Git Version Control

Stephan Müller, ISG D-PHYS

HS 2025

Schedule

IT Security

Linux Basics I + II

Git Version Control

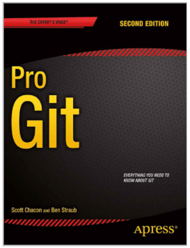
Python Ecosystem I - IV

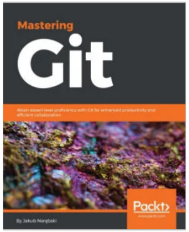
System Aspects

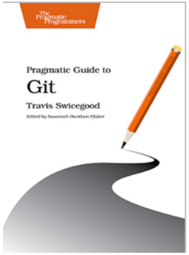
- Checkout compenv.phys.ethz.ch
- Give Feedback!
  - isg@phys.ethz.ch
  - chat.phys.ethz.ch
  - HPT H

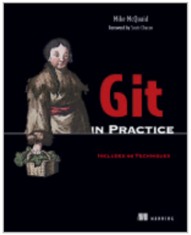
”git gets easier once you get the basic idea that branches are homeomorphic endofunctors mapping submanifolds of a Hilbert space.”

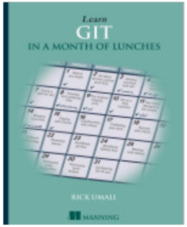
"git gets easier once you get the basic idea that branches are homeomorphic endofunctors mapping submanifolds of a Hilbert space."

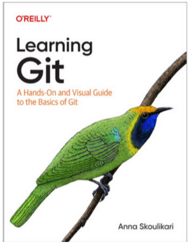


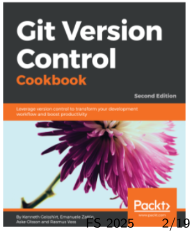










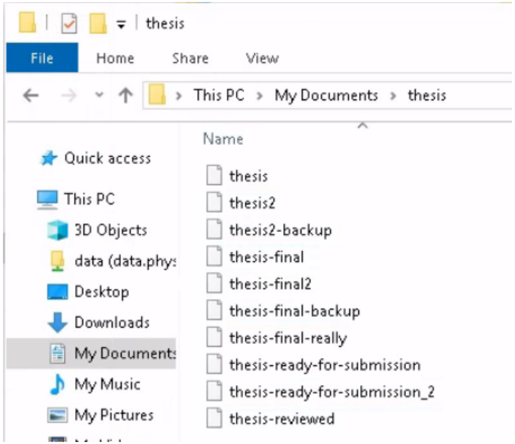


Learning Goals

- Git ≠ GitHub
- Fundamental Concepts
  - Commit
  - HEAD
  - branch
  - Work Dir + Index + Log
- History Concepts
  - Inspect
  - Navigate
  - Modify
- Remotes

THIS IS GIT.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP.

Why use a version control system?



thesis
thesis2
thesis2-backup
thesis-final
thesis-final2
thesis-final-backup
thesis-final-really
thesis-ready-for-submission
thesis-ready-for-submission_2
thesis-reviewed

Does your project look like this?

Benefits

- Tracking changes
- Experimentation
- Collaboration
  - If not now, maybe later
- Easy rollbacks
- ~~Storage efficiency~~
- Simplified backups
- Looks good in your CV

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - "I named it Git – because I am a git."

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - "I named it Git – because I am a git."
- SubVersion (svn)

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - "I named it Git – because I am a git."
- SubVersion (svn)
- Concurent Version System (cvs)

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - "I named it Git – because I am a git."
- SubVersion (svn)
- Concurent Version System (cvs)
- Mercurial (hg)

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - ”I named it Git – because I am a git.”
- SubVersion (svn)
- Concurent Version System (cvs)
- Mercurial (hg)
- Fossil
  - good Git alternative

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - "I named it Git – because I am a git."
- SubVersion (svn)
- Concurent Version System (cvs)
- Mercurial (hg)
- Fossil
  - good Git alternative
- game of trees (got)
  - git compatable, but easier

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - "I named it Git – because I am a git."
- SubVersion (svn)
- Concurent Version System (cvs)
- Mercurial (hg)
- Fossil
  - good Git alternative
- game of trees (got)
  - git compatable, but easier

Definition: Software Forge

A **forge** is a central web-based platform for collaboration hosting extra tools

- Version control
- Bug tracking
- Continious Integration

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - "I named it Git – because I am a git."
- SubVersion (svn)
- Concurent Version System (cvs)
- Mercurial (hg)
- Fossil
  - good Git alternative
- game of trees (got)
  - git compatable, but easier

Definition: Software Forge

A **forge** is a central web-based platform for collaboration hosting extra tools

- Version control
- Bug tracking
- Continious Integration

- GitHub
  - github.com

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - "I named it Git – because I am a git."
- SubVersion (svn)
- Concurent Version System (cvs)
- Mercurial (hg)
- Fossil
  - good Git alternative
- game of trees (got)
  - git compatable, but easier

Definition: Software Forge

A **forge** is a central web-based platform for collaboration hosting extra tools

- Version control
- Bug tracking
- Continious Integration

- GitHub
  - github.com
- GitLab
  - gitlab.phys.ethz.ch
  - gitlab.ethz.ch

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - "I named it Git – because I am a git."
- SubVersion (svn)
- Concurent Version System (cvs)
- Mercurial (hg)
- Fossil
  - good Git alternative
- game of trees (got)
  - git compatable, but easier

Definition: Software Forge

A **forge** is a central web-based platform for collaboration hosting extra tools

- Version control
- Bug tracking
- Continious Integration

- GitHub
  - github.com
- GitLab
  - gitlab.phys.ethz.ch
  - gitlab.ethz.ch
- Gitea
  - Easy self-hosting

Version Control Systems: Git $\neq$ Github

Definition: Version Control System

A **version control system** (VCS) or **source code management system** (SCM) helps to maintain different versions of files.

- Git
  - De facto standard in open source
  - Built in 2005 to manage the Linux kernel source code
  - "I named it Git – because I am a git."
- SubVersion (svn)
- Concurent Version System (cvs)
- Mercurial (hg)
- Fossil
  - good Git alternative
- game of trees (got)
  - git compatable, but easier

Definition: Software Forge

A **forge** is a central web-based platform for collaboration hosting extra tools

- Version control
- Bug tracking
- Continious Integration

- GitHub
  - github.com
- GitLab
  - gitlab.phys.ethz.ch
  - gitlab.ethz.ch
- Gitea
  - Easy self-hosting
- Codeberg.org
- ...

Git Concepts: Commits & HEAD

Git Concepts: Commits & HEAD

- Git tracks files and directories

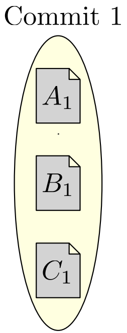
Git Concepts: Commits & HEAD

- Git tracks files and directories

Definition: Commit

A *commit* is a snapshot on **all** tracked files at a given time.

