

GIT DEEP DIVE

ABOUT ME



Jordan Duabe -- @j4ckofalltrades

- Software Engineer @ WiseTime
- Polyglot dev & open-source enthusiast
- Command-line ninja 🥷
- Has one too many MKBs 🖱️

AGENDA

- What is git (and some basic concepts)
- Diving into `.git`
- Configuring `git`
- The feature branch workflow
- `git` commands to add to your toolkit

WHAT IS GIT?

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

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.



xkcd: xkcd.com/1597/

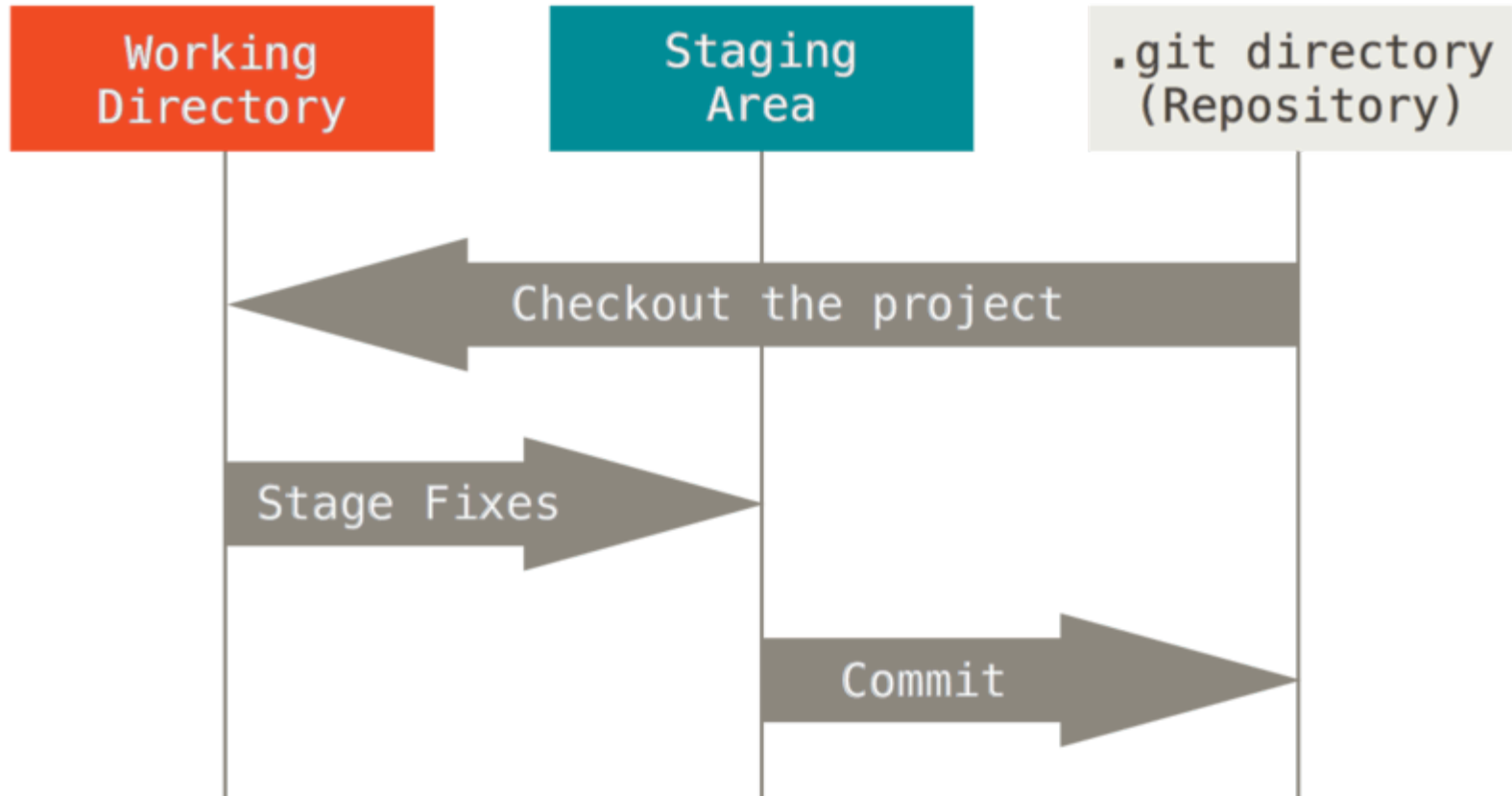
A SHORT HISTORY OF GIT

- Written in April 2005 by Linus Torvalds (creator of Linux)
- Maintained (since July 2005) by Junio Hamano
- v1.0 release was on December 2005
- v2.44 (latest release) at time of writing

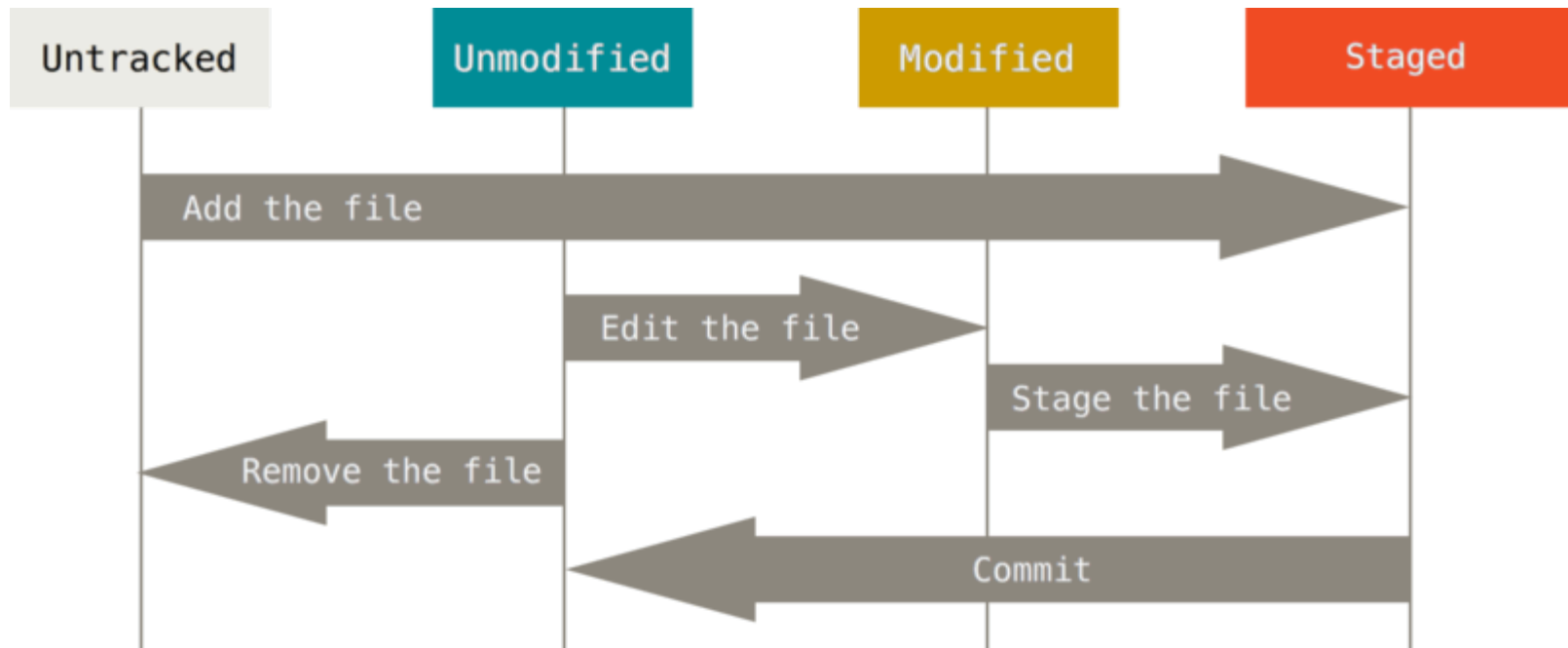
GET GIT WITH GIT

```
git clone https://github.com/git/git
```

