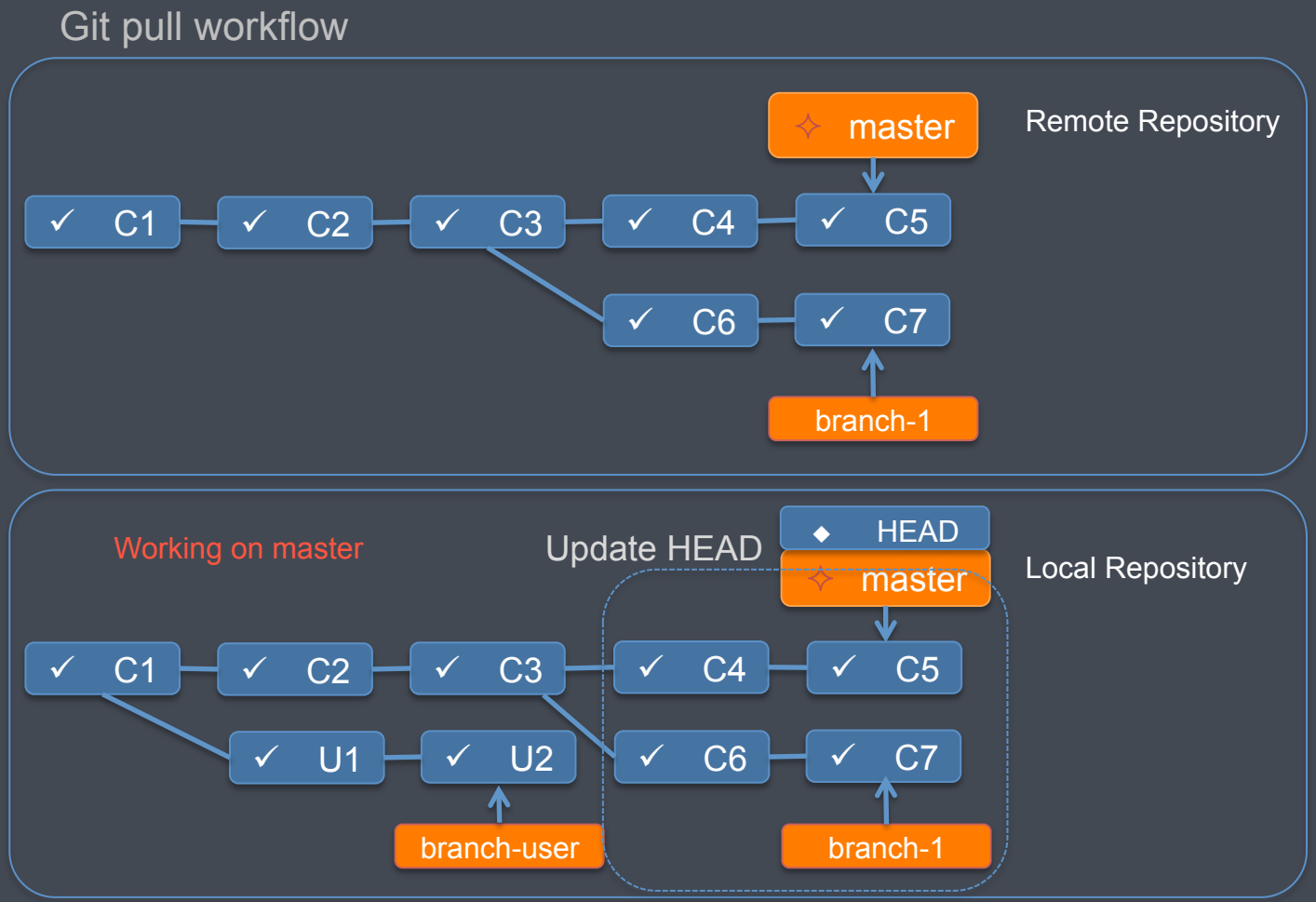
Git Pull Workflow



Git Pull Workflow



Git Pull Workflow



Share Local Repos

If user-a wants to share his repository to user-b:

- With ssh

user-a need to enable ssh server on his machine

```
# On user-b's side
$ git remote add user-a-repo [user@]user-a_host_address:path/to/repos
```

- Enable git service:

