

# Git Notes

I will teach you some knowledge of life.

Muffle a lot of money →

by Shuxiao



# Do you use Git

Of course!

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.



# Topics

## 1. Download and organize git repo

in Go src style, use glone

## 2. Rebase

2 cases in daily development

## 3. Merge or rebase?

Differences and scenarios

## 4. How to

some cases we meet in daily development

## 5. Some diff tool in terminal and configuration I use

# 1. Download and organize git repo

in Go src style, use glone

1. Current status
2. Tool



# 1.1 How code organized in Go src

This is tree-like structure in \$GOROOT/src

```
├── github.com
│   ├── alessio
│   │   └── shellescape
│   ├── BurntSushi
│   │   └── toml
│   ├── evanphx
│   │   └── json-patch
│   ├── jessevdk
│   │   └── go-flags
│   ├── mattn
│   │   └── go-isatty
│   ├── mitchellh
│   │   └── go-homedir
│   ├── pelletier
│   │   └── go-toml
│   ├── pkg
│   │   └── errors
│   └── spf13
│       ├── cobra
│       └── pflag
```

# 1.3 Tool can help

## Glone

<https://github.com/keaising/glone>

### SUPPORT

```
git://github.com/zsh-users/zsh-completions.git
https://github.com/zsh-users/zsh-completions
```

### USAGE

```
taiga@clinkz ~/code/go/src
Δ glone git://github.com/zsh-users/zsh-completions.git
git clone git@github.com:zsh-users/zsh-completions.git /home/taiga/code/github.com/zsh-users/zsh-completions
Cloning into '/home/taiga/code/github.com/zsh-users/zsh-completions'...
remote: Enumerating objects: 4761, done.
remote: Counting objects: 100% (126/126), done.
remote: Compressing objects: 100% (71/71), done.
remote: Total 4761 (delta 63), reused 112 (delta 55), pack-reused 4635
Receiving objects: 100% (4761/4761), 1.78 MiB | 1.89 MiB/s, done.
Resolving deltas: 100% (3006/3006), done.
```

## 2. Rebase

1. Edit latest commits history
2. Integrate changes from one branch into another

## 2.1 Edit latest commits history

Current status

[github.com/keaising/auto-mouse-keyboard](https://github.com/keaising/auto-mouse-keyboard)

```
* 7907930 - (HEAD -> test, tag: v1.3.2, origin/main, main) change windows release (6 months ago) <keaising>
* 2e05e48 - (tag: v1.3.1) bugfix and run specific file (6 months ago) <keaising>
* 9aadc52 - update README (6 months ago) <keaising>
* 5aa8708 - (tag: v1.3.0) read all .conf in current working directory (6 months ago) <keaising>
* 92fb077 - (tag: v1.2.1) bugfix (6 months ago) <keaising>
* b2616ce - (tag: v1.2.0) Merge pull request #1 from keaising/workflow (6 months ago) <keaising>
|\
| * 9058389 - add macos package (6 months ago) <keaising>
| * a525111 - add macos (6 months ago) <keaising>
| * ce80f5f - add test CI (6 months ago) <keaising>
| * 82e9a96 - add test CI (6 months ago) <keaising>
| * 93a449c - add release (6 months ago) <keaising>
| * 6fe1371 - upload artifacts after building (6 months ago) <keaising>
| * 7d282f6 - add windows (6 months ago) <keaising>
| * 99d9958 - follow ubuntu setup (6 months ago) <keaising>
| * 20822c2 - fix ci (6 months ago) <keaising>
| * fccbe92 - Create go.yml (6 months ago) <Shuxiao WANG>
|/
* 8c3dad1 - (tag: v1.1.0) change S to ms and add support for loop (6 months ago) <keaising>
```



## 2.1 Edit latest commits history

Target status: the latest 5 commits will be squashed into 1 commit

[github.com/keaising/auto-mouse-keyboard](https://github.com/keaising/auto-mouse-keyboard)

```
* 13584ba - squash commit (75 seconds ago) <keaising>
* b2616ce - (tag: v1.2.0) Merge pull request #1 from keaising/workflow (6 months ago) <keaising>
|\
| * 9058389 - add macos package (6 months ago) <keaising>
| * a525111 - add macos (6 months ago) <keaising>
| * ce80f5f - add test CI (6 months ago) <keaising>
| * 82e9a96 - add test CI (6 months ago) <keaising>
| * 93a449c - add release (6 months ago) <keaising>
| * 6fe1371 - upload artifacts after building (6 months ago) <keaising>
| * 7d282f6 - add windows (6 months ago) <keaising>
| * 99d9958 - follow ubuntu setup (6 months ago) <keaising>
| * 20822c2 - fix ci (6 months ago) <keaising>
| * fccbe92 - Create go.yml (6 months ago) <Shuxiao WANG>
|/
* 8c3dad1 - (tag: v1.1.0) change S to ms and add support for loop (6 months ago) <keaising>
* b56721f - update config convert (6 months ago) <keaising>
* 6756c93 - update readme (6 months ago) <keaising>
```

## 2.1 Edit latest commits history

Full process of squash

1. Tell git: we want to start editing the latest 5 commits
2. Determine the commands of each commit's modifications
3. Edit each commit based on commands in step 2
  - a. edit as command declared(commit message or diff)
  - b. resolve all possible conflicts in the editing
  - c. use ``git add . && git commit && git rebase --continue`` to finish editing for one commit
4. Squash finish