(It is NOT the "diff" between versions)



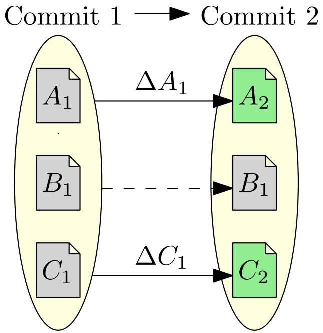
Git Concepts: Commits & HEAD

- Git tracks files and directories
- Git remebers all snapshots
  - Work on files & `git commit`

Definition: Commit

A *commit* is a snapshot on **all** tracked files at a given time.

(It is NOT the "diff" between versions)



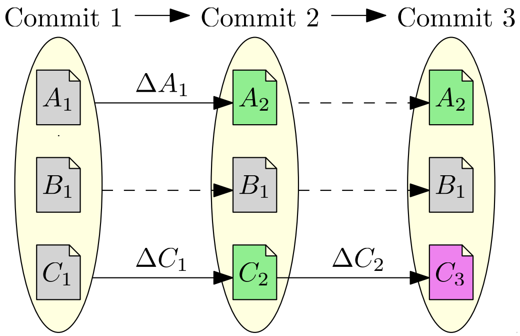
Git Concepts: Commits & HEAD

- Git tracks files and directories
- Git remebers all snapshots
  - Work on files & `git commit`
  - Work on files & `git commit`

Definition: Commit

A *commit* is a snapshot on **all** tracked files at a given time.

(It is NOT the "diff" between versions)



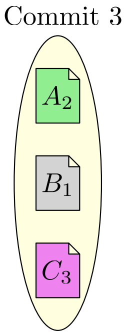
Git Concepts: Commits & HEAD

- Git tracks files and directories
- Git remebers all snapshots
  - Work on files & `git commit`
  - Work on files & `git commit`
- You always have a coherent view on all files

Definition: Commit

A *commit* is a snapshot on **all** tracked files at a given time.

(It is NOT the "diff" between versions)



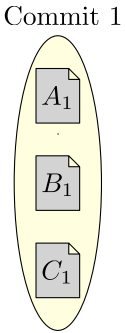
Git Concepts: Commits & HEAD

- Git tracks files and directories
- Git remebers all snapshots
  - Work on files & `git commit`
  - Work on files & `git commit`
- You always have a coherent view on all files
  - You can move back and forth in history
    - `git switch Commit1`*

Definition: Commit

A *commit* is a snapshot on **all** tracked files at a given time.

(It is NOT the "diff" between versions)



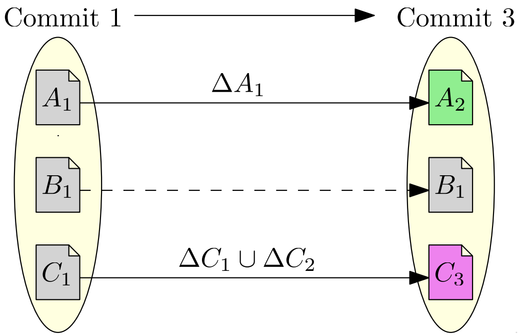
Git Concepts: Commits & HEAD

- Git tracks files and directories
- Git remebers all snapshots
  - Work on files & `git commit`
  - Work on files & `git commit`
- You always have a coherent view on all files
  - You can move back and forth in history
    - `git switch Commit1`*
  - You can compare diffenrent versions
    - `git diff Commit1..Commit3`

Definition: Commit

A *commit* is a snapshot on **all** tracked files at a given time.

(It is NOT the "diff" between versions)



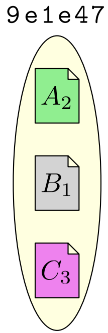
Git Concepts: Commits & HEAD

- Git tracks files and directories
- Git remebers all snapshots
  - Work on files & `git commit`
  - Work on files & `git commit`
- You always have a coherent view on all files
  - You can move back and forth in history
    - `git switch Commit1`*
  - You can compare diffenrent versions
    - `git diff Commit1..Commit3`
- Commits have hash names:
  - `9e1e4739a9ca2e4750011ceaac818c16982e`
  - `9e1e47` - any prefix is OK, if it is unique

Definition: Commit

A *commit* is a snapshot on **all** tracked files at a given time.

(It is NOT the "diff" between versions)



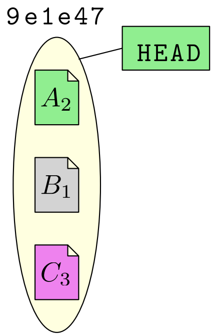
Git Concepts: Commits & HEAD

- Git tracks files and directories
- Git remebers all snapshots
  - Work on files & `git commit`
  - Work on files & `git commit`
- You always have a coherent view on all files
  - You can move back and forth in history
    - `git switch Commit1`*
  - You can compare diffenrent versions
    - `git diff Commit1..Commit3`
- Commits have hash names:
  - `9e1e4739a9ca2e4750011ceaac818c16982e`
  - `9e1e47` - any prefix is OK, if it is unique
- The current commit is refered to as `HEAD`

Definition: Commit

A *commit* is a snapshot on **all** tracked files at a given time.

(It is NOT the "diff" between versions)



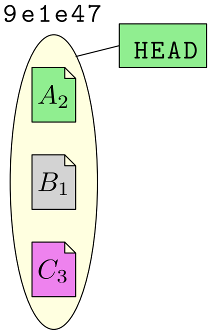
Git Concepts: Commits & HEAD

- Git tracks files and directories
- Git remebers all snapshots
  - Work on files & `git commit`
  - Work on files & `git commit`
- You always have a coherent view on all files
  - You can move back and forth in history
    - `git switch Commit1`*
  - You can compare diffenrent versions
    - `git diff Commit1..Commit3`
- Commits have hash names:
  - `9e1e4739a9ca2e4750011ceaac818c16982e`
  - `9e1e47` - any prefix is OK, if it is unique
- The current commit is refered to as `HEAD`

Definition: Commit

A *commit* is a snapshot on **all** tracked files at a given time.

(It is NOT the "diff" between versions)



Definition: HEAD

`HEAD` is a spacial pointer, referring to the commit* your working tree is currently based on.

Git Concepts: Branches

Git Concepts: Branches

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)

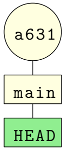
Git Concepts: Branches

- History advances on branches
  - The default branch is called main or master
  - HEAD points to a branch

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



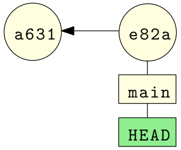
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



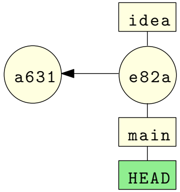
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called ”idea”
  - i.e. create a new pointer

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



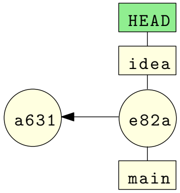
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called "idea"
  - i.e. create a new pointer
- `git switch idea`
  - Make the branch `idea` *active*
  - i.e. move the `HEAD` pointer

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



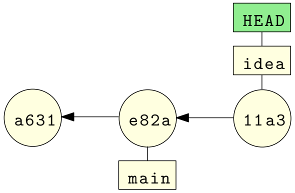
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called "idea"
  - i.e. create a new pointer
- `git switch idea`
  - Make the branch `idea` *active*
  - i.e. move the `HEAD` pointer