THREE MAIN SECTIONS




STATUS LIFECYCLE OF YOUR FILES



DIVING INTO `.git`

git INTERNALS

 Here be dragons

```
myrepo/.git/
├── HEAD
├── config
├── hooks
│   └── pre-commit
├── index
├── objects
├── refs
│   ├── heads
│   └── main
```

Demo: Diving into `.git`

CONFIGURING GIT

INITIAL SETUP

The `git config` subcommand lets you get and set configuration variables that control all aspects of how Git looks and operates.

```
# system wide configuration
git config --system

# user specific configuration
git config --global

# repo specific configuration
git config [--local]
```

Each level overrides values in the previous level.

BASIC CONFIGURATION ITEMS

```
# identity
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com

# preferred editor
# defaults to "vi" on most systems
git config --global core.editor "code --wait"
```

CONDITIONAL CONFIG

```
[includeIf "gitdir:~/Developer/work/"]  
  path = ~/.gitconfig.work  
  
[includeIf "gitdir:~/Developer/oss/"]  
  path = ~/.gitconfig.oss
```




Alias your git commands

```
git config --global alias.br branch
git config --global alias.ci commit
git config --global alias.di diff
git config --global alias.lg 'log --oneline --graph'
git config --global alias.rs restore
git config --global alias.sh stash
git config --global alias.st 'status -sb'
git config --global alias.sw switch
```



Configure a global gitignore

```
git config --global core.excludesFile gitignore.global
```



Enable autocorrect

```
git config --global help.autocorrect <option>
```

```
# where option is one of
```

```
# 1. immediate (run closest command immediately)
```

```
# 2. prompt (ask for confirmation)
```

```
# 3. <timeout> (in deciseconds before running the command)
```

Configure auto push upstream for new branches

```
git config --global push.autoSetupRemote true
```

This removes the need for `git push --set-upstream <remote> <branch>`



Configure **delta** as the pager tool

git-delta

```
[pager]
  diff = delta
  log = delta
  reflog = delta
  show = delta
  blame = delta

[delta]
  features = side-by-side line-numbers decorations
  syntax-theme = Solarized (dark)
  plus-style = "syntax #003800"
  minus-style = "syntax #3f0001"
```



Configure commit signing

About commit signature verification

```
git config --global commit.gpgSign true
```

Update README.md



octocat committed on Sep 2, 2022 ✓

Verified



2f410e1



GETTING HELP

The following are equivalent ways of getting manual page (manpage) help for any git command.

```
# detailed help
git add --help

# concise tl;dr version
git add -h
```

THE FEATURE BRANCH WORKFLOW

GETTING A GIT REPOSITORY

```
# Option 1: Clone an existing repository
```

```
git clone <repo>
```

```
# Option 2: Initializing a repo in an existing directory
```

```
git init
```

```
git remote add <remote> <remote-url>
```

```
# to check if the remote was properly configured
```

```
git remote --verbose
```

```
# the update the remote URLs can also be updated
```

```
# generally used when the remote repository is renamed
```

```
# or when switching hosting providers e.g. GitHub
```

```
git remote set-url <remote> <new-remote-url>
```

CREATING A BRANCH

```
# create a new branch  
git branch <branch-name>
```

```
# create a branch and automatically 'switch' to it  
git switch --create <branch-name>
```

MOVING BETWEEN BRANCHES

```
git switch <branch-name>
```

```
git switch - # switch back to previous branch
```

WHAT ABOUT `git checkout`?

As of v2.23 (Q3 2019) `git checkout` was split out into two new commands, `switch` and `restore`.