## 2.1 Edit latest commits history

Step 1. Tell git: we want to start editing the latest 5 commits

Use command to re-edit the latest 5 commits

```
git rebase -i HEAD~5
```

## 2.1 Edit latest commits history

Step 2. Determine the commands of each commit's modifications

After running commands in step 1, git will open your default editor to allow you edit message below

```
pick 92fb077 bugfix
squash 5aa8708 read all .conf in current working directory
squash 9aadc52 update README
squash 2e05e48 bugfix and run specific file
squash 7907930 change windows release name amk-windows.exe=> amk.exe

# Rebase b2616ce..7907930 onto 2e05e48 (5 commands)
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
```

And we can see, there are more commands like `\pick / reword / edit / fixup / drop` can be used in

## 2.1 Edit latest commits history

Step 2. Determine the commands of each commit's modifications

You need to be careful about the sort of commits, it is reversed from git log

### 1. Rebase message

```
oldest ->    pick 92fb077 bugfix
            ->    squash 5aa8708 read all .conf in current working directory
            ->    squash 9aadc52 update README
            ->    squash 2e05e48 bugfix and run specific file
latest ->    squash 7907930 change windows release name amk-windows.exe=> amk.exe
```

### 2. Git log

```
latest ->    * 7907930 - (HEAD -> test, tag: v1.3.2, main) change windows release (6 months ago) <keaising>
            ->    * 2e05e48 - (tag: v1.3.1) bugfix and run specific file (6 months ago) <keaising>
            ->    * 9aadc52 - update README (6 months ago) <keaising>
            ->    * 5aa8708 - (tag: v1.3.0) read all .conf in current working directory (6 months ago) <keaising>
oldest ->    * 92fb077 - (tag: v1.2.1) bugfix (6 months ago) <keaising>
```

## 2.1 Edit latest commits history

Step 3. Edit each commit based on commands in step 2

After editing declaration message and closing the editor, git will open an new editor window for editing each commit message

In this case, we only keep one commit, so we only need to edit once

If there are more commits kept, you need to edit mulitple times

This is what I want to save in the squashed commit's message

```
This is the combination of 5 commits.
```



## 2.1 Edit latest commits history

### Step 4. Final result

```
* b7b2bc3 - (HEAD -> test) This is the combination of 5 commits. (75 seconds ago) <keaising>
* b2616ce - (tag: v1.2.0) Merge pull request #1 from keaising/workflow (6 months ago) <keaising>
|\
| * 9058389 - add macos package (6 months ago) <keaising>
| * a525111 - add macos (6 months ago) <keaising>
| * ce80f5f - add test CI (6 months ago) <keaising>
| * 82e9a96 - add test CI (6 months ago) <keaising>
| * 93a449c - add release (6 months ago) <keaising>
| * 6fe1371 - upload artifacts after building (6 months ago) <keaising>
| * 7d282f6 - add windows (6 months ago) <keaising>
| * 99d9958 - follow ubuntu setup (6 months ago) <keaising>
| * 20822c2 - fix ci (6 months ago) <keaising>
| * fccbe92 - Create go.yml (6 months ago) <Shuxiao WANG>
|/
* 8c3dad1 - (tag: v1.1.0) change S to ms and add support for loop (6 months ago) <keaising>
* b56721f - update config convert (6 months ago) <keaising>
* 6756c93 - update readme (6 months ago) <keaising>
```

## 2.2 Integrate changes, similar to Merge

Rebase branchA based on branchB

1. Rebase will repeat all commit from branchA to branchB
2. Rebase will update commit history of the branchA
3. Rebase will not create a `merge` commit like `b2616ce`

```
* b7b2bc3 - (HEAD -> test) This is the combination of 5 commits. (75 seconds ago) <keaising>
* b2616ce - (tag: v1.2.0) Merge pull request #1 from keaising/workflow (6 months ago) <keaising>
|\
| * 9058389 - add macos package (6 months ago) <keaising>
| * 93a449c - add release (6 months ago) <keaising>
| * 6fe1371 - upload artifacts after building (6 months ago) <keaising>
| * 99d9958 - follow ubuntu setup (6 months ago) <keaising>
| * fccbe92 - Create go.yml (6 months ago) <Shuxiao WANG>
|/
* 8c3dad1 - (tag: v1.1.0) change S to ms and add support for loop (6 months ago) <keaising>
```

4. After rebase, branchA can merge into branchB in `fast forward` mode
5. If conflict exists in rebase process, something different

