INTRODUCTION TO GIT

Øyvind Kheradmandi

Håkon Wikene

AGENDA

Introduction and history

How Git works “under the hood”

Basic commands

Exercise 1 – Basics

Branching

Merging and rebasing

Exercise 2

Advanced topics

AGENDA

- Introduction and history
- How git works “under the hood”
- Basic commands
- Exercise 1 – Basics
- Branching
- Merging and rebasing
- Exercise 2
- Advanced topics

What is Git

- Distributed version control system
- The whole repository is on every machine
- Most commands are run locally
- Made to be fast and suited for multiple people working on the repo simultaneously and independently of each other

History

- Created by Linus Torvalds in 2005
- Made for development of the Linux kernel
- Removal of the free use of BitKeeper spawned the creation of Git
- No existing free VCSs at the time met the demands of the Linux kernel

Some goals of Git

- **Speed**
- **Simple design**
- **Strong support for non-linear development (thousands of parallel branches)**
- **Fully distributed**
- **Able to handle large projects like the Linux kernel efficiently (speed and data size)**

AGENDA

Introduction and history

How git works “under the hood”

Basic commands

Exercise 1 – Basics

Branching

Merging and rebasing

Exercise 2

Advanced topics

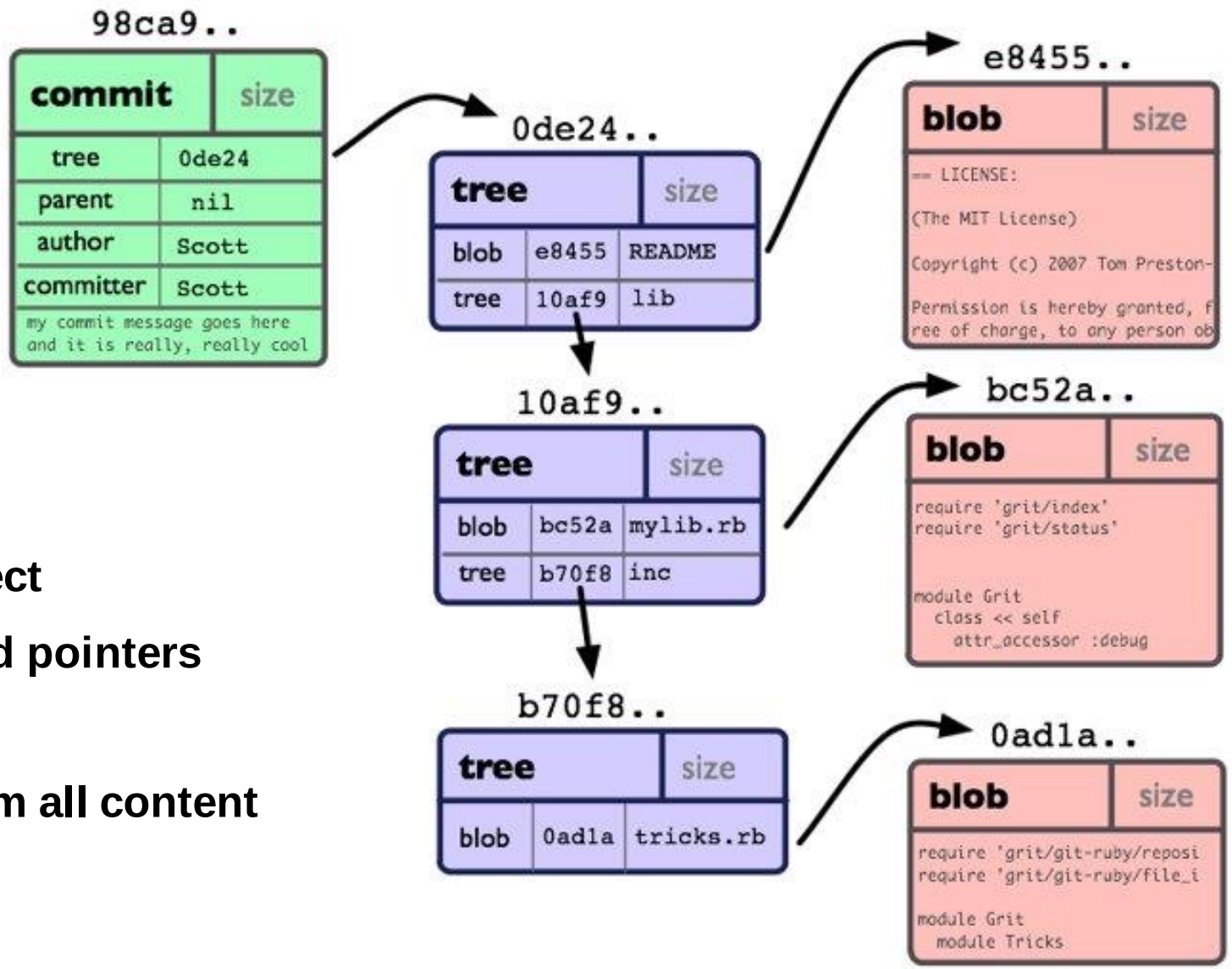
… but first a few terms

- repository – a collection of commits
- working tree – a folder with an associated repository
- commit – a snapshot of the working tree at some point
- checkout – updating working tree from repository
- branch – a name for a specific commit (that changes)
- tag – also a name for a commit (but constant)
- master – typically used as the mainline branch
- HEAD – what is currently checked out
- index – staging area for next commit
- man gitglossary

git init

```
.git/
   branches/
  config
  description
  HEAD
  hooks/
  index
  info/
     exclude
  logs/
  objects/
     pack/
  refs/
     heads/
     tags/
```

Stores snapshots



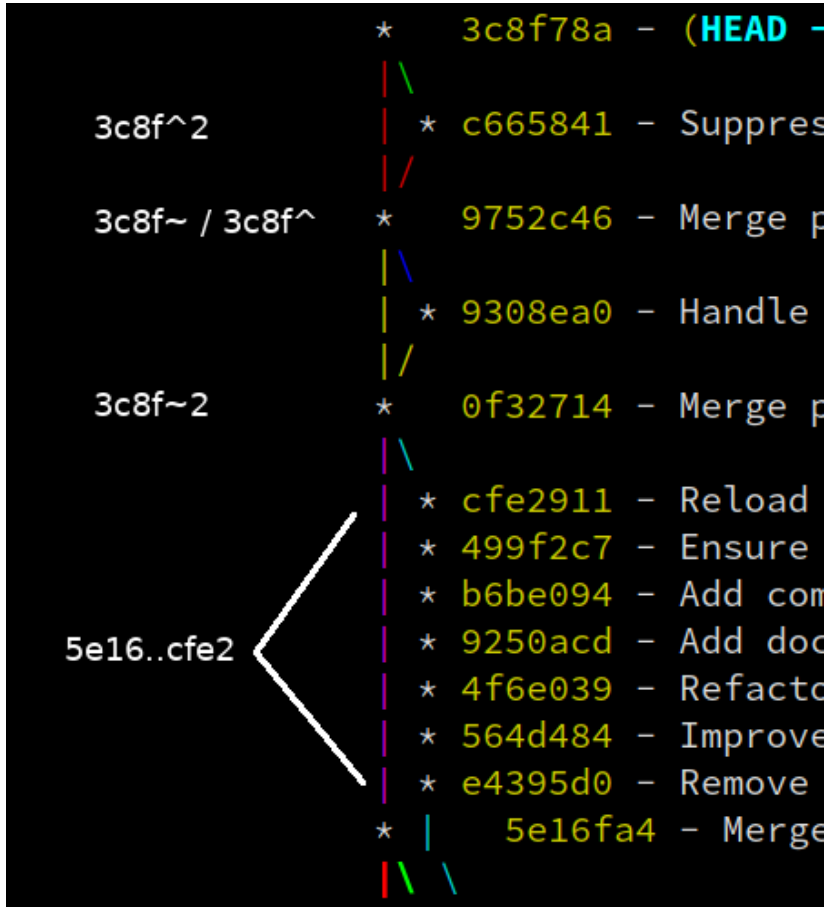
- Takes a hash of the content
- Stores the content as objects
- Any change makes a new object
- Tree-objects for file names and pointers
- Doesn't track renames
- Commit hash is computed from all content in the commit
- Packfiles