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



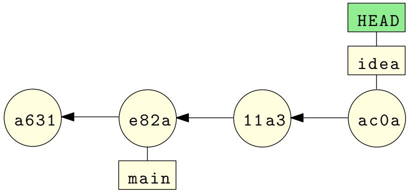
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called "idea"
  - i.e. create a new pointer
- `git switch idea`
  - Make the branch `idea` *active*
  - i.e. move the `HEAD` pointer

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



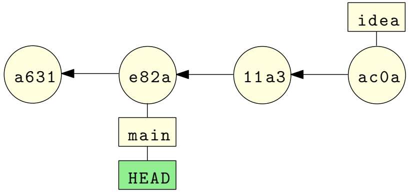
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called "idea"
  - i.e. create a new pointer
- `git switch idea`
  - Make the branch `idea` *active*
  - i.e. move the `HEAD` pointer
- `git switch main`
  - Switch back to branch `main`

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



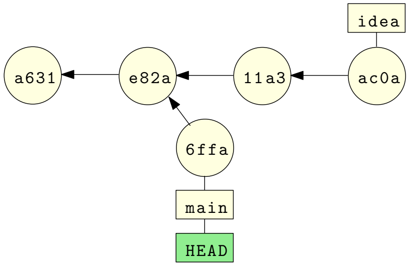
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called "idea"
  - i.e. create a new pointer
- `git switch idea`
  - Make the branch `idea` *active*
  - i.e. move the `HEAD` pointer
- `git switch main`
  - Switch back to branch `main`

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



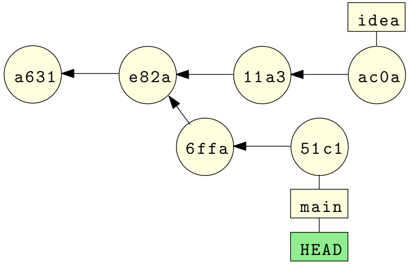
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called "idea"
  - i.e. create a new pointer
- `git switch idea`
  - Make the branch `idea` *active*
  - i.e. move the `HEAD` pointer
- `git switch main`
  - Switch back to branch `main`

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



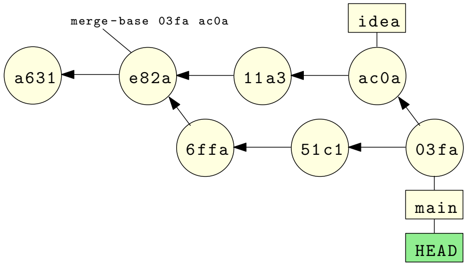
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called "idea"
  - i.e. create a new pointer
- `git switch idea`
  - Make the branch `idea` *active*
  - i.e. move the `HEAD` pointer
- `git switch main`
  - Switch back to branch `main`
- `git merge idea`
  - Apply "new deltas from `idea`" to `main`
  - There may be *merge conflicts*!

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



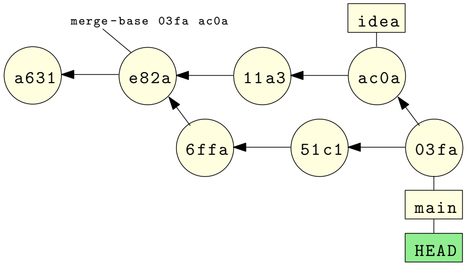
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called "idea"
  - i.e. create a new pointer
- `git switch idea`
  - Make the branch `idea` *active*
  - i.e. move the `HEAD` pointer
- `git switch main`
  - Switch back to branch `main`
- `git merge idea`
  - Apply "new deltas from `idea`" to `main`
  - There may be *merge conflicts*!

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



Detached HEAD

If `HEAD` does not point to a branch, it is called a **detached HEAD**. `git switch` back to a branch!

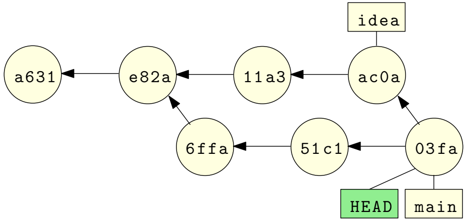
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called "idea"
  - i.e. create a new pointer
- `git switch idea`
  - Make the branch `idea` *active*
  - i.e. move the `HEAD` pointer
- `git switch main`
  - Switch back to branch `main`
- `git merge idea`
  - Apply "new deltas from `idea`" to `main`
  - There may be *merge conflicts*!

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



```
git switch --detach 03fa
```

Detached HEAD

If `HEAD` does not point to a branch, it is called a **detached HEAD**. `git switch` back to a branch!

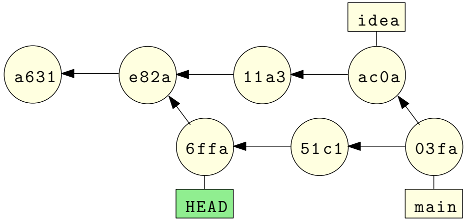
Git Concepts: Branches

- History advances on branches
  - The default branch is called `main` or `master`
  - `HEAD` points to a branch
  - The branch moves with the `HEAD` pointer
- `git branch idea`
  - Create a new branch called "idea"
  - i.e. create a new pointer
- `git switch idea`
  - Make the branch `idea` *active*
  - i.e. move the `HEAD` pointer
- `git switch main`
  - Switch back to branch `main`
- `git merge idea`
  - Apply "new deltas from `idea`" to `main`
  - There may be *merge conflicts*!

Defintion: Branch

A **branch** is a pointer to a *single* commit.

(It is NOT a set or leg of commits)



```
git switch --detach 6ffa
```

Detached HEAD

If `HEAD` does not point to a branch, it is called a **detached HEAD**. `git switch` back to a branch!

Getting Started

Installation

- Installation (git-scm.com/downloads)
  - Linux/Mac: `apt install git-all` / `brew install git`
  - Win: gitforwindows.org
- Command line interface
  - `git [verb] [options] [args]`
    - eg. `git add A.txt`
    - eg. `git log --oneline origin..HEAD`
  - `man git-[verb]`

Getting Started

Installation

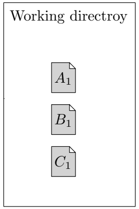
- Installation (git-scm.com/downloads)
  - Linux/Mac: `apt install git-all` / `brew install git`
  - Win: gitforwindows.org
- Command line interface
  - `git [verb] [options] [args]`
    - ▶ eg. `git add A.txt`
    - ▶ eg. `git log --oneline origin..HEAD`
  - `man git-[verb]`

Minimal first time config

- `git config --global user.name "Ann␣Yusar"` (or `$GIT_AUTHOR_NAME`)
- `git config --global user.email "any@ethz.ch"` (or `$GIT_AUTHOR_EMAIL`)
- `git config --global core.editor vim` (or `$GIT_EDITOR`)

