HPS

https://hps.vi4io.org

Christian Köhler

Introduction to Git

A free and open source VCS

Table of contents

1 Introduction

2 Setup and Configuration

3 Creating commits

4 Managing branches

Learning Objectives

- Know the purpose of VCS in general and Git in particular
- Set up and configure Git
- Create local and clone remote repositories
- Craft and review commits
- Interact with local and remote branches

Why version control systems (VCS)?

- Track changes in your project
- Be able to jump to the last known working state
- Explore different (potentially experimental “throwaway”) branches of development
- Attach meaningful notes to each set of changes aka. “commit”
- Collaborate with others

What is Git?

- Initial release in 2005 by Linus Torvalds
- Used for developing the Linux Kernel
- Previously: BitKeeper (proprietary)

What is Git?

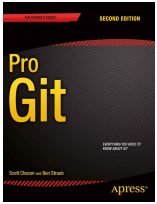
- Initial release in 2005 by Linus Torvalds
- Used for developing the Linux Kernel
- Previously: BitKeeper (proprietary)

What is Git?

- Initial release in 2005 by Linus Torvalds
- Used for developing the Linux Kernel
- Previously: BitKeeper (proprietary)
- `https://git-scm.com/`
  - Documentation
  - Git command reference

What is Git?

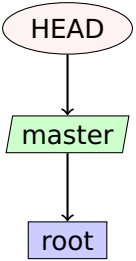
- Initial release in 2005 by Linus Torvalds
- Used for developing the Linux Kernel
- Previously: BitKeeper (proprietary)
- `https://git-scm.com/`
  - Documentation
  - Git command reference
  - Ebook: *Scott Chacon, Ben Straub - Pro Git*



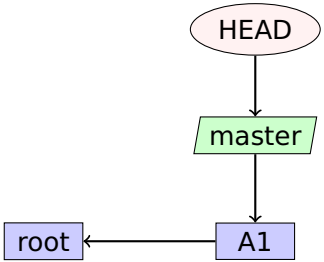
Git terminology

- Git projects are called *repositories*.
- Fully distributed, i.e. each local clone contains the entire project history
- Versions of the managed file tree are called *commit*s.
- They form a graph where each new commit has at least one parent.
- Branches are easily created - they’re just named pointers to commits.
- `HEAD` points to the branch that will receive the next commit.
- Important commits (e.g. release versions) can get a named (even annotated, signed) *tag* pointing to them.

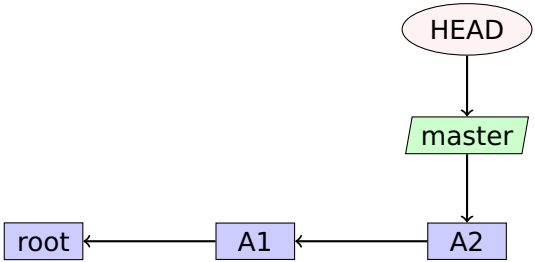
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



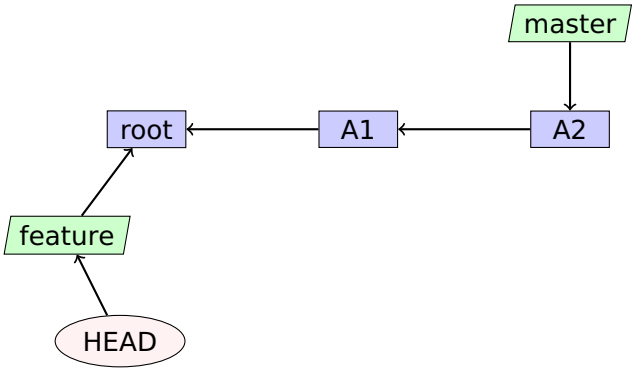
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



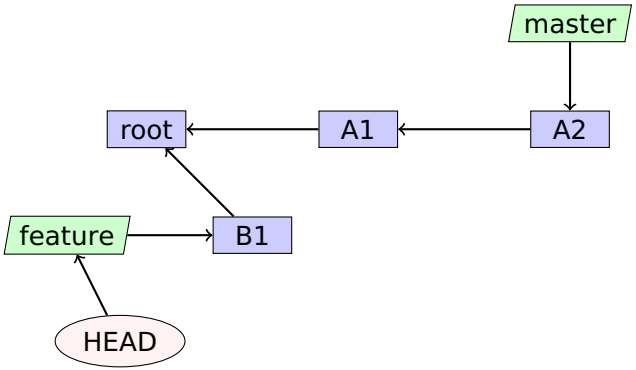
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