References

- Pointers to a commit
- Branches are references
- Remote branches
- Tags, simple and annotated
- HEAD
- Detached HEAD
- FETCH_HEAD

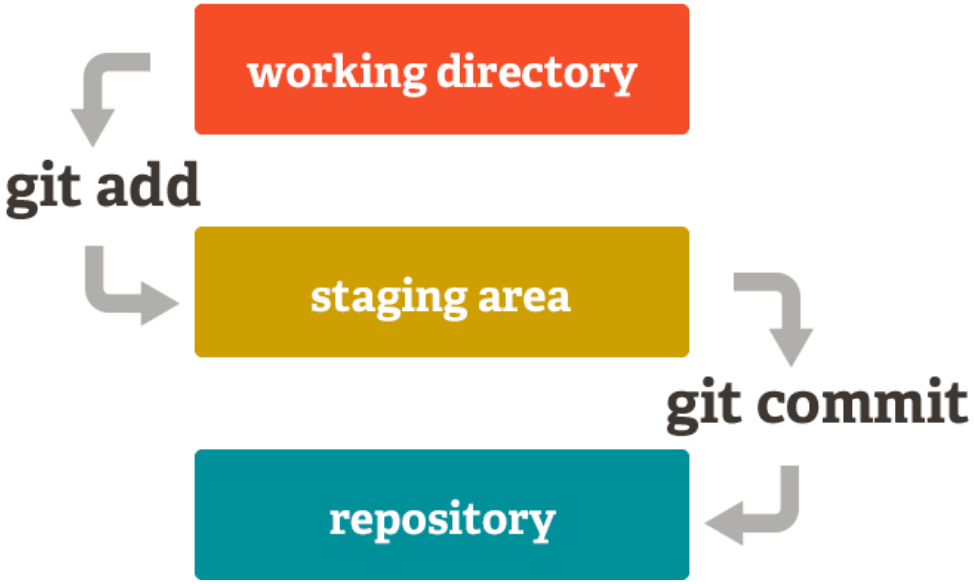
Revisions

- Any reference
- Any hash
- `<rev>^ vs. <rev>~`
- `<rev1>..<rev2>`
- `<rev>:<path>`
- `<rev>@{n}`
- `<rev>@{date}`
- `<rev>^{/text}, :/text`
- `man gitrevisions`



Staging area

- The next content to commit



AGENDA

Introduction and history

How git works “under the hood”

Basic commands

Exercise 1 – Basics

Branching

Merging and rebasing

Exercise 2

Advanced topics

Fork vs Clone

- **Fork creates a copy of the repository to your own Github repository**
- **Can be used when:**
  - You want to use others work as a basis of your own.
  - You want to contribute to other repositories, but do not have access to the repository.

Fork vs Clone

Itera / git-fagkveld

Unwatch 46 | Star 0 | Fork 34

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

No description, website, or topics provided.

Edit

Add topics

16 commits | 2 branches | 0 releases | 2 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

trygveaa Add slides from presentation — Latest commit 87be51a on 9 Mar

exercise-1	Write README for exercise 1	6 months ago
exercise-2	Add minus operation	6 months ago
README.md	Update README.md	6 months ago
git-presentation.pdf	Add slides from presentation	6 months ago

README.md

Clone

- **`git clone`** https://github.com/<your-username>/git-fagkveld

Adding files

- **git add <file>** - adds file to staging area
- **git add .** - adds all files in current folder to staging area
- **git add -A** - adds all changed files in repository to staging area

Show status

- **`git status`** – List the files you've changed and those you still need to add or commit

```
C:\git\git-fagkveld [feature/testing +0 ~1 -0 | +0 ~1 -0 !]> git status
On branch feature/testing
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   exercise-1/my-file.txt

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:   exercise-2/calc.js
```