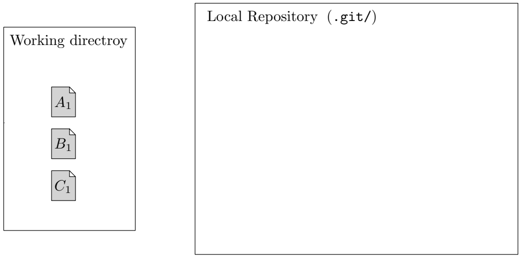
Local repository

- git works on directories



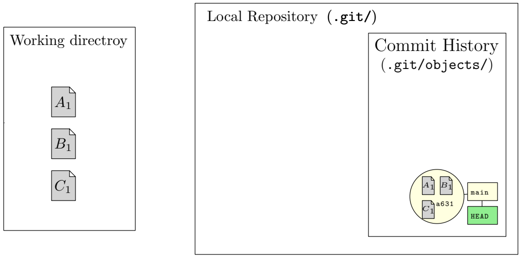
Local repository

- git works on directories
  - `git init` creates .git/



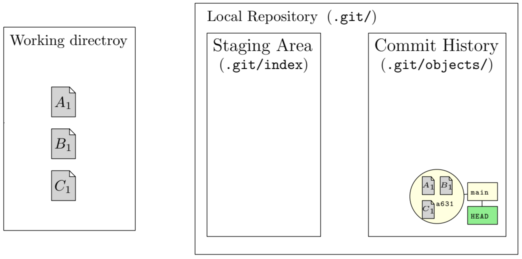
Local repository

- git works on directories
  - `git init` creates .git/
- Commit History
  - files, commits, ... are stored in .git/objects/
  - Objects are indexed by their content



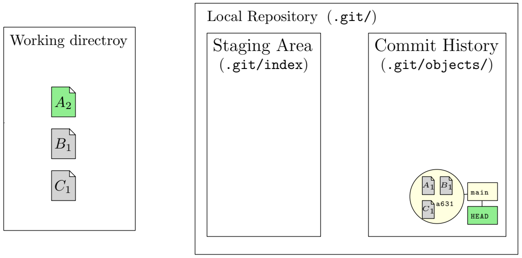
Local repository

- git works on directories
  - `git init` creates .git/
- Commit History
  - files, commits, ... are stored in .git/objects/
  - Objects are indexed by their content
- Staging Area / Index
  - Area to draft a commit



Local repository

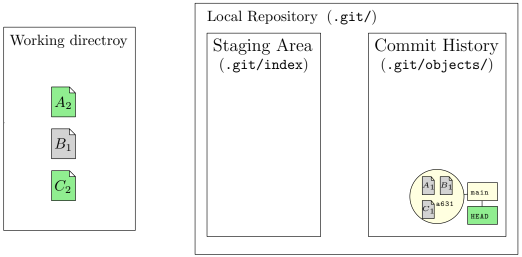
- git works on directories
  - `git init` creates .git/
- Commit History
  - files, commits, … are stored in .git/objects/
  - Objects are indexed by their content
- Staging Area / Index
  - Area to draft a commit



- Change A

Local repository

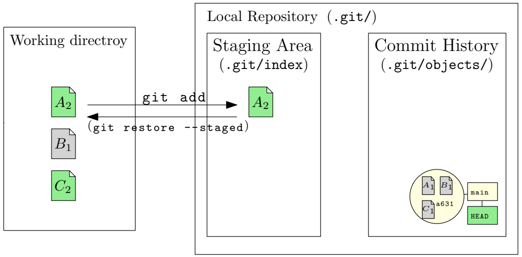
- git works on directories
  - `git init` creates .git/
- Commit History
  - files, commits, ... are stored in .git/objects/
  - Objects are indexed by their content
- Staging Area / Index
  - Area to draft a commit



- Change A
- Change C

Local repository

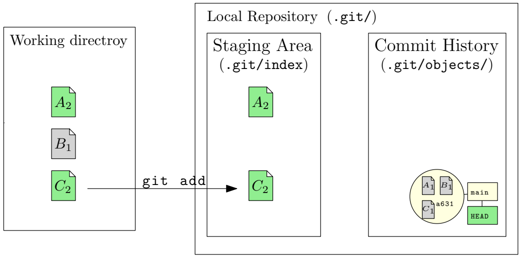
- git works on directories
  - `git init` creates .git/
- Commit History
  - files, commits, ... are stored in .git/objects/
  - Objects are indexed by their content
- Staging Area / Index
  - Area to draft a commit
  - `git add`/`restore --staged`



- Change A
- Change C
- `git add A`

Local repository

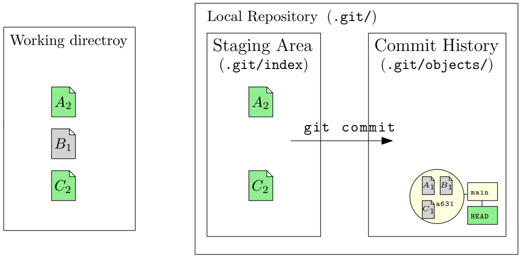
- git works on directories
  - `git init` creates .git/
- Commit History
  - files, commits, ... are stored in .git/objects/
  - Objects are indexed by their content
- Staging Area / Index
  - Area to draft a commit
  - `git add`/`restore --staged`



- Change A
- Change C
- `git add A`
- `git add C`

Local repository

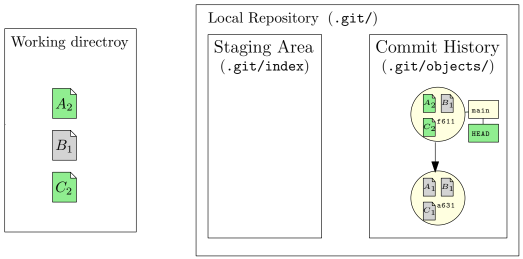
- git works on directories
  - `git init` creates .git/
- Commit History
  - files, commits, ... are stored in .git/objects/
  - Objects are indexed by their content
- Staging Area / Index
  - Area to draft a commit
  - `git add/restore --staged`
- `git commit`
  - Add changes to the history
    - *Author and Date*
    - *Commit message*



- Change A
- Change C
- `git add A`
- `git add C`
- `git commit`

Local repository

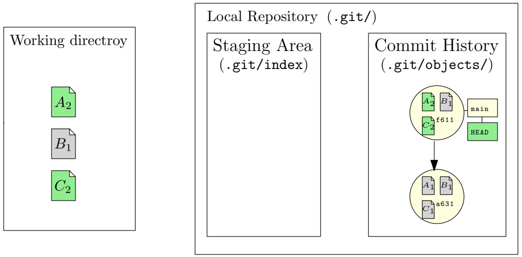
- git works on directories
  - `git init` creates .git/
- Commit History
  - files, commits, ... are stored in .git/objects/
  - Objects are indexed by their content
- Staging Area / Index
  - Area to draft a commit
  - `git add/restore --staged`
- `git commit`
  - Add changes to the history
    - *Author and Date*
    - *Commit message*
  - `HEAD` and branch pointer move
  - Staging area is empty again



- Change A
- Change C
- `git add A`
- `git add C`
- `git commit`

Local repository

- git works on directories
  - `git init` creates .git/
- Commit History
  - files, commits, ... are stored in .git/objects/
  - Objects are indexed by their content
- Staging Area / Index
  - Area to draft a commit
  - `git add`/`restore --staged`
- `git commit`
  - Add changes to the history
    - *Author and Date*
    - *Commit message*
  - `HEAD` and branch pointer move
  - Staging area is empty again