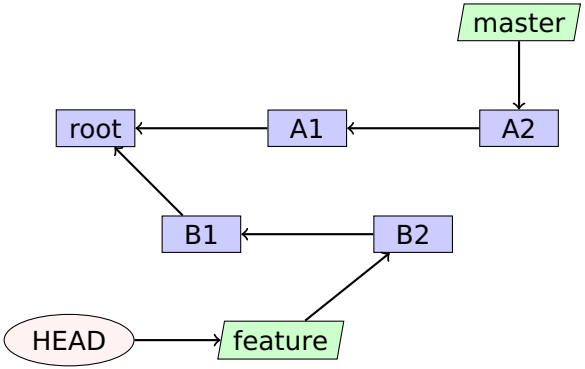
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



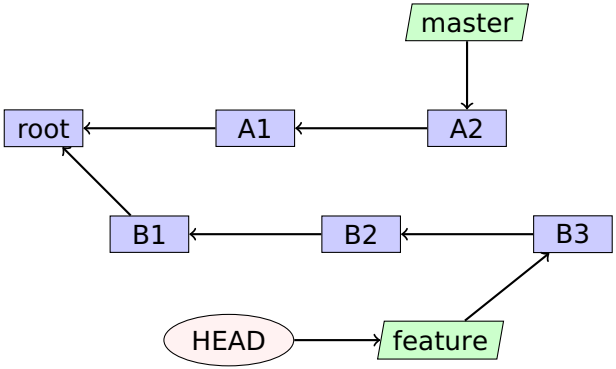
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



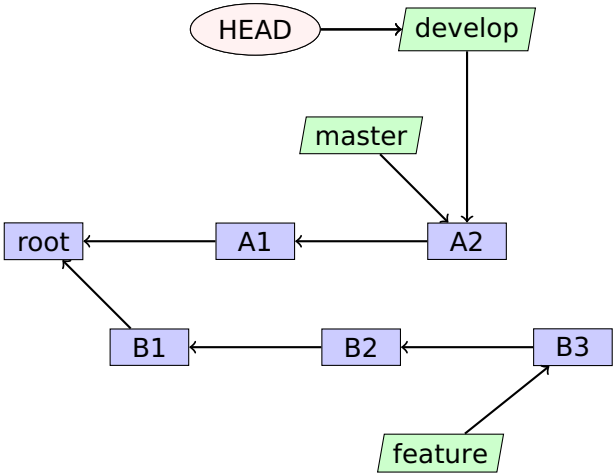
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



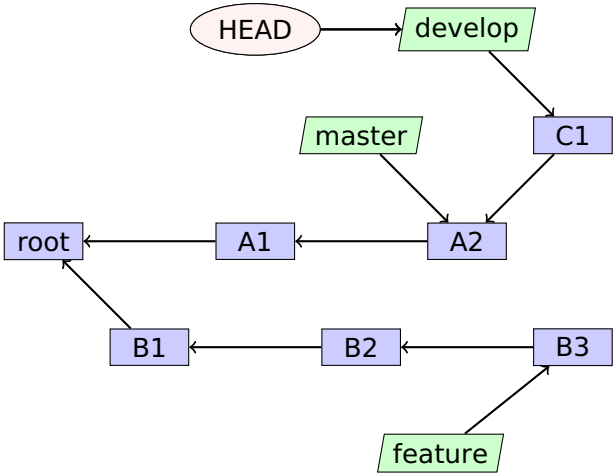
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



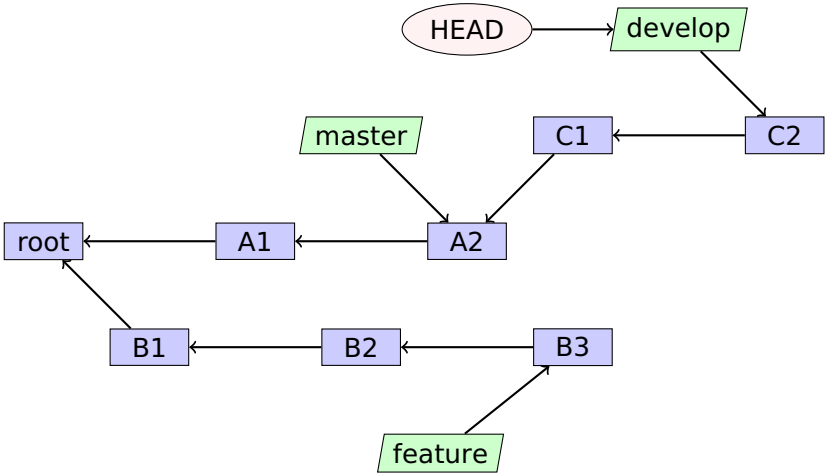
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