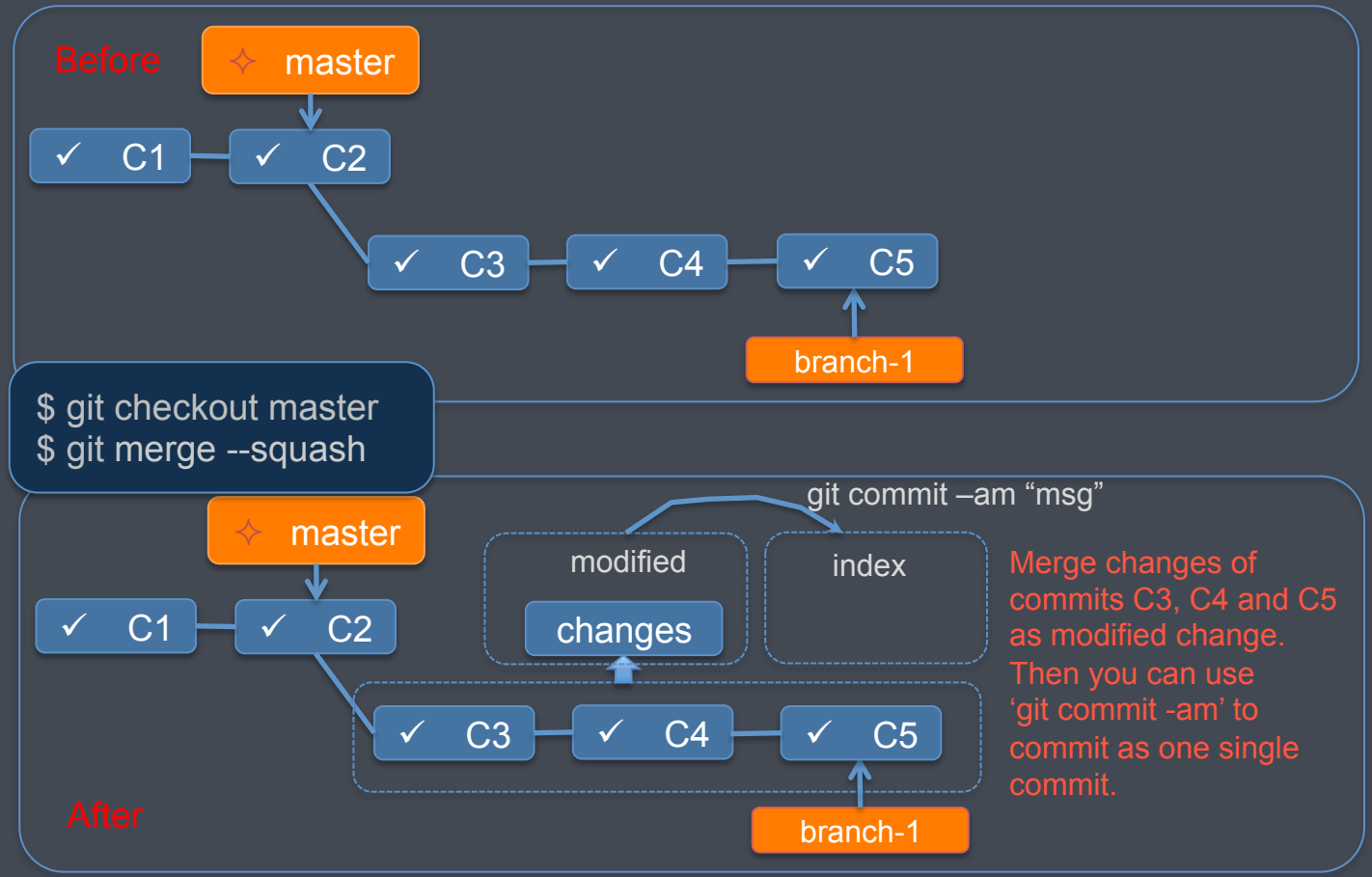
git integrated a daemon service.

```
# start the git service on user-a side
$ git daemon --reuseaddr --verbose --export-all --base-path=/path/to/root/
of/repos

# user-b adds remote repos as git protocol url
$ git remote add user-a-repo git://user-a_host_address:repo_name
```