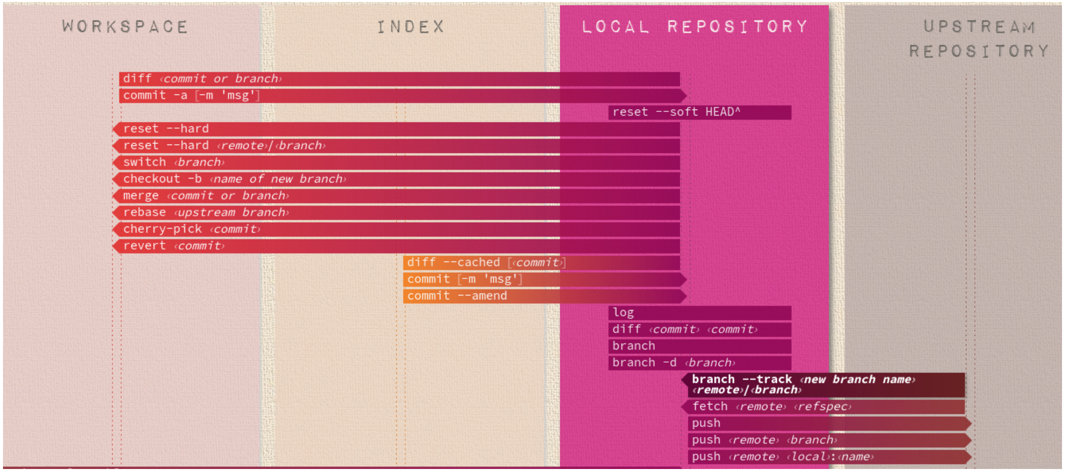


- Change A
- Change C
- `git add A`
- `git add C`
- `git commit`

How to keep the overview?

`git status` is your friend!

A lot of git is about mananging its areas!



Demo

Task

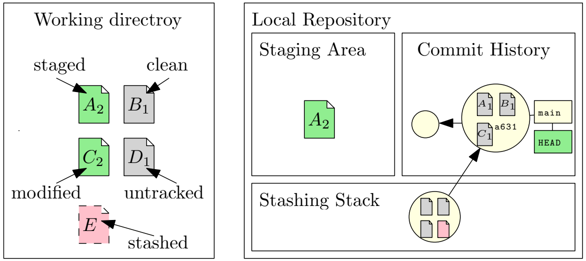
- Write some poems and version control them via git:
  - Create a `poem` folder
  - Initialize a git repository
  - Add some poems
  - Commit your work
- Tools:
  - `git init`
  - `git add`
  - `git commit`
- Check regularly what is going on:
  - `git status`
  - `tree -a`

- `mkdir poems`
- `cd poems`
- `git init`
  - Creates .git/
- (`tree -a`)
- (`fd sample .git -x rm`)
  - Removes example files
- Write some poems
- `git add geometry.txt flowers.txt`
  - Adds files to staging area
- (`git status`)
- `git commit`
  - Adds a new commit

Git States

Staged

In the staging area, not yet commited. Added by a `git add`.



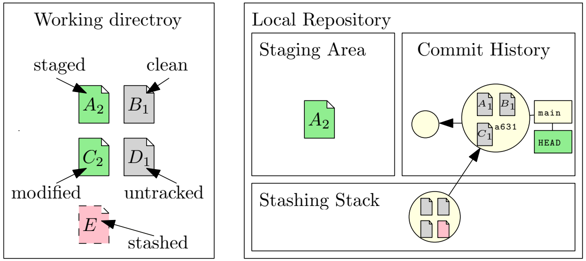
Git States

Staged

In the staging area, not yet commited. Added by a `git add`.

Clean

Tracked, unmodiefied compared to its last version.



Git States

Staged

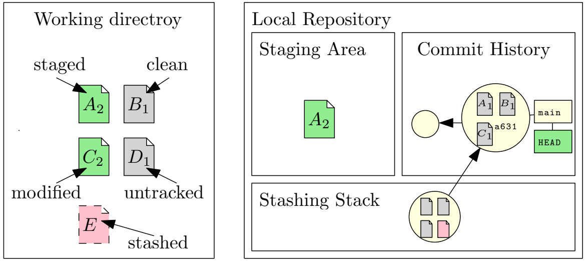
In the staging area, not yet commited. Added by a `git add`.

Clean

Tracked, unmodiefied compared to its last version.

Modified

Tracked and different from its last version.



Git States

Staged

In the staging area, not yet commited. Added by a `git add`.

Clean

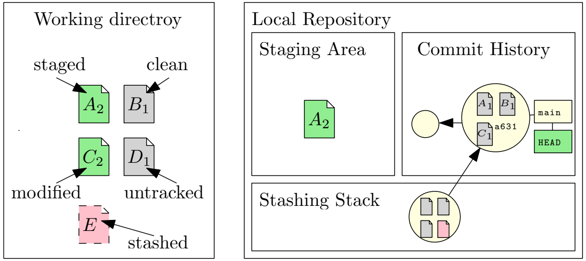
Tracked, unmodiefied compared to its last version.

Modified

Tracked and different from its last version.

Stashed

Clean, because changes are *stashed away* with `git stash`.



Git States

Staged

In the staging area, not yet commited. Added by a `git add`.

Clean

Tracked, unmodiefied compared to its last version.

Modified

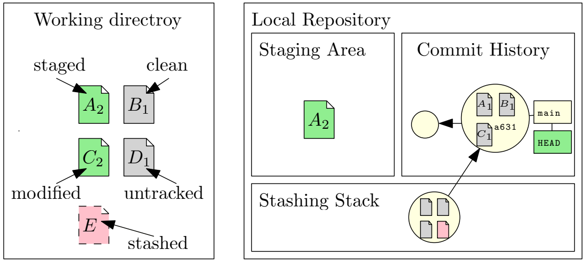
Tracked and different from its last version.

Stashed

Clean, because changes are *stashed away* with `git stash`.

Untracked

Never added, git does not care.



Inspecting History: git log

Inspect the commit log

- `git log [<options>] [<revision-range>] [-- file]`

  `--stat`: Show statistics, how many changes per file
  `--patch` / `-p`: Show differences
  `--graph`: Draw history as graph
  `--max-count/-n` N: Show at most N commits
  `--all`: Add all branches (to rev-range)
  `--oneline`: Only show summary line
  `--` FILE: Show commits which changed FILE

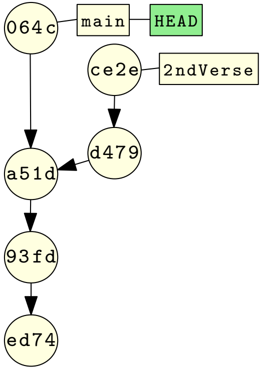
Inspecting History: git log

Inspect the commit log

- `git log [<options>] [<revision-range>] [-- file]`

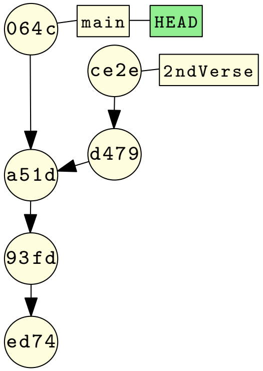
  `--stat`: Show statistics, how many changes per file
  `--patch` / `-p`: Show differences
  `--graph`: Draw history as graph
  `--max-count/-n` N: Show at most N commits
  `--all`: Add all branches (to rev-range)
  `--oneline`: Only show summary line
  `--` FILE: Show commits which changed FILE

```
  git log --oneline --all --graph

* 064c6f1 (HEAD -> main) Fix flower poem
| * ce2e2f0 (2ndVerse) Improve math
| * d479f9f Add 1st draft
|/
* a51df54 Make it rhyme
* 93fde7d Complete flower poem
* ed74ade Add poems
```