## 2.2 Integrate changes, similar to Merge

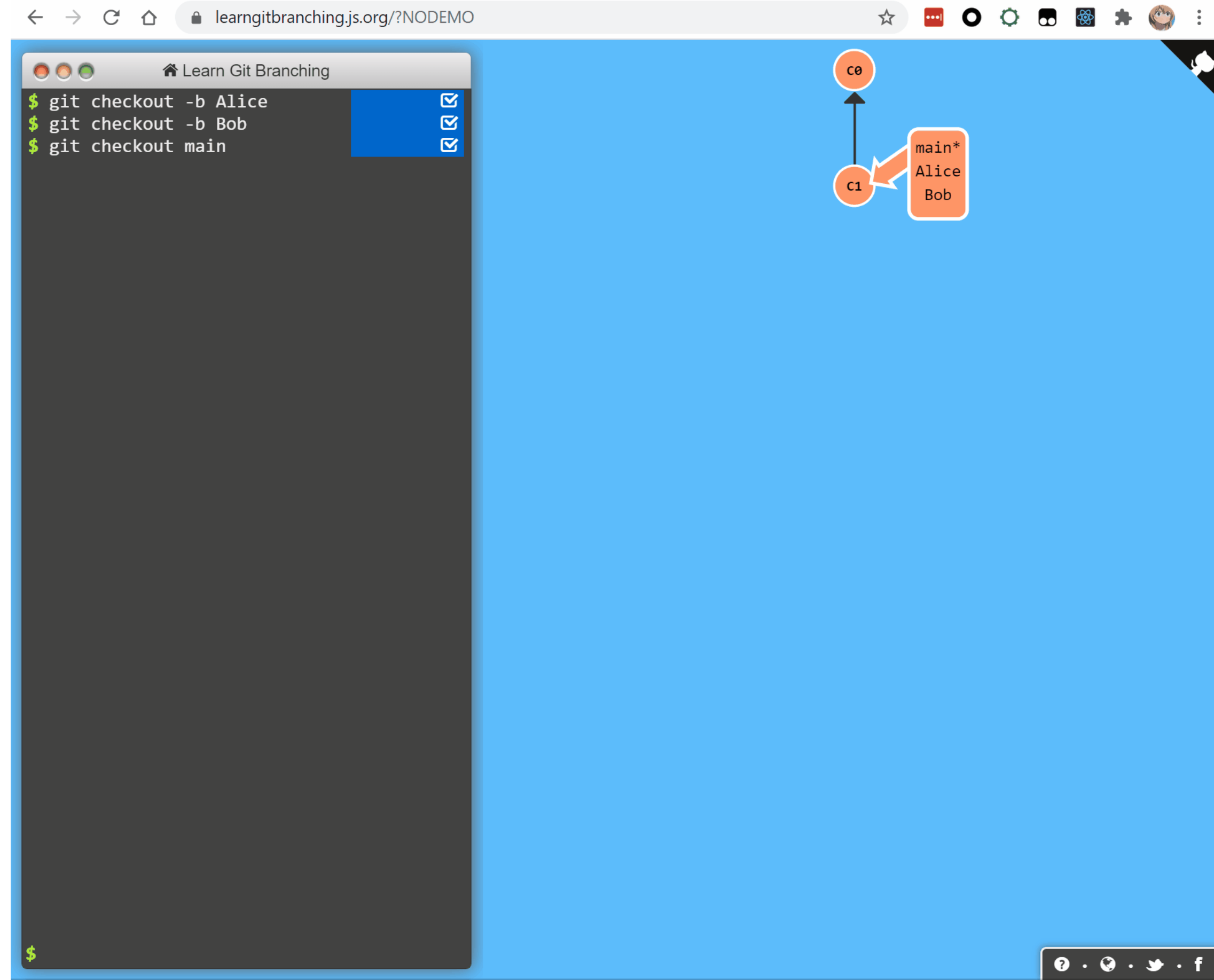
Demo in [learngitbranching.js.org](https://learngitbranching.js.org)

### Merge

Alice and Bob are two independent developers

They both contribute to branch in their own name