Old command

`git checkout --
<pathspec>`

`git checkout <branch-
name>`

`git checkout -b
<branch-name>`

New command

`git restore <pathspec>`

`git switch <branch-
name>`

`git switch --create
<branch-name>`

git checkout

checkout: split part of it to new command 'switch'

"git checkout" doing too many things is a source of confusion for many users (and it even bites old timers sometimes). To remedy that, the command will be split into two new ones: switch and restore. The good old "git checkout" command is still here and will be until all (or most of users) are sick of it.

See the new man page for the final design of switch. The actual implementation though is still pretty much the same as "git checkout" and not completely aligned with the man page.

Signed-off-by: Nguyễn Thái Ngọc Duy <pclouds@gmail.com>

Signed-off-by: Junio C Hamano <gitster@pobox.com>

The **commit** that added the switch command to git.

IGNORING FILES

Tell git to ignore certain files and directories by creating a `.gitignore` file in your working directory.



This is applied in addition to a global gitignore (if configured).

VIEWING THE STATUS OF YOUR FILES

```
git status
```



For a tl;dr status

```
git status --short
```

REMOVING UNTRACKED FILES

`git clean`

```
# list files to be removed
```

```
git clean --dry-run
```

```
# remove files recursively (also remove untracked directories)
```

```
git clean -df
```



TRACKING NEW FILES AND/OR STAGING MODIFIED FILES

git add

```
git add --all # add everything
git add . # everything under the current directory
git add src # everything under the src directory
git add *.txt # all txt files under the current directory
git add test/*.js # all js files under the test directory
```



Use the **--dry-run** option to see what files will be staged

 **git stage** is an alias for **git add**

INTERACTIVE ADD

```
git add -p
```

 Demo: Selectively staging changes from a file

VIEWING CHANGES

git diff

```
git diff # show changes in the working tree
git diff --staged # show changes in the staging area
# show all changes (working tree and staging area)
git diff HEAD # or
git diff @ # @ is an alias for HEAD
```

COMMITTING CHANGES

`git commit`

```
git commit # opens up the default editor
git commit --message "wrote some code" # skips the editor
# bypass the staging area and commit directly
git commit --only --message "wrote some code" <paths>
# bypass the staging area and commit *all files* directly
git commit --all --message "wrote some code"
```

COMMIT MESSAGES

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

xkcd: xkcd.com/1296/

VIEWING COMMIT LOGS / HISTORY

`git log`

```
git log # show full history (paged view)
git log -<number-of-commits> # limit output to only n commits
git log --graph # draw a graphical representation
git log --patch # show changes made for the commit
git log --patch -1 # limit output to only the latest commit
```


VIEW A SPECIFIC COMMIT

```
git show <commit-hash>
```

UNDOING CHANGES



Discard changes from the working directory

```
git restore <pathspec>
```



Discard changes from the staging area

```
git restore --staged <pathspec>
```

UNDOING COMMITS

 **Soft reset (undo commit but keep changes)**

```
git reset <commit-hash>
```

 **Hard reset (undo commit and discard changes)**

```
git reset --hard <commit-hash>
```

AMENDING COMMITS

```
git commit --amend
```

Use the `--amend` flag to add files to an existing commit and/or update the commit message.



This creates a new commit!

To avoid possible conflicts and diverged branches, only run this command on unpushed commits.

reflog TO THE RESCUE

```
git reflog
```

Records when the tips of branches and other references were updated in the local repository.



Use this command to go back to a known "working" version of your code.

PUSHING CHANGES

```
git push <remote> <branch>
```

The `git push` command is used to upload local repository content to a remote repository.

FETCHING CONTENTS FROM THE REMOTE

```
# fetch all branches and changes from remote
git fetch <remote>

# fetch a specific branch from remote
git fetch <origin> <branch>
```

To list all remote branches, run `git branch --remotes`.