Inspecting History: git log

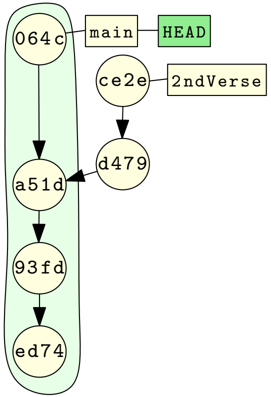
```
git log --oneline d479..064c
git log --oneline ^d479 064c

064c6f1 (HEAD -> main) Fix flower poem
```



Inspecting History: git log

```
git log --oneline d479..064c
git log --oneline ^d479 064c

064c6f1 (HEAD -> main) Fix flower poem
```

- Show everything reachable from 064c

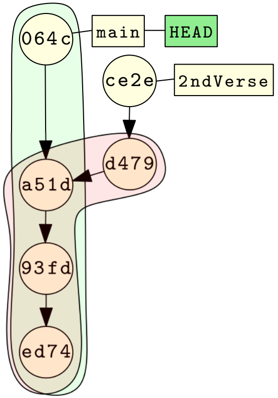


Inspecting History: git log

```
git log --oneline d479..064c
git log --oneline ^d479 064c
```

```
064c6f1 (HEAD -> main) Fix flower poem
```

- Show everything reachable from `064c`
- Exclude everything reachable from `d479`



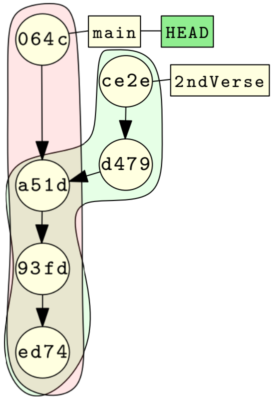
Inspecting History: git log

```
git log --oneline d479..064c
git log --oneline ^d479 064c

064c6f1 (HEAD -> main) Fix flower poem
```

- Show everything reachable from `064c`
- Exclude everything reachable from `d479`

```
git log --oneline main..2ndVerse

ce2e2f0 (2ndVerse) Improve math
d479f9f Add 1st draft
```

- What is on `2ndVerse` but not on `main`?

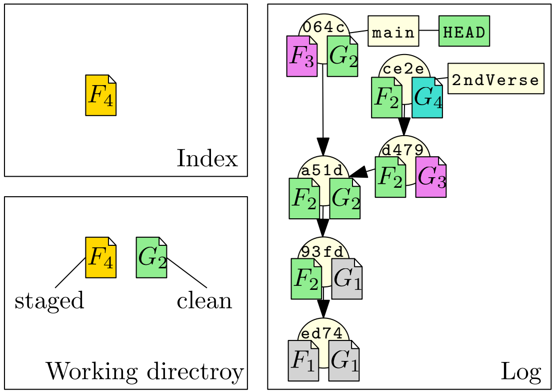


Navigating History

Moving HEAD

`git reset [<mode>] 93fd`

Move `HEAD` and its branch to a commit.



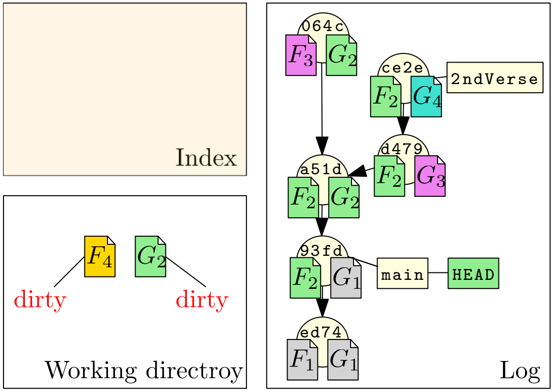
Navigating History

Moving HEAD

`git reset [<mode>] 93fd`

Move `HEAD` and its branch to a commit.

`--mixed`: Update Index (Default)



Navigating History

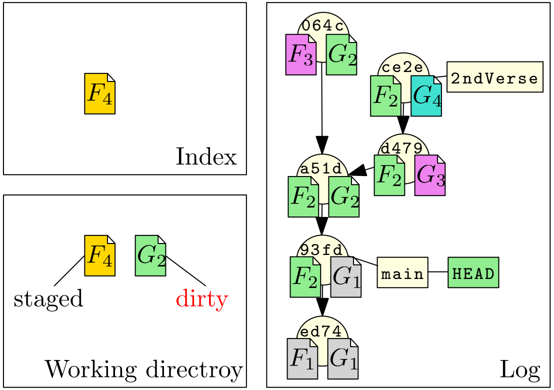
Moving HEAD

`git reset [<mode>] 93fd`

Move `HEAD` and its branch to a commit.

`--mixed`: Update Index (Default)

`--soft`: Update nothing



Navigating History

Moving HEAD

`git reset [<mode>] 93fd`

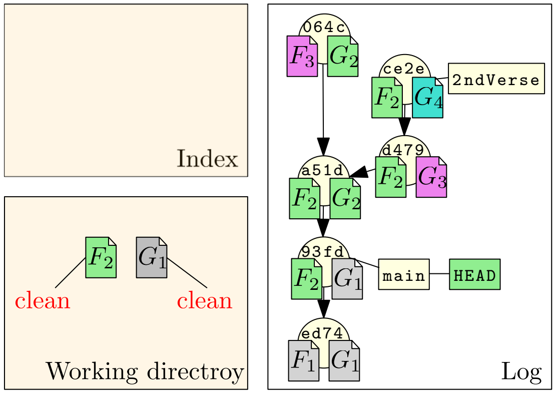
Move `HEAD` and its branch to a commit.

`--mixed`: Update Index (Default)

`--soft`: Update nothing

`--hard` Update Index + Workdir

- You will lose changes!



Navigating History

Moving HEAD

`git reset [<mode>] 93fd`

Move `HEAD` and its branch to a commit.

`--mixed`: Update Index (Default)

`--soft`: Update nothing

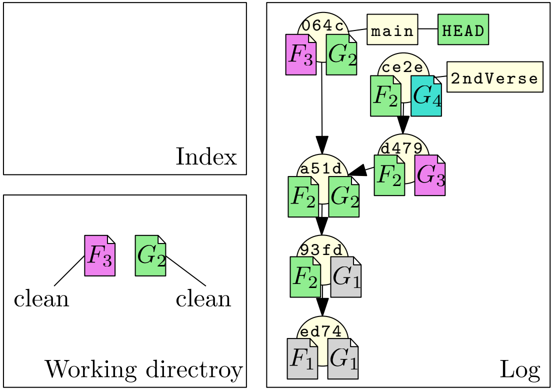
`--hard` Update Index + Workdir

– You will lose changes!

Switching branches

`git switch <branch>`

- Supersedes `git checkout`



Navigating History

Moving HEAD

`git reset [<mode>] 93fd`

Move `HEAD` and its branch to a commit.