Committing changes

- **git commit** - opens default editor to enter commit message
- **git commit -m "*<commit message>*"** - commit with message in one line

Committing changes

- **`git commit –m "Some commit message"`**

```
C:\git\git-fagkveld [feature/testing ↑1 +0 ~1 -0 !]> git status
On branch feature/testing
Your branch is ahead of 'origin/feature/testing' by 1 commit.
  (use "git push" to publish your local commits)
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:   exercise-2/calc.js

no changes added to commit (use "git add" and/or "git commit -a")
```

Working with remotes

- **`git push`** - move your changes to the remote branch

Working with remotes

```
C:\dev\git-fagkveld\exercise-1 [working-with-remotes ↑1]> git push
To https://github.com/itera/git-fagkveld
 ! [rejected]        working-with-remotes -> working-with-remotes (fetch first)
error: failed to push some refs to 'https://github.com/itera/git-fagkveld'
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.
```

Working with remotes

- **git fetch -** download changes to your local repository
- **git pull -** fetch and merge changes
- **git pull --rebase** – fetch and rebase changes

Best practices

- Commit early, commit often
- One commit should only include related changes
- Include later fixups for that commit in it
  - Given that it hasn't been pushed yet
- Describe what the commit does and why it should be done

Best practices – Git recommends

- **Imperative form**
- **Wrap the lines to about 72 characters**
- **From Pro Git** (https://git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project)

```
Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary.  Wrap it to
about 72 characters or so.  In some contexts, the first
line is treated as the subject of an email and the rest of
the text as the body.  The blank line separating the
summary from the body is critical (unless you omit the body
entirely); tools like rebase can get confused if you run
the two together.

Further paragraphs come after blank lines.

  - Bullet points are okay, too

  - Typically a hyphen or asterisk is used for the bullet,
    preceded by a single space, with blank lines in
    between, but conventions vary here
```

AGENDA

Introduction and history

How git works “under the hood”

Basic commands

Exercise 1 – Basics

Branching

Merging and rebasing

Exercise 2

Advanced topics

Exercise 1

https://github.com/ltera/git-fagkveld

Read the “Getting started” guide.

AGENDA

Introduction and history

How git works “under the hood”