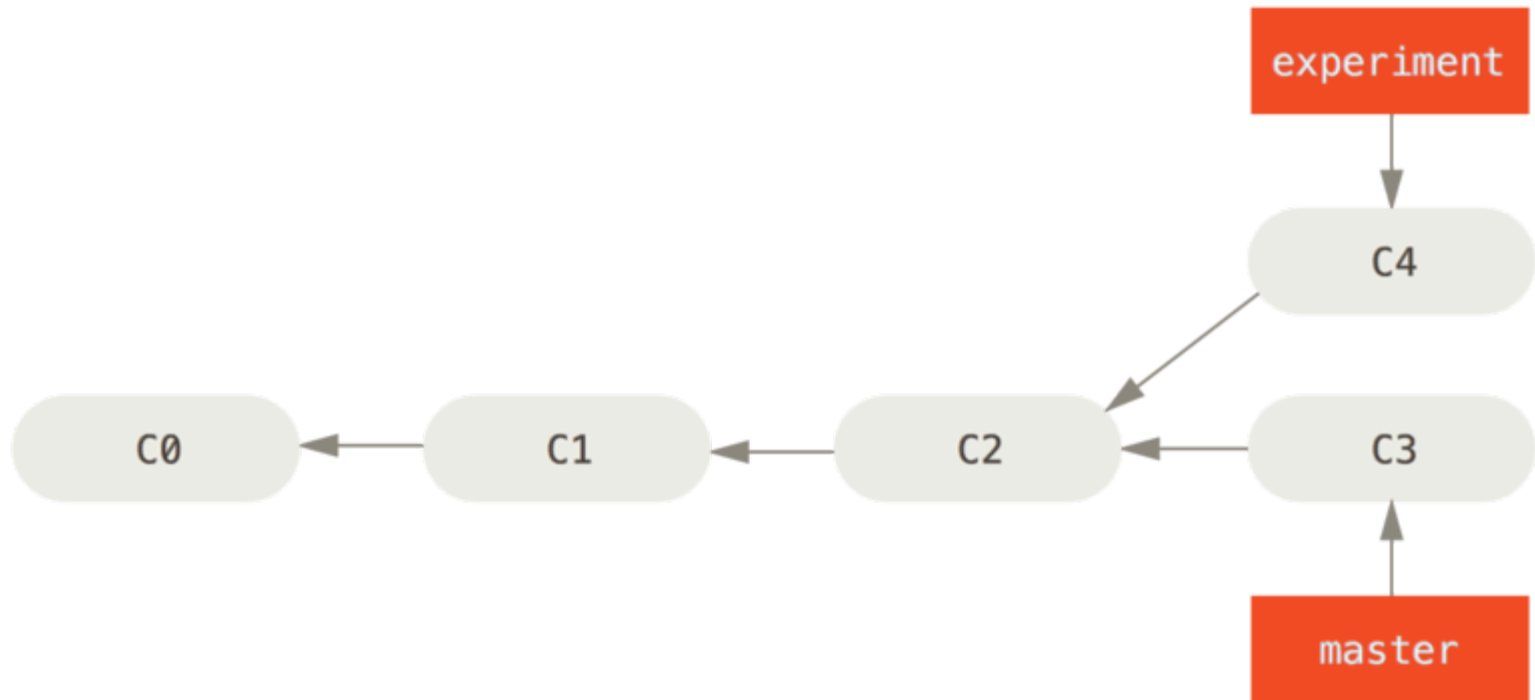
PULL

```
git pull <remote> <branch>
```

Basically an alias for:

```
# assuming the current branch is main  
git fetch origin main  
git merge origin/main
```


DIVERGED HISTORY



RECONCILE DIVERGED BRANCHES

```
hint: You have divergent branches and need to specify how to
reconcile them. You can do so by running one of the following
commands sometime before your next pull:
```

```
hint:
```

```
hint:   git config pull.rebase false  # merge
```

```
hint:   git config pull.rebase true   # rebase
```

```
hint:   git config pull.ff only       # fast-forward only
```

```
hint:
```

```
hint: You can replace "git config" with "git config --global"
set a default preference for all repositories. You can also pa
--rebase, --no-rebase, or --ff-only on the command line to
override the configured default per invocation.
```

```
fatal: Need to specify how to reconcile divergent branches.
```