Commit as Single Commit

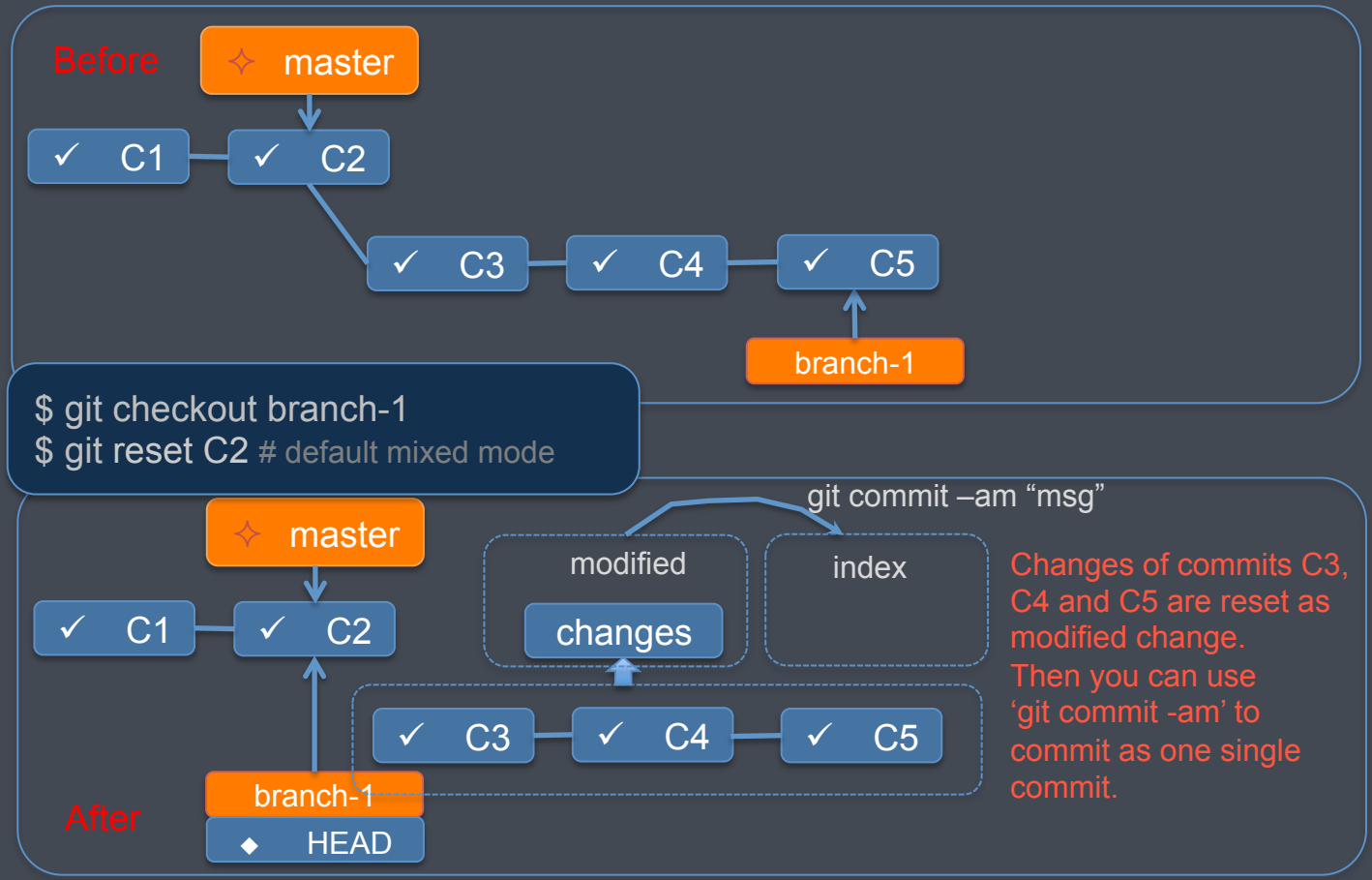
git merge --squash

Use merge --squash to commit as one single commit



git reset <commit>

You can also use reset to do that.