Basic commands

Exercise 1 – Basics

Branching

Merging and rebasing

Exercise 2

Advanced topics

Branching - Commands

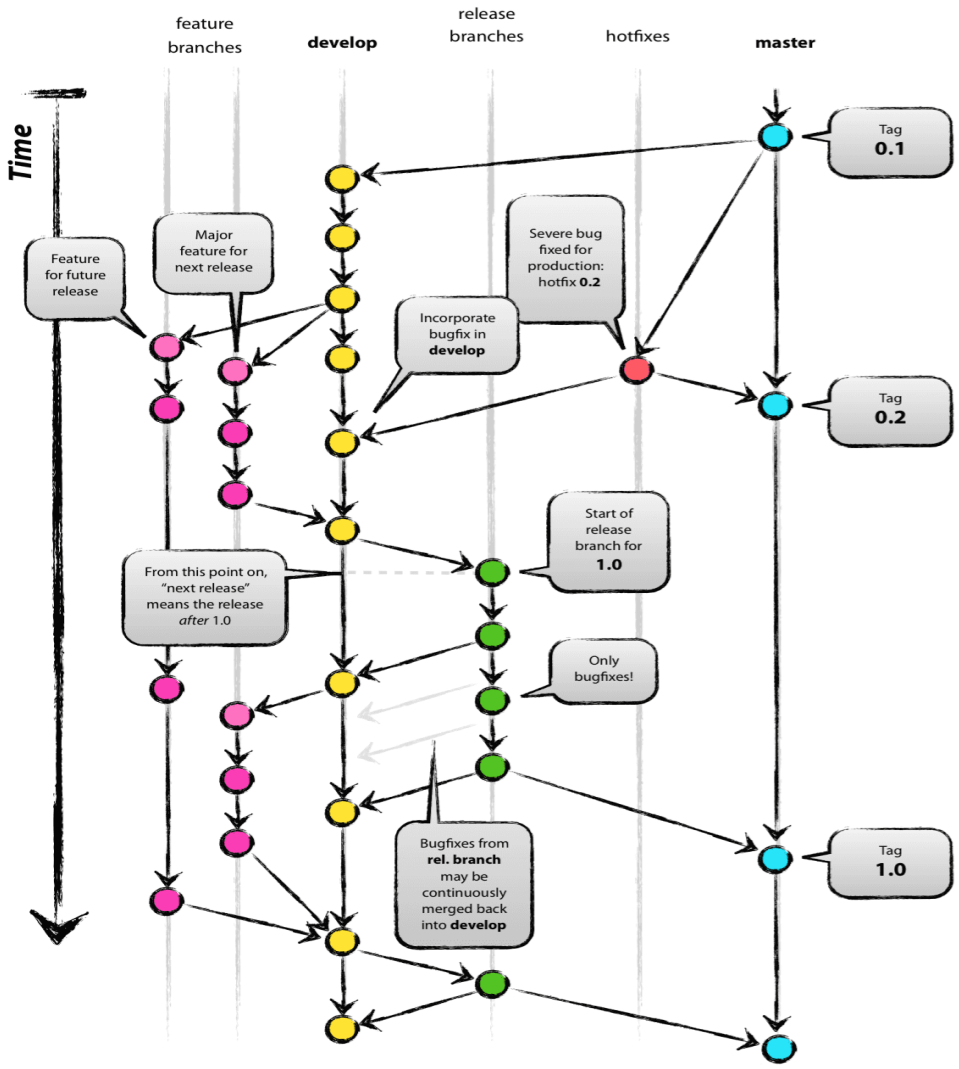
- **git branch <branch-name>** - creates a new local branch
- **git checkout <branch-name>** - checkout the new branch
- **git checkout –b <branch-name>** - creates and checkout new branch
- **git push --set-upstream origin branch-name** - sets the remote branch of the current local branch
- **git branch -d <branch-name>** - delete branch locally
- **git push origin --delete <branch-name> -** delete branch on remote

Branching

- **Main branches** - e.g. master
- **Feature branches** - used when developing new features e.g. feature/my-new-feature
- **Long-lived branches** – e.g. develop

Branching models

- **Git flow**
- **Simpler alternatives**



AGENDA

Introduction and history

How git works “under the hood”