`--mixed`: Update Index (Default)

`--soft`: Update nothing

`--hard` Update Index + Workdir

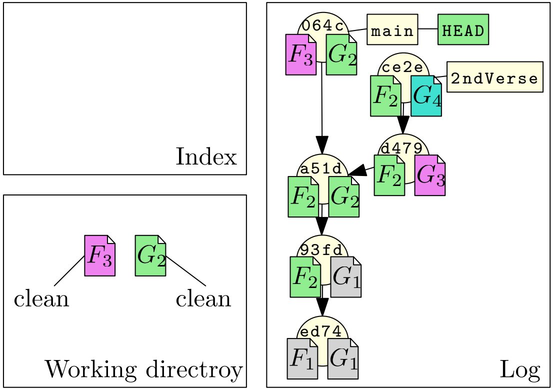
- You will lose changes!

Switching branches

`git switch <branch>`

- Supersedes `git checkout`

```
git switch 2ndVerse
```



Navigating History

Moving HEAD

`git reset [<mode>] 93fd`

Move `HEAD` and its branch to a commit.

`--mixed`: Update Index (Default)

`--soft`: Update nothing

`--hard` Update Index + Workdir

– You will lose changes!

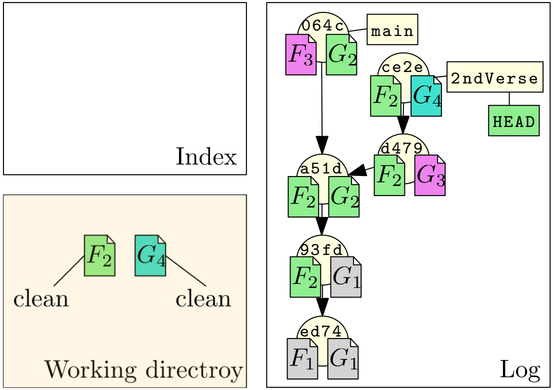
Switching branches

`git switch <branch>`

- Supersedes `git checkout`

`git switch 2ndVerse`

- Move `HEAD` to `2ndVerse`
- Update Working Dir



Rebasing: Rewriting History

Re-apply commits to a new base

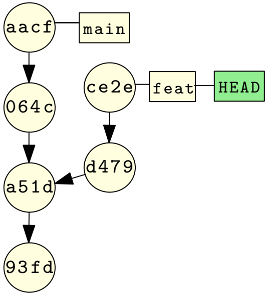
`git rebase [<options>] <upstream>`

- Re-apply differences with `upstream` on top of `upstream`

  `--interactive`/`-i`: Manually apply changes

```
git rebase main
```

- Apply changes not in `main` on top of `main`



Rebasing: Rewriting History

Re-apply commits to a new base

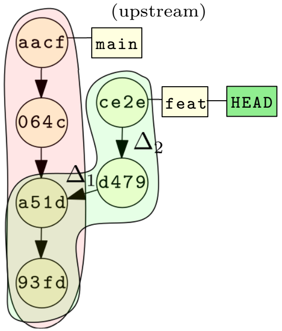
`git rebase [<options>] <upstream>`

- Re-apply differences with `upstream` on top of `upstream`

  `--interactive`/`-i`: Manually apply changes

`git rebase main`

- Apply changes not in `main` on top of `main`
  - Identify commits: `git log upstream..HEAD`



Rebasing: Rewriting History

Re-apply commits to a new base

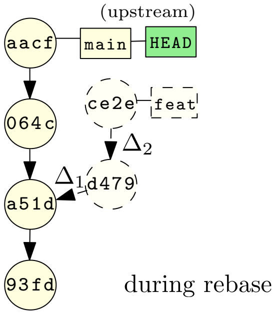
`git rebase [<options>] <upstream>`

- Re-apply differences with `upstream` on top of `upstream`

  `--interactive/-i`: Manually apply changes

`git rebase main`

- Apply changes not in `main` on top of `main`
  - Identify commits: `git log upstream..HEAD`
  - (Hard reset) Workdir to `upstream`
    and calculate patches Δ_1, Δ_2 and remove commits



Rebasing: Rewriting History

Re-apply commits to a new base

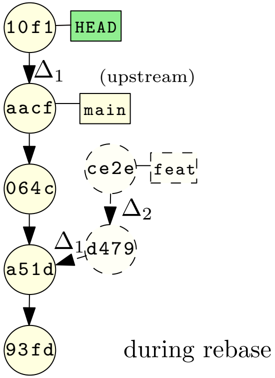
`git rebase [<options>] <upstream>`

- Re-apply differences with `upstream` on top of `upstream`

  `--interactive`/`-i`: Manually apply changes

```
git rebase main
```

- Apply changes not in `main` on top of `main`
  - Identify commits: `git log upstream..HEAD`
  - (Hard reset) Workdir to `upstream` and calculate patches Δ_1, Δ_2 and remove commits
  - Apply patches, one-by-one



Rebasing: Rewriting History

Re-apply commits to a new base

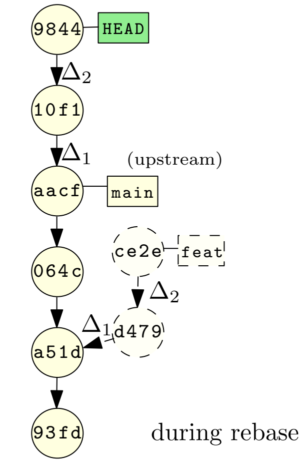
`git rebase [<options>] <upstream>`

- Re-apply differences with `upstream` on top of `upstream`

  `--interactive`/`-i`: Manually apply changes

`git rebase main`

- Apply changes not in `main` on top of `main`
  - Identify commits: `git log upstream..HEAD`
  - (Hard reset) Workdir to `upstream` and calculate patches Δ_1, Δ_2 and remove commits
  - Apply patches, one-by-one



Rebasing: Rewriting History

Re-apply commits to a new base

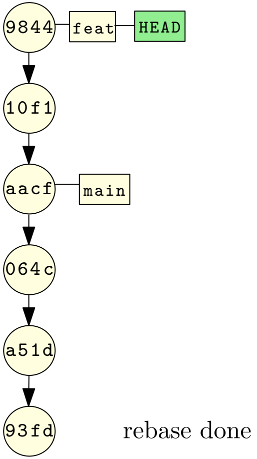
`git rebase [<options>] <upstream>`

- Re-apply differences with `upstream` on top of `upstream`

  `--interactive`/`-i`: Manually apply changes

`git rebase main`

- Apply changes not in `main` on top of `main`
  - Identify commits: `git log upstream..HEAD`
  - (Hard reset) Workdir to `upstream` and calculate patches Δ_1, Δ_2 and remove commits
  - Apply patches, one-by-one
  - Move branches



Rebasing: Rewriting History

Re-apply commits to a new base

`git rebase [<options>] <upstream>`

- Re-apply differences with `upstream` on top of `upstream`

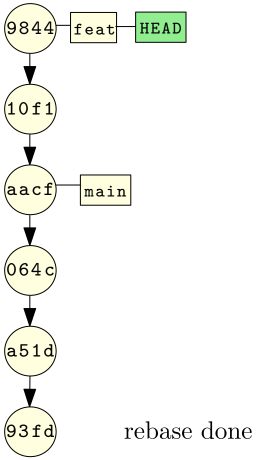
  `--interactive`/`-i`: Manually apply changes

`git rebase main`

- Apply changes not in `main` on top of `main`
  - Identify commits: `git log upstream..HEAD`
  - (Hard reset) Workdir to `upstream` and calculate patches Δ_1, Δ_2 and remove commits
  - Apply patches, one-by-one
  - Move branches

Benefits of rebase

- Avoids merges, keeps the history simple.
- Rework commits (or their massages)
- NEVER REBASE SHARED COMMITS!