When work finished, they ``merge`` their commit into common branch named ``main``



## 2.2 Integrate changes, similar to Merge

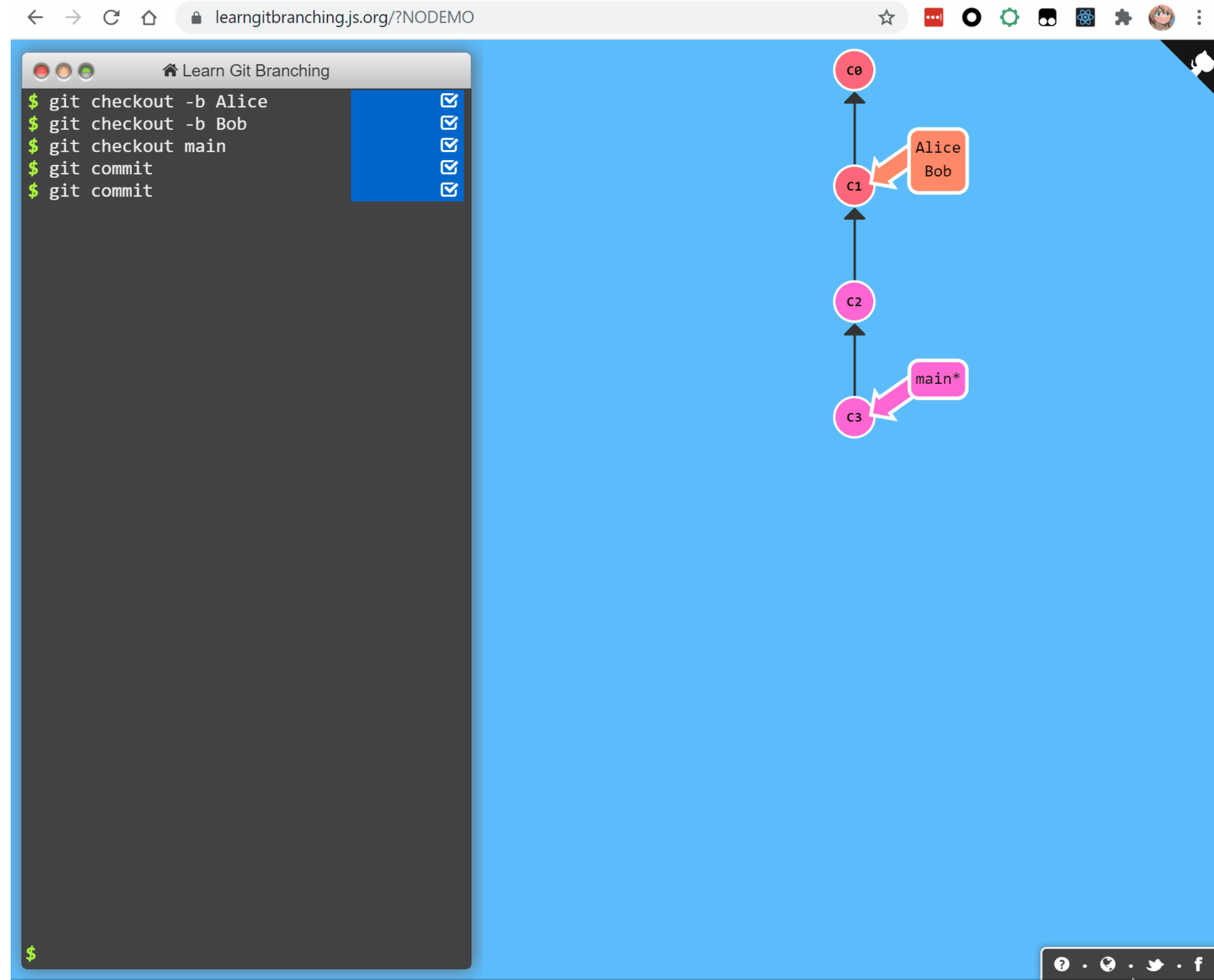
Demo in [learngitbranching.js.org](http://learngitbranching.js.org)