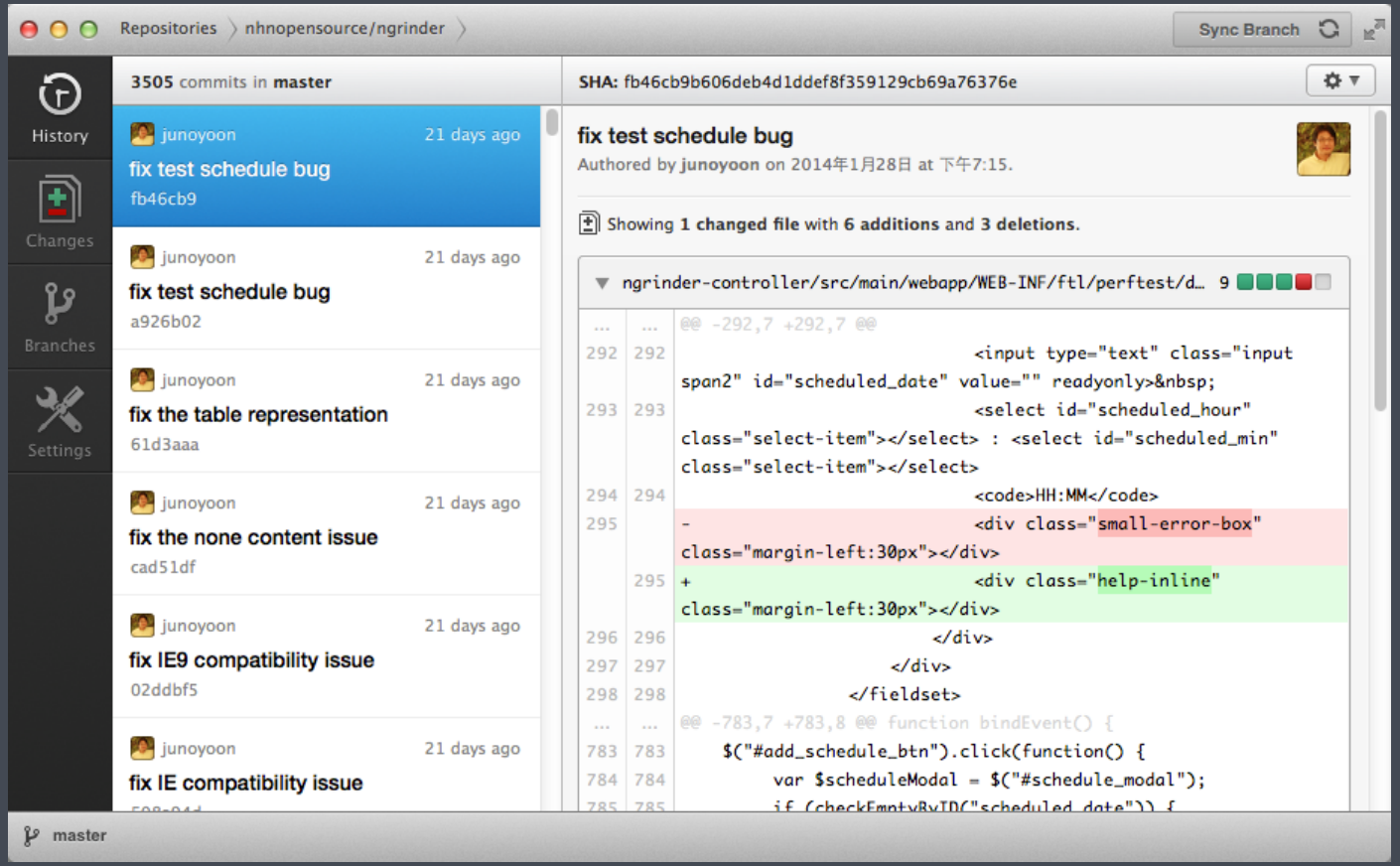
Git remote practice

- Always pull before push.
- If you are working on share branch, push/pull frequently to avoid too much conflicts.
- You should keep all commits normally.
- But sometime, you need to keep commits clean, e.g. the open source project. Then we need 2 remote repositories, one for release and one fore trunks. Make commit on release repos clean. Use ‘rebase’ or ‘merge --squash’.