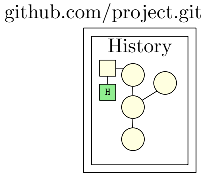
Remote Repositories

Remote repository

A **remote** is a *pointer* to an other copy of the repository, usually on an other machine or server.

- Remotes are managed with `git remote`
- Inter-repo commands are `git clone`/`fetch`/`push`/`pull`
- Remotes in a forge are often *bare repositories*, i.e. have no working dir and no index

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



Remote Repositories

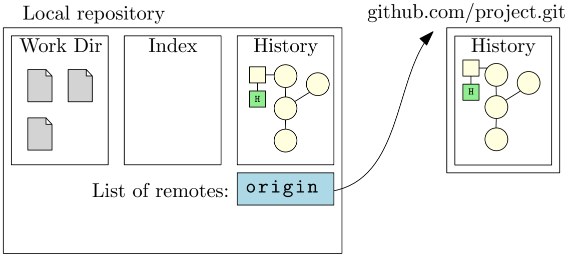
Remote repository

A **remote** is a *pointer* to an other copy of the repository, usually on an other machine or server.

- Remotes are managed with `git remote`
- Inter-repo commands are `git clone`/`fetch`/`push`/`pull`
- Remotes in a forge are often *bare repositories*, i.e. have no working dir and no index

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

- Create a local copy of a repository (including working dir and index)
- Create a remote called `origin`
- Create a "remote-tracking branch"



Remote Repositories

Remote repository

A **remote** is a *pointer* to an other copy of the repository, usually on an other machine or server.

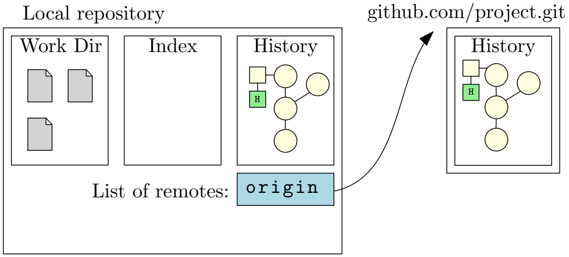
- Remotes are managed with `git remote`
- Inter-repo commands are `git clone`/`fetch`/`push`/`pull`
- Remotes in a forge are often *bare repositories*, i.e. have no working dir and no index

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

- Create a local copy of a repository (including working dir and index)
- Create a remote called `origin`
- Create a "remote-tracking branch"

`git remote add testing <URL>`

- Add another remote



Remote Repositories

Remote repository

A **remote** is a *pointer* to an other copy of the repository, usually on an other machine or server.

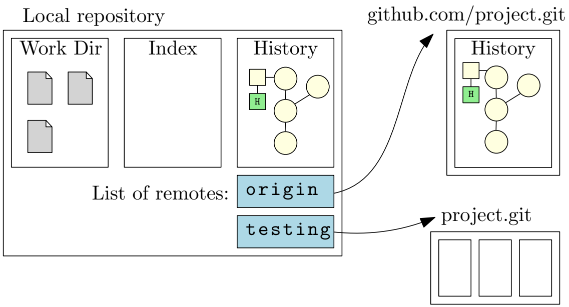
- Remotes are managed with `git remote`
- Inter-repo commands are `git clone`/`fetch`/`push`/`pull`
- Remotes in a forge are often *bare repositories*, i.e. have no working dir and no index

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

- Create a local copy of a repository (including working dir and index)
- Create a remote called `origin`
- Create a "remote-tracking branch"

`git remote add testing <URL>`

- Add another remote

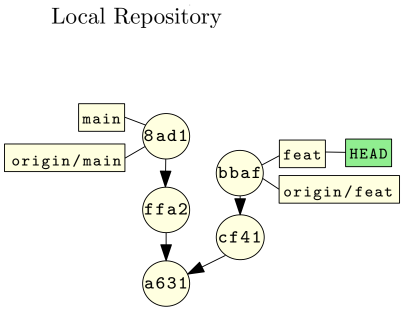


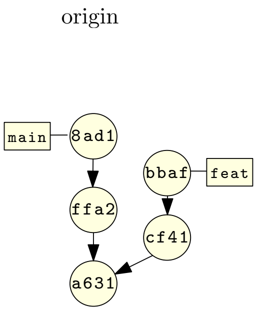
Remote Repositories

Definition: Remote-tracking branch

A **remote-tracking branch** is a *local* branch referencing the state of a branch on a remote repository (called *upstream branch*).

- e.g. `origin/main` is a local branch reffering to the `main` branch in the `origin` repository
- Can not be moved manually





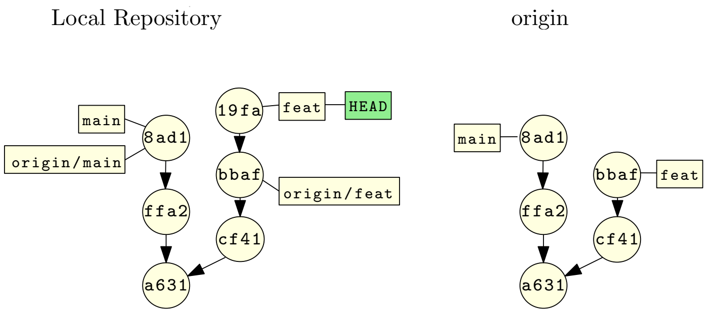
Remote Repositories

Definition: Remote-tracking branch

A **remote-tracking branch** is a *local* branch referencing the state of a branch on a remote repository (called *upstream branch*).

- e.g. `origin/main` is a local branch reffering to the `main` branch in the `origin` repository
- Can not be moved manually

- new local commit
  - `origin/feat` stays



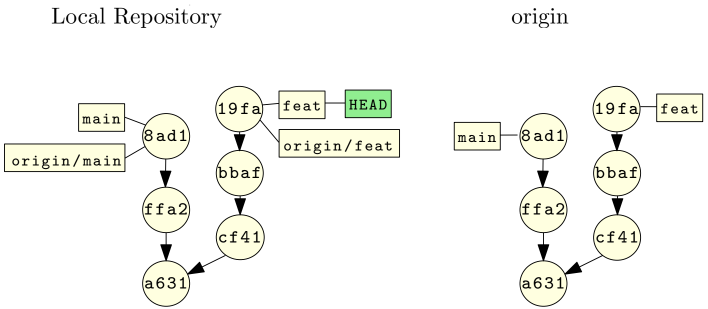
Remote Repositories

Definition: Remote-tracking branch

A **remote-tracking branch** is a *local* branch referencing the state of a branch on a remote repository (called *upstream branch*).

- e.g. `origin/