MERGE

```
# Merging main to your feature branch
```

```
git switch main
```

```
git pull origin main
```

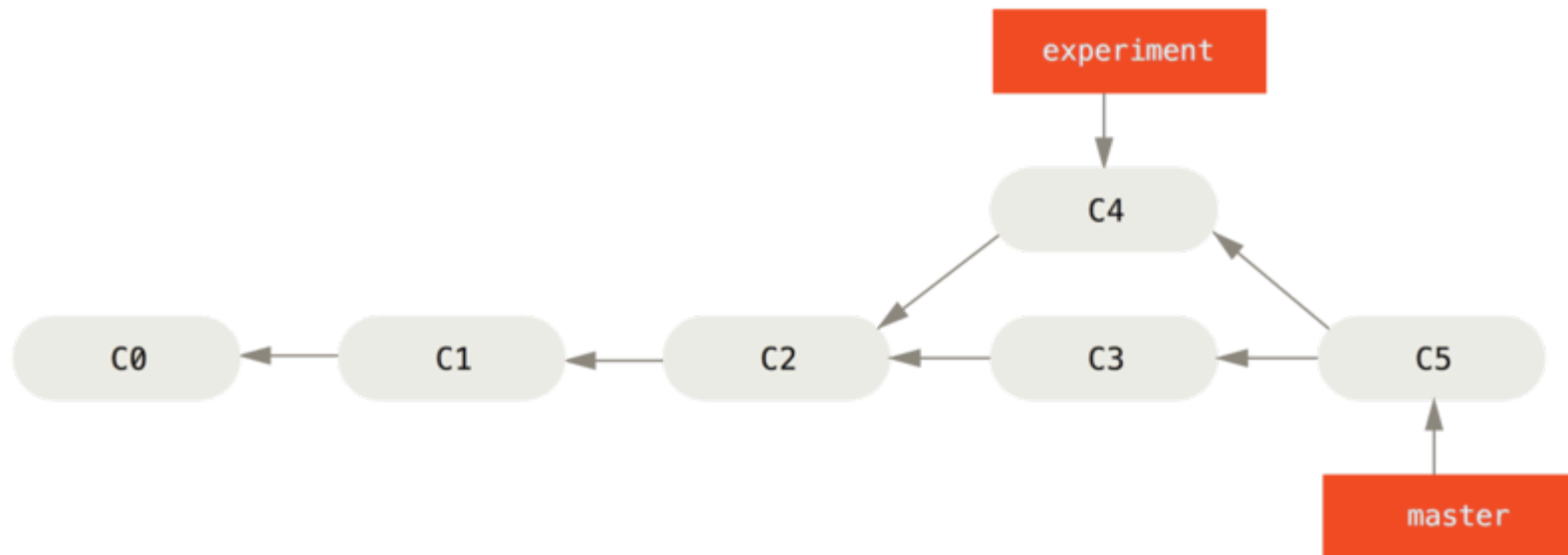
```
git switch my-feature-branch
```

```
git merge main
```

```
# Alternatively pull directly from your feature branch
```

```
git pull origin main
```

MERGE (VISUAL)