Basic commands

Exercise 1 – Basics

Branching

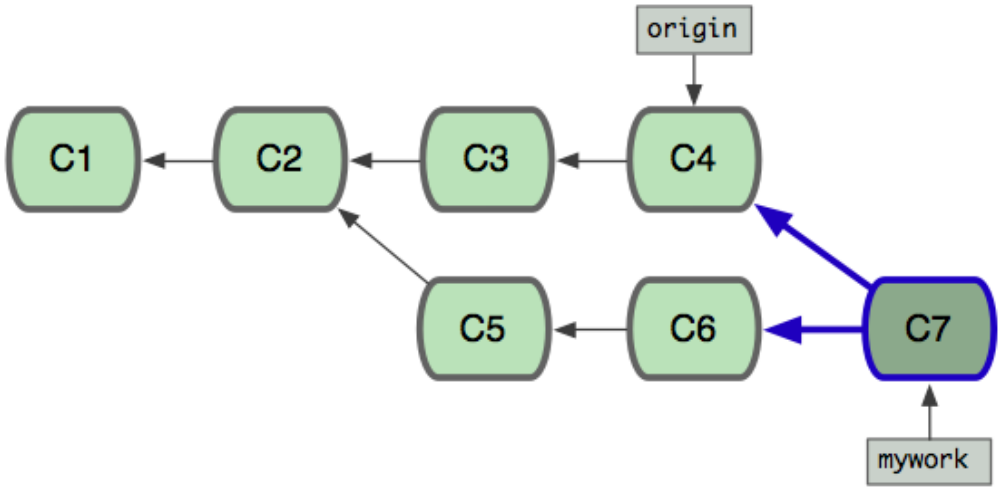
Merging and rebasing

Exercise 2

Advanced topics

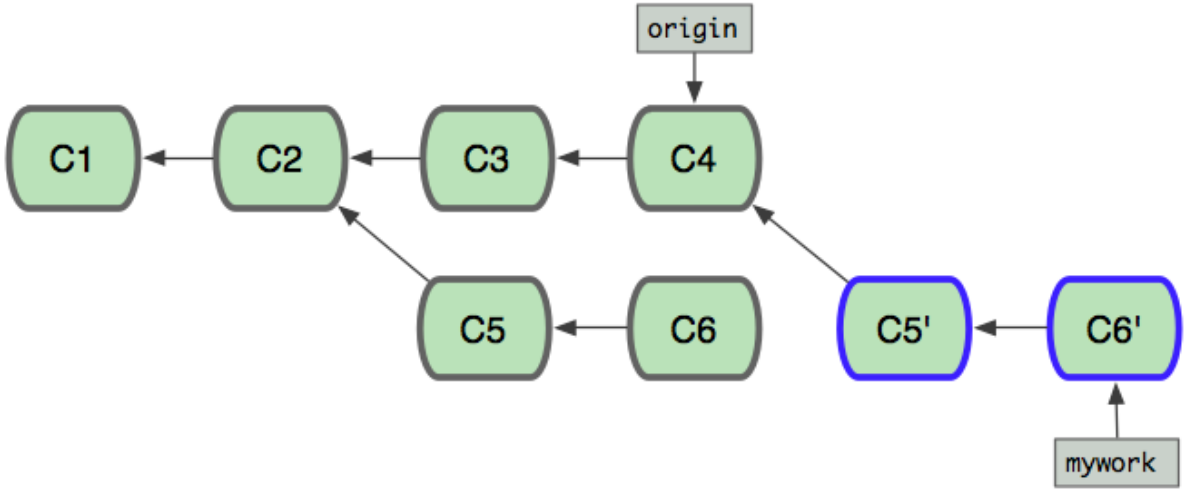
How does a merge work

- Fast-forward vs merge commit (--ff-only, --no-ff)
- Parents of a merge commit
- Merge commits stores all of the content, as any other commit
  - Evil merges



What is a rebase

- Starts from a specific commit
- Reapplies the commits on top of that
- Allows you to make changes to earlier commits



Merge conflicts

- Appears when two changes are done in the same place
- Git can't know what is correct, as it doesn't know semantics
- Resolve directly in file, or with a tool
  - `mergetool`
  - `merge.conflictstyle = diff3`
- You can ignore whitespace: -Xignore-space-change

When to use merge vs. rebase

- Use rebase to keep history clean in your own branches
- Merge in feature branches
- Merge long-lived branches
- Rebase when fetching from upstream
- Rebase to keep feature branch up-to-date (generally)
- git merge --squash
- Different opinions. Some argue never to merge, some argue never to rebase

Disadvantages with merge/rebase to keep your branch up-to-date

- Disadvantages with rebase
  - "Lies" about the history
  - Individual commits may not build
  - May cause more conflicts
  - Makes it hard to find a good place to check out
  - Timestamps in history will be out-of-order
- Disadvantages with merge
  - History can become harder to read

How to change earlier commits

- `git commit --amend`
- `git commit --fixup + git rebase --autosquash`
- `git rebase -i`
- **Be careful when rebasing merge commits**
- **Removing commits**

Merging/splitting the last two commits

- Splitting the last commit
  1. git reset -p HEAD~
  2. git commit --amend
  3. git commit
- Merging the last two commits
  1. git rebase -i HEAD~2
  2. … or git reset --soft HEAD~ && git commit --amend

AGENDA

Introduction and history

How git works “under the hood”

Basic commands

Exercise 1 – Basics

Branching

Merging and rebasing

Exercise 2

Advanced topics

Exercise 2

- Amend a commit
- Fix an earlier commit
- Use interactive rebase
- Move a commit to another branch
- Merge back to master
- Resolve conflicts

AGENDA

Introduction and history

How git works “under the hood”

Basic commands

Exercise 1 – Basics

Branching

Merging and rebasing

Exercise 2

Advanced topics

Reflog

- History of all your changes
- Default reflog, for HEAD
- Reflog for each branch