### Rebase

Alice and Bob are two independent developers

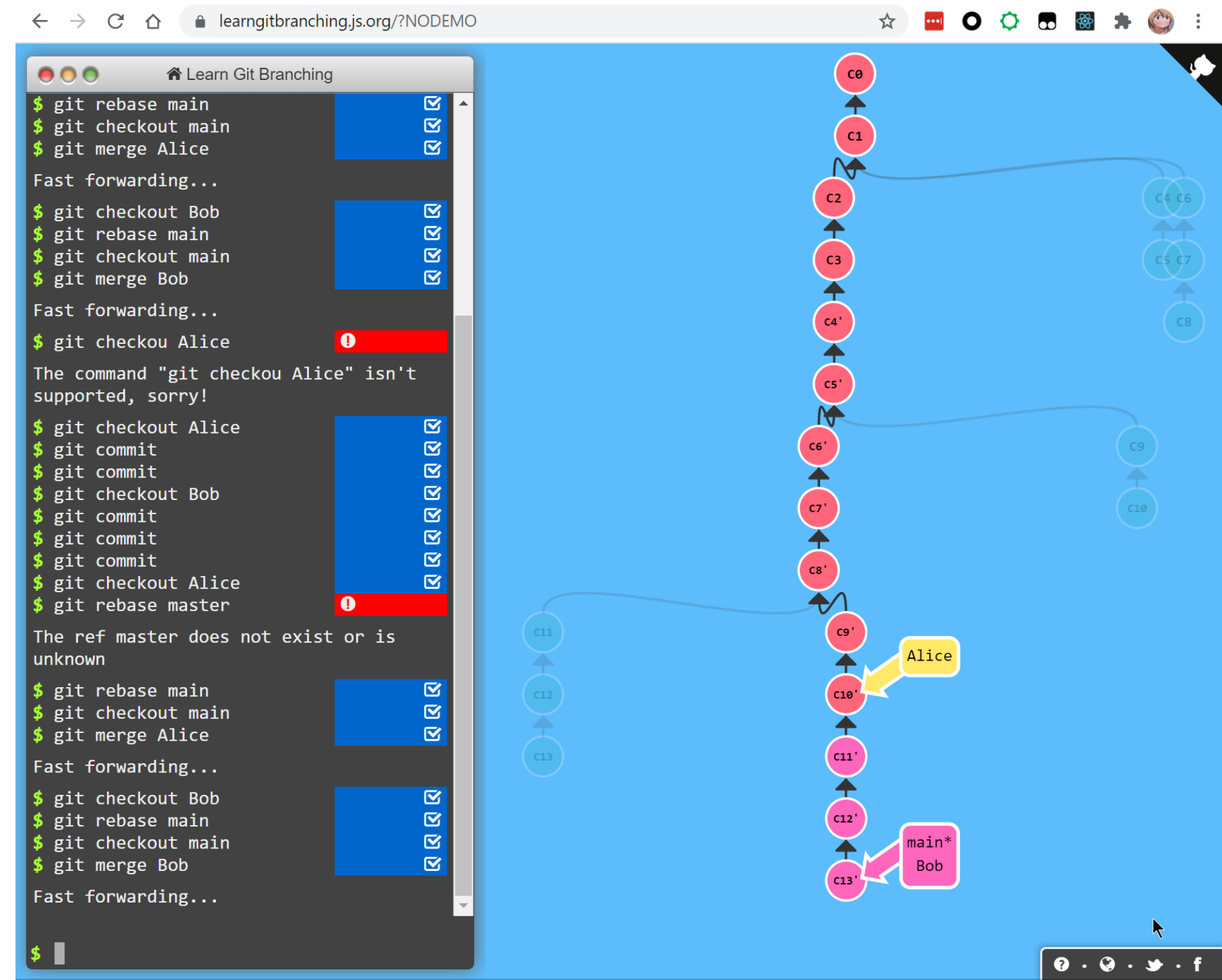
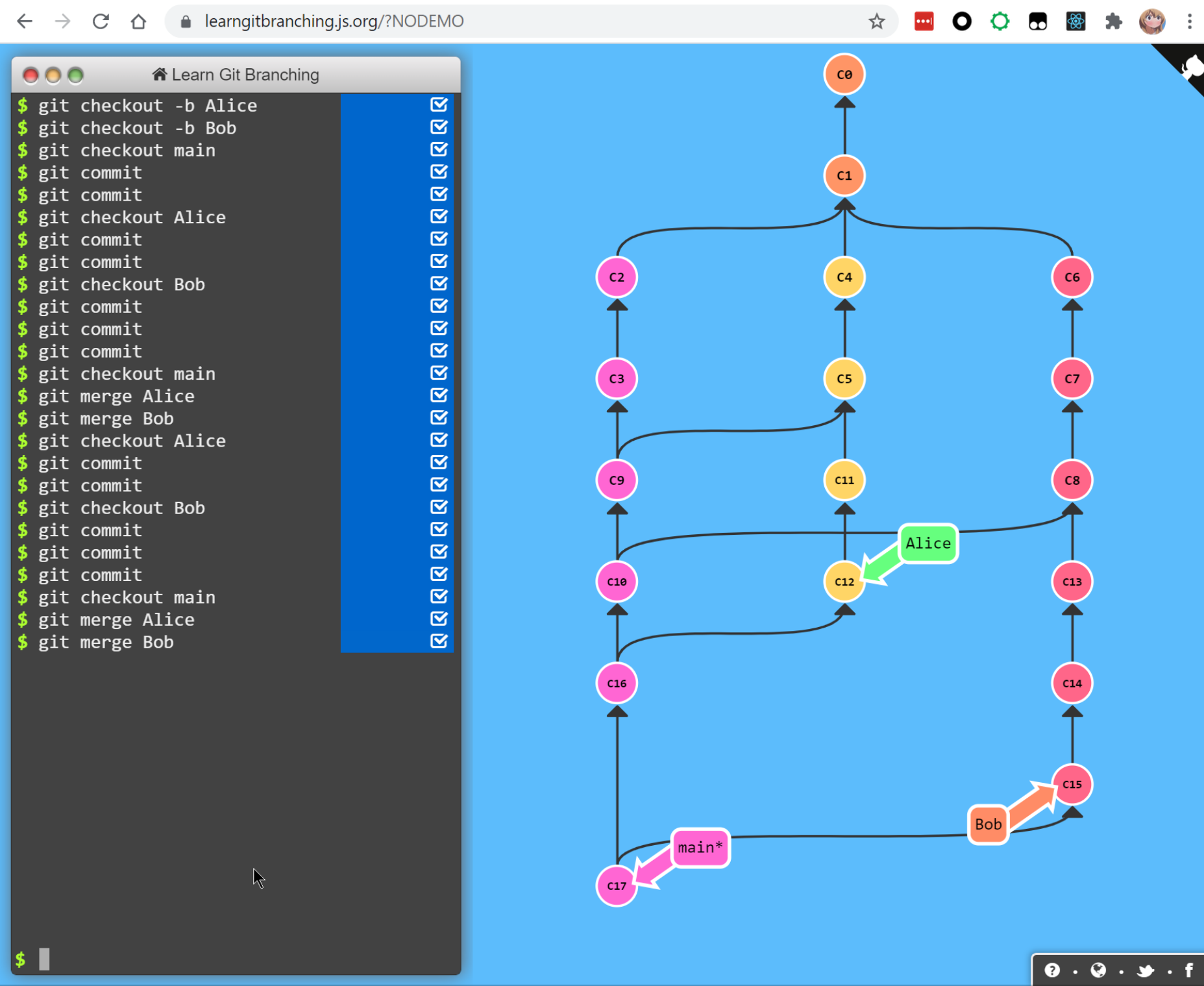
They both contribute to branch in their own name

When work finished, they `rebase` their own branch base on `main`, and then merge it into `main`