WHAT REALLY HAPPENS



REBASE

```
# Rebasing main on to your feature branch
git switch main
git pull origin main
git switch my-feature-branch
git rebase main

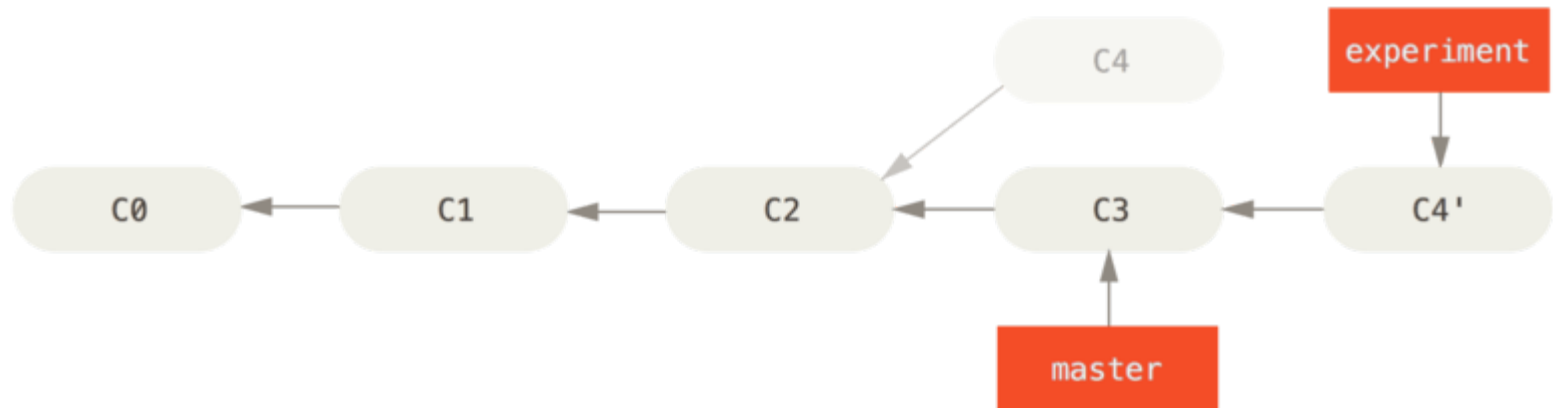
# Alternatively pull directly from your feature branch
git pull origin main --rebase
```



This command rewrites the commit history

Do not rebase commits that exist outside your repository and that people may have based work on.

REBASE (VISUAL)





Demo: Creating a pull request on GitHub

GIT COMMANDS TO ADD TO YOUR TOOLKIT

git blame

Show what revision and author last modified each line of a file.

```
git blame <file>

# limit to a specific line range
git blame -L <start,end> <file>

# limit to a specific function
git blame -L <funcname> <file>
```

git blame

Show what revision and author last modified each line of a file.

```
git blame <file>

# limit to a specific line range
git blame -L <start,end> <file>

# limit to a specific function
git blame -L <funcname> <file>
```

My favorite command

git blame

Show what revision and author last modified each line of a file.

```
git blame <file>

# limit to a specific line range
git blame -L <start,end> <file>

# limit to a specific function
git blame -L <funcname> <file>
```

My favorite command

... until I see the result.

git grep

Print lines matching a pattern.

```
git grep <search-term> [pathspec]
```

git revert

Revert some existing commits.

```
git revert <commit>
```

```
# Revert a specific commit
```

```
git revert <commit-hash>
```

```
# Revert a commit relative to the latest commit
```

```
# This reverts the 4th latest commit
```

```
git revert HEAD~3
```

```
# Open an editor and edit the default message
```

```
git revert --edit <commit>
```

git stash

Stash the changes in a dirty working directory away.

```
git stash create <message>

# Applying stashes
git stash list
# apply specific stash from list
git stash apply stash{0}
# apply stash at top of stack (LIFO) then delete it
git stash pop
```

git cherry-pick

Apply the changes introduced by some existing commits (from some other branch).

```
git cherry-pick <commit-hash>
```


git bisect

Use binary search to find the commit that introduced a bug.

```
git bisect <subcommand> <options>

# Basic commands: start, bad, good
git bisect start
git bisect bad
git bisect good <commit> # can also be a tag

# Cleaning up
git bisect reset
```

ATTRIBUTION

- Images from [xkcd](#) and [louim](#)
- Diagrams from the [Pro Git](#) book

RECOMMENDED READING

- [Pro Git by Scott Chacon](#)
- [Julia Evans' blog](#)
- [git documentation](#)

SLIDES



QUESTIONS

???

In case of fire



 1. `git commit`

 2. `git push`

 3. `leave building`