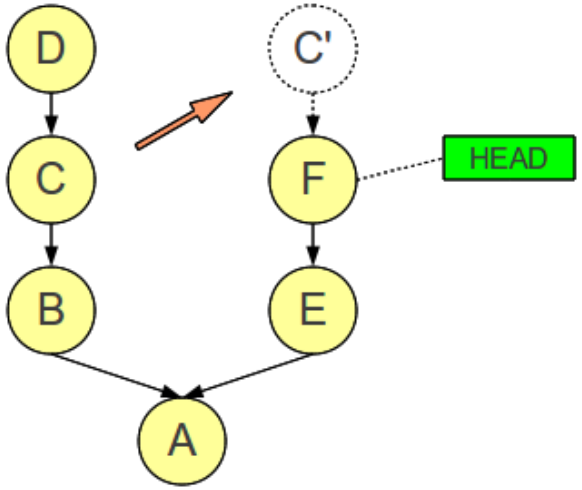
```
0f32714 HEAD@{0}: pull: Fast-forward
5e16fa4 HEAD@{1}: pull --prune: Fast-forward
3eff1de HEAD@{2}: pull: Fast-forward
d02bde0 HEAD@{3}: pull --prune: Fast-forward
5608e22 HEAD@{4}: checkout: moving from master-
f4e2ab2 HEAD@{5}: merge trygveaa/multiline-mess
1d4511d HEAD@{6}: merge FETCH_HEAD: Merge made
5608e22 HEAD@{7}: checkout: moving from master
5608e22 HEAD@{8}: pull --prune: Fast-forward
42b7746 HEAD@{9}: checkout: moving from master-
5c20f93 HEAD@{10}: merge FETCH_HEAD: Merge made
```

Stash

- Records state of working tree and resets to HEAD
- Useful when...
  - Moving changes to different branch
  - Need to do something small on a different branch
- refs/stash
- git stash list / show / drop / push / pop / clear

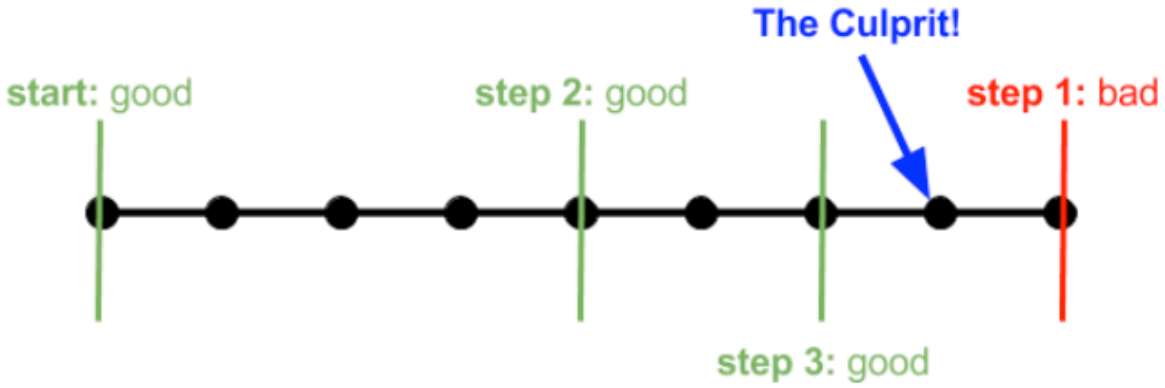
Cherry-pick

- Copies commits from one branch to another
- The commit hash changes



Bisect

- Used to figure out when bugs are introduced
- Does a binary search through the history



Worktrees

- Allows you to have multiple checkouts of a repo
- All the checkouts shares the git directory
  - I.e. commits, references, stash, etc.
- Not allowed to check out a branch multiple places simultaneously

Tags

- Simple tags
  - Refs/heads/tags
- Annotated tags
  - Stores date, tagger, message, signature
- git tag to list tags
- git tag to create tag, "-a" to create annotated tags

Thanks!

Questions?