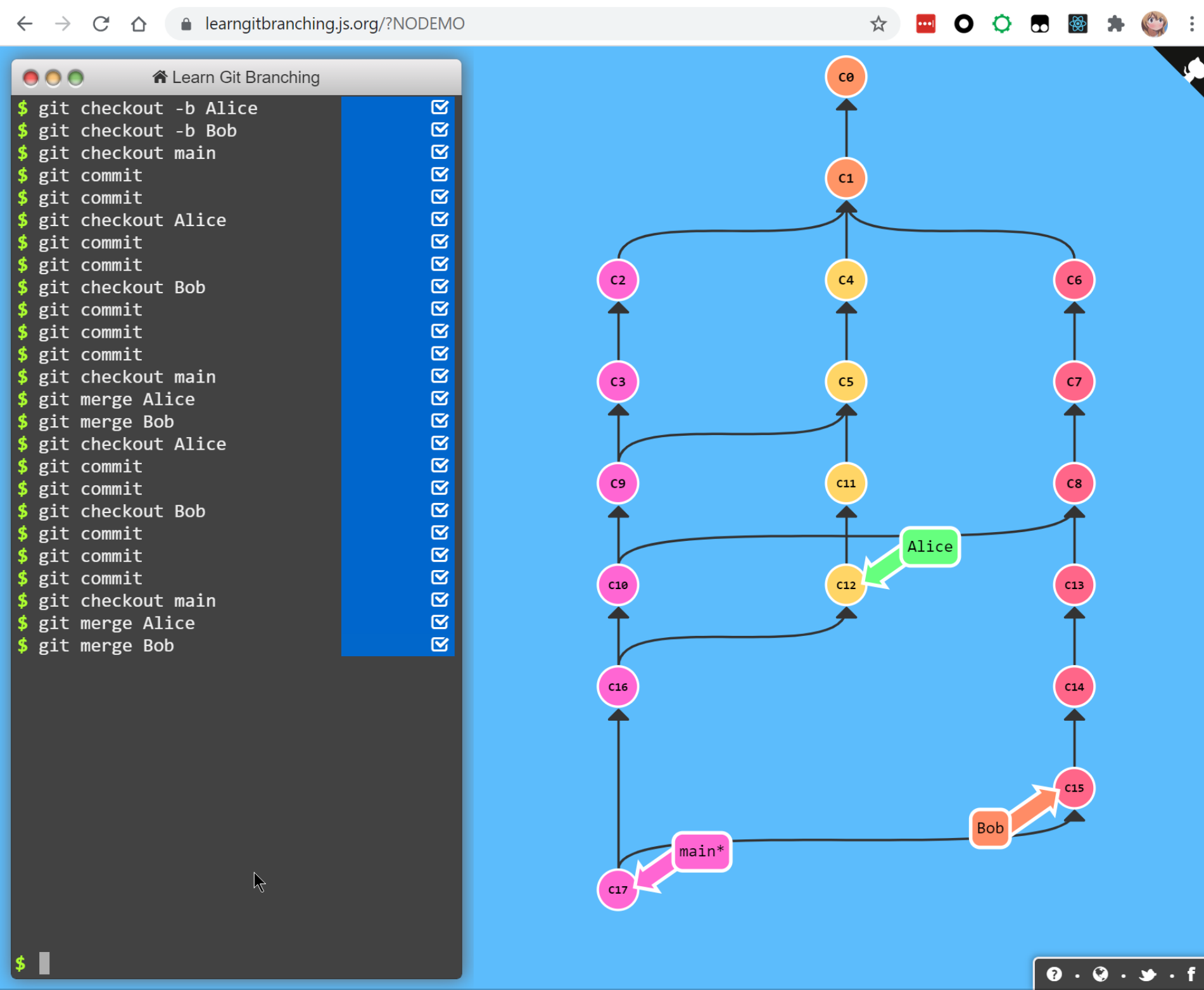
# 2.2 Integrate changes, similar to Merge

## Result: Merge v.s. Rebase



# 2.2 Integrate changes, similar to Merge

## Details in merge result



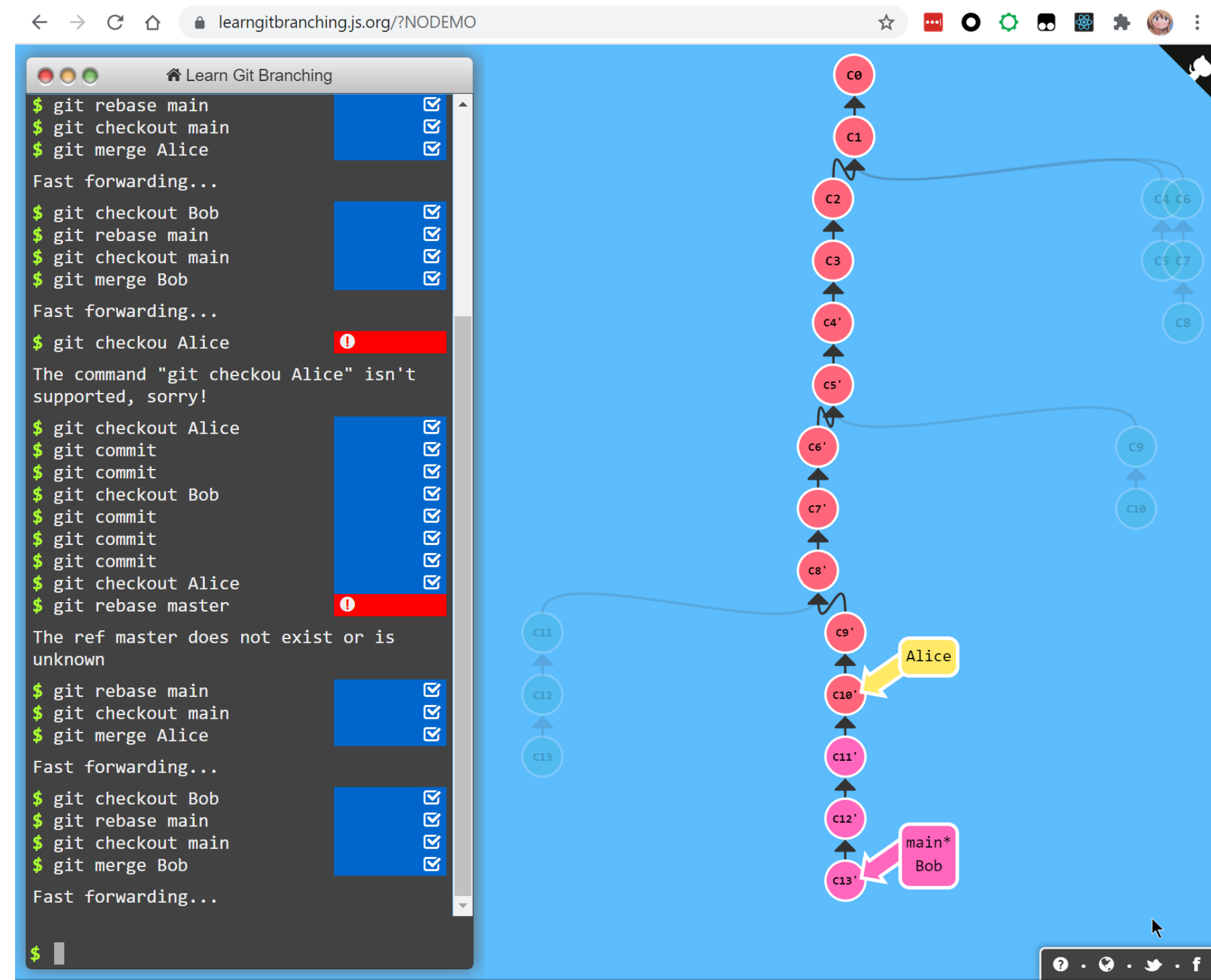
1. Merge will create `merge` commits like `c9`, `c10`, `c16`, `c17``
2. These merge commits contain nothing about file changing, they are just records of the merge of 2 branch
3. All commits history will be kept, without any modification