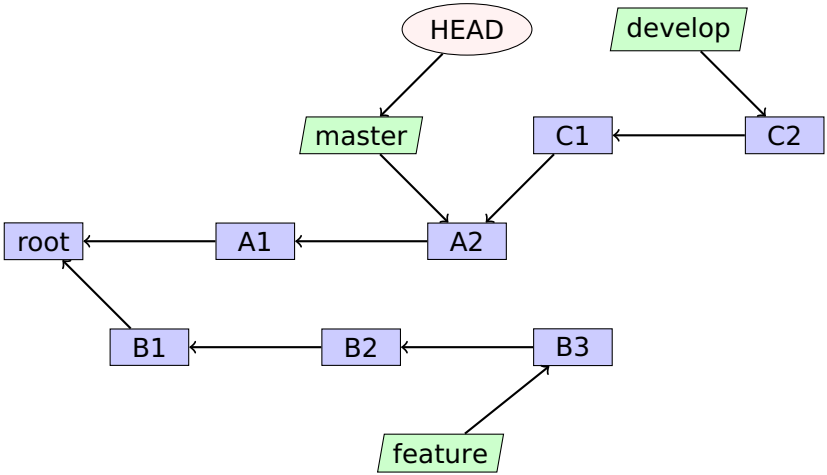
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



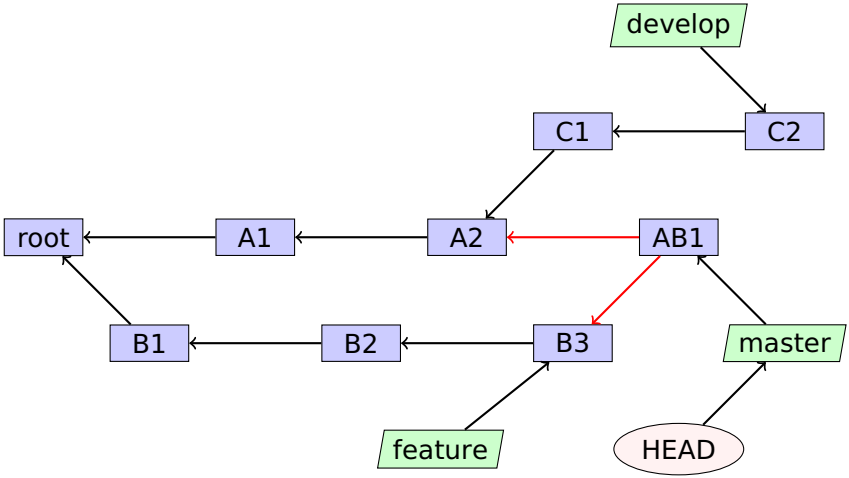
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



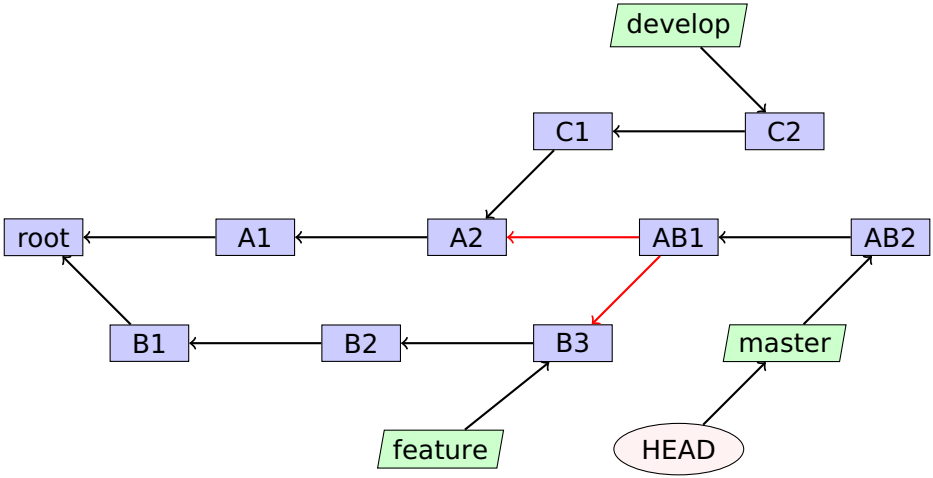
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