Git Tool

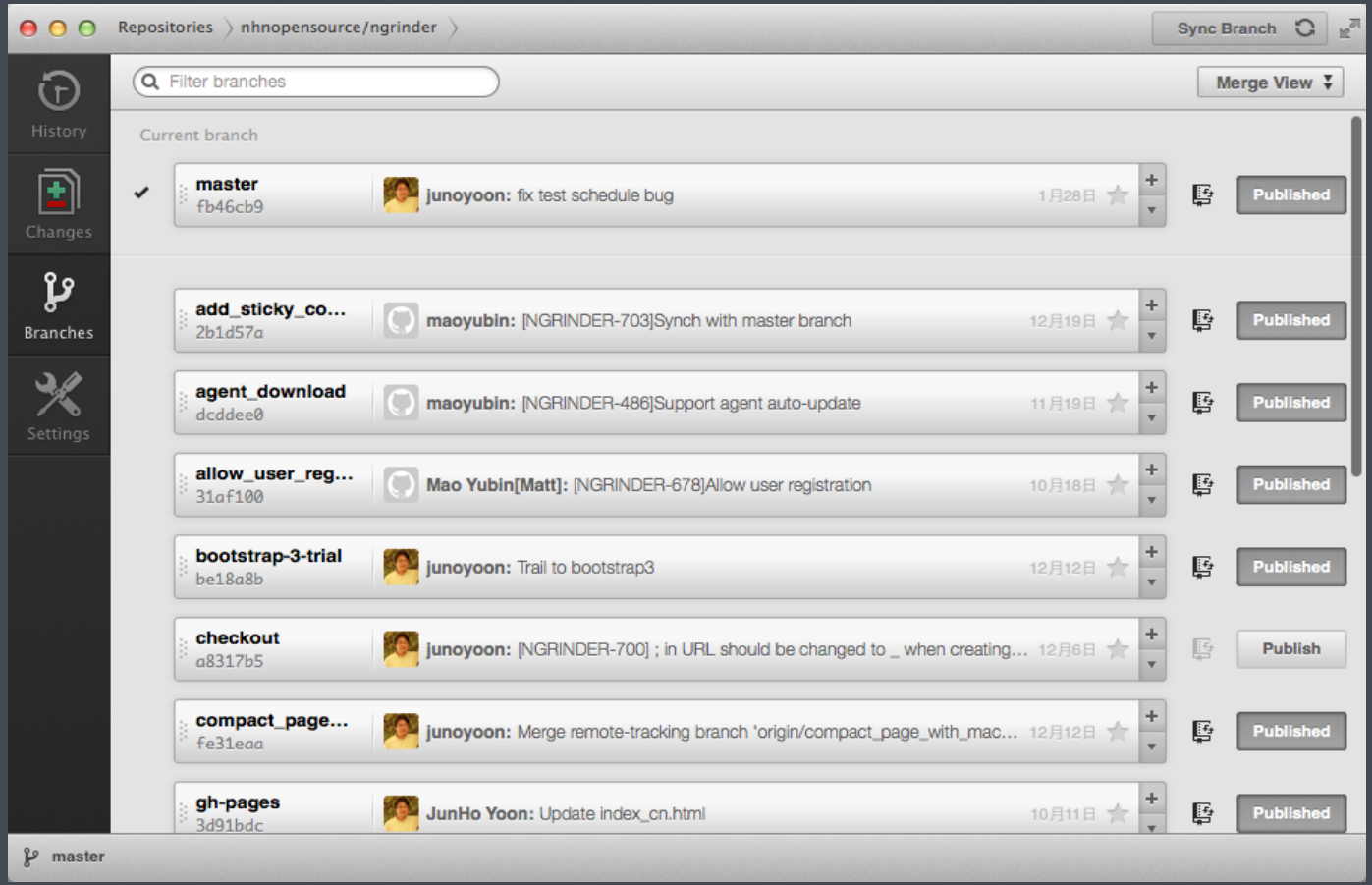
GitHub GUI Tool

GitHub - official github client tool. Both Win/Mac platforms.



GitHub GUI Tool

GitHub - official github client tool. Both Win/Mac platforms.



Git Resources

- Official git site

http://git-scm.com

- Step-by-step tutorial

http://try.github.com

http://gitimmersion.com/lab_01.html

- Online Book

http://git-scm.com/book

http://www-cs-students.stanford.edu/~blynn/gitmagic

- Cheat sheet

http://www.git-tower.com/blog/git-cheat-sheet-detail