## 2.2 Integrate changes, similar to Merge

### Details in rebase result

1. Rebase will not create ``merge`` commits like merge command
2. All commit from dev branch will be modified, like ``c4'`, c5'`, c6'`, c7'`, etc.`, and the origin commits will disappear`
3. If everyone rebase ``main`` before merging into ``main``, git log will be a straight line rather than a reversed tree



## 2.2 Integrate changes, similar to Merge

### If conflicts exist

#### Rule

1. You need resolve them, both in merge and rebase
2. Conflict cases will be same, both in merge and rebase
3. Merge calls for one resolving step, because merge just **merge** 2 branch's ``HEAD``
4. Rebase maybe call for multiple resolving process, it depends on how many commits conflict

## 2.2 Integrate changes, similar to Merge

### If conflicts exist

#### Example

- Integrate branch ``Alice`` into branch ``main``
- ``Alice`` has 5 new commits before ``main``, 3 of them have conflicts with ``main``
  - If ``merge``, you need resolve all conflicts of 3 commits in only **ONE** step
  - If ``rebase``, you need resolve conflicts in 3 step, **ONE** step for **ONE** conflicted commit
- ``rebase`` like ``cherry-pick`` every commit into ``main`` in turn

# 3. Merge or rebase?

It depends

1. Differences
2. Scenarios

# 3.1 Differences between Merge & Rebase

- 1. Merge will save all origin history, Rebase will update them
- 2. In Rebase, each conflict must be resolved in separately in each commit
- 3. You need to resolve all conflicts in both

Compare with `Merge`

Type	Origin history	Resolve conflict resolved separately	All conflicts need resolved
Merge	✓	✗	✓
Rebase	✗	✓	✓

## 3.2 Scenarios

### Merge

1. History need to be saved.
  - a. merge a dev branch with many commits into release branch
  - b. merge your branch into a long term development common branch
2. Maybe you will cherry-pick or roll back in the future
3. You are not sure which method you should use



## 3.2 Scenarios

### Rebase

1. Squash some commits
2. Edit commit history (`git rebase -i HEAD~5`)
3. Pull from origin same name branch

It depends, most of time rebase is preferred, rebase makes history more clean.

```
git pull origin master  
=>  
git pull origin master --rebase/-r
```

Or change your `~/.gitconfig` with commands

```
git config --global pull.rebase true
```

## 4. How to xxx

Some cases we meet in daily development

1. Return to some commits
2. Move commit to another branch
3. Work with teammates on the same/different branch

## 4.1 Return to some commits

### 1. Reset HEAD

```
git reset [commit-id]
```

### 2. Check out a new branch from specified commit

```
git checkout -b [new-branch-name] [commit-id(default:HEAD)]
```

Both of them will move `HEAD` to the specified commit, in general, checkout a new branch is preferred

## 4.2 Move commits to another branch

``

``cherry-pick`` can move one or more commits to other branch.

If conflicts exist, you need to resolve them all, like a `git rebase`.