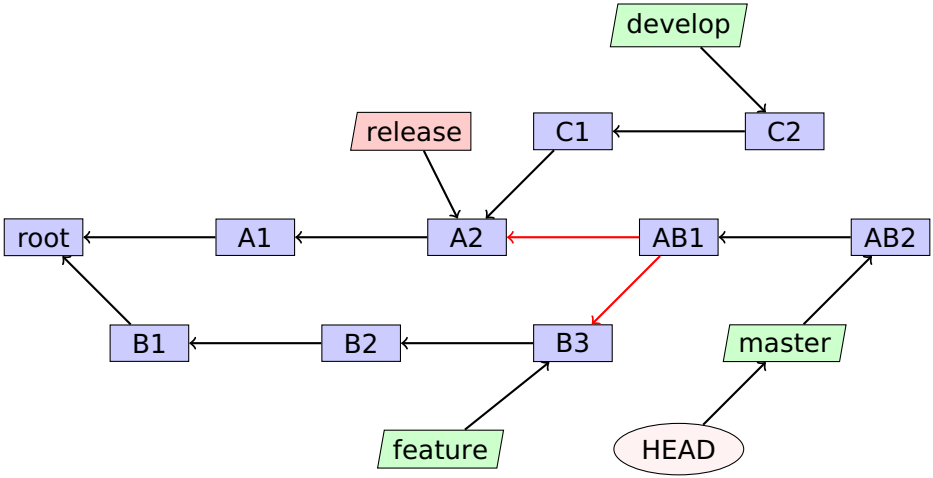
Fundamental Structure: Directed Acyclic Graph (DAG) of commits



Fundamental Structure: Directed Acyclic Graph (DAG) of commits



Fundamental Structure: Directed Acyclic Graph (DAG) of commits



Installing Git

- **Linux distributions**
  Use your package manager of choice, e.g. `apt install git`
- **MacOS**
  Installation is possible via Homebrew: `brew install git`
- **Windows**
  Download and run the Git installer `https://git-scm.com/download/win`

Configuring Git

- First, we make sure that git is installed and ready to use:
  `git --version`
- Each commit contains your name <NAME> and mail adress <EMAIL>, so let's set those:
  `git config --global user.name "<NAME>"`
  `git config --global user.email "<EMAIL>"`
  Omitting the `--global` switch would configure them for your current repo.
- Let's check the global Git configuration:
  `git config --global --list`

Creating and cloning repositories

- We can locally create a new, empty repository with
  `git init`
  The commit objects and other internal Git data will be stored in a hidden subdirectory `.git`.
- Usually we'd like to get a local copy of a remote repo at <URL>, done via
  `git clone <URL>`
- There are many options to show the history leading to the commit <C>, e.g.:
  `git log --decorate --graph --oneline [<C>]`
  - Without specifying any commits, the history to the current commit is shown.
  - There are many GUIs available as well, cf.
    https://git-scm.com/downloads/guis

Creating commits

- The current state of the working directory can be queried as follows:
  `git status`
  This will show changed and new, untracked files.
- Best practice for files that are produced by your build:
  - Include in `.gitignore` file and don't commit them.
  - These files can be bootstrapped for most programming languages at gitignore.io.
  - Examples: `*.o` for a C project or `*.pdf` for LaTeX
- To *stage* all changed files, i.e. mark as part of the next commit:
  `git add .`
  - Replace `.` by a filename (pattern) to be more specific.
  - In general, aim to add related changes, "craft your commits".
- Finally, we can create the new commit with a descriptive message using:
  `git commit -m "<MESSAGE>"`
  - For longer messages, omit the `-m` parameter: The editor specified via

Local and remote branches

- Let's first get an overview of the available branches:
  `git branch`
- With the `-a` switch we get *remote tracking* branches as well.
- A new branch <BRANCH> can be created with
  `git branch <BRANCH>`
- ...and selected as the current one (i.e. representing the working tree) with
  `git switch <BRANCH>`
- In order to merge in the commits from <OTHER_BRANCH> we use
  `git merge <OTHER_BRANCH>`
  - Git is really smart about automatically resolving merge conflicts, at least for text files.
  - If this fails, we have to manually edit the colloding files and then create the merge commit.

Local and remote branches

- Remote repositories can be shown with:
  `git remote`
  - With the `-v` switch each URL is shown as well.
- When cloning a git repo, the source is automatically configured as the remote `origin`.
- We can configure a new remote <REMOTE> at <URL> as follows:
  `git remote add <REMOTE> <URL>`
- The <URL> can be given by
  - a web tool like *GitHub*, *GitLab* or *Gitea* or
  - the path to a *bare* repo (created with `git init --bare` and not containing a working tree).

Local and remote branches

- Showing branches with full verbosity reveals *remote tracking branches*:
  `git branch -vv`
  Again, these are automatically created when cloning from a remote repo.
- In order to update the remote tracking branch:
  `git fetch`
- This can be combined to automatically merge into the local branch:
  `git pull`
- Finally, we can upload locally new commits to the remote branch with:
  `git push`
- Tracking can be set manually, e.g. for <BRANCH> to track <REMOTE> with:
  `git push -u <REMOTE> <BRANCH>`