Move A...B commits to current branch, include A

```
git cherry-pick A^..B
```

# 5. Tools and configuration tricks

## 1. Tools

- a. Fork: beautiful git tool cross platform
- b. GitUI: operate every chunk in VIM/terminal
- c. diff-so-fancy & delta: better diff tool

## 2. Configuration tricks

- a. alias
- b. global gitignore
- c. global git hook
- d. delete origin branch in local automaticly
- e. use different configuration for different account depends on path

# 5.1 Tools demonstration

Fork: beautiful git tool cross platform



# 5.1 Tools demonstration

GitUI: operate every chunk in VIM mode

# 5.1 Tools demonstration

diff-so-fancy: Good-lookin' diffs

```
diff --git a/libs/header_clean/header_clean.pl b/libs/header_clean/header_clean.pl
```

```
modified: libs/header_clean/header_clean.pl
```

# 5.1 Tools demonstration

delta: A viewer for git and diff output

```
src/core/instance/render-helpers/resolve-scoped-slots.js

2
3 export function resolveScopedSlots (
4   fns: ScopedSlotsData, // see flow/vnode
5
6   hasDynamicKeys: boolean,
7   contentHashKey: number,
8   res?: Object
9 ): { [key: string]: Function, $stable: boolean } {
10   res = res || { $stable: !hasDynamicKeys }
11   for (let i = 0; i < fns.length; i++) {
12     const slot = fns[i]
13     if (Array.isArray(slot)) {
14       resolveScopedSlots(slot, hasDynamicKeys, null, res)
15     } else if (slot) {
16       // marker for reverse proxying v-slot without scope on this.$slots
17       if (slot.proxy) {
18
19
20       }
21     }
22     if (contentHashKey) {
23       res.$key = contentHashKey
24     }
25   }
26   return res
27 }
```

```
2
3 export function resolveScopedSlots (
4   fns: ScopedSlotsData, // see flow/vnode
5   res?: Object,
6   // the following are added in 2.6
7   hasDynamicKeys?: boolean,
8   contentHashKey?: number
9 ): { [key: string]: Function, $stable: boolean } {
10   res = res || { $stable: !hasDynamicKeys }
11   for (let i = 0; i < fns.length; i++) {
12     const slot = fns[i]
13     if (Array.isArray(slot)) {
14       resolveScopedSlots(slot, res, hasDynamicKeys)
15     } else if (slot) {
16       // marker for reverse proxying v-slot without scope on this.$slots
17       if (slot.proxy) {
18
19
20       }
21     }
22     if (contentHashKey) {
23       (res: any).$key = contentHashKey
24     }
25   }
26   return res
27 }
```

## 5.2 Configuration tricks

### 1. alias

Edit your `~/.gitconfig`

```
[alias]
  new = checkout -b
  cm = commit -m
  st = !echo 'untracked' && git ls-files . --exclude-standard --others && \
      echo '\nunstaged' && git diff --stat && \
      echo '\nstaged' && git diff --cached --stat
  com = checkout master
  unstage = reset HEAD--
  last = log-1 HEAD
  lg = log \
      --color \
      --graph \
      --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' \
      --abbrev-commit
  amend = commit --amend --no-edit
  dif = diff HEAD
```

You can use `git lg` to view all logs

## 5.2 Configuration tricks

2. global `.gitignore`

Edit your `~/.gitconfig`

```
[core]
  excludesfile = ~/.git-config/.gitignore_global
```

3. global hook

```
[core]
  hooksPath = ~/.git-config/hooks/
```

## 5.2 Configuration tricks

4. delete deleted remote branch in local automatically

```
[remote "origin"]  
  prune = true
```

5. use different configuration for different account depends on path

```
[includeIf "gitdir:~/code/github.com/"]  
  path = ~/.git-config/.gitconfig_github  
[includeIf "gitdir:~/code/gitlab.com/"]  
  path = ~/.git-config/.gitconfig_gitlab
```

# One more thing

shell enhancements

## zsh only

1. zsh highlight
2. zsh suggestion

## All shells matter

3. fzf: fuzzy finder in everything
4. z.lua: navigate faster by learning your habits
5. modern unix: Alternatives to common unix commands, a.k.a RIIR

# One more thing

shell enhancements

zsh highlight: Fish shell-like syntax highlighting for Zsh

Some examples:

Before: `% echo $'Hello, world\x21'`

After: `% echo $'Hello, world\x21'`

Before: `% echo 'Pay $5 to Joe'`  
`% echo "Pay $5 to Joe"`

After: `% echo 'Pay $5 to Joe'`  
`% echo "Pay $5 to Joe"`

Before: `% ( foo=42 }`

After: `% ( foo=42 }`

Before: `% echo 3> /proc/.../vm/drop_caches`


After: `% echo 3> /proc/.../vm/drop_caches`



# One more thing

shell enhancements

zsh autosuggestions: Fish-like fast/unobtrusive autosuggestions for zsh



A screenshot of a terminal window with a dark background. The prompt 'taiga@clinkz ~/code/github.com' is displayed in orange and green. Below it, a yellow triangle icon indicates an autosuggestion, followed by a white cursor character.


```
taiga@clinkz ~/code/github.com  
Δ
```

# One more thing

shell enhancements

fzf: A general-purpose command-line fuzzy finder

A terminal window with a dark background. The prompt is 'taiga@clinkz ~/code/github.com/keaising/slidev/git master [!?] via  v16.3.0'. Below the prompt is a yellow triangle icon and a grey cursor bar.

```
taiga@clinkz ~/code/github.com/keaising/slidev/git master [!?] via  v16.3.0
Δ █
```

# One more thing

shell enhancements

z.lua: A new cd command that helps you navigate faster by learning your habits

# Appendix: Tools in this slide

Glory to the Creators!

Name	URL
Powered by	<u>Slidev</u>
Theme	Self updated <u>@slidev/theme-seriph</u>
GIF recording	<u>ScreenToGif</u>
Host on	<u>GitHub Pages</u>

# Learn More

- Share this git notes
- Configure zsh from 0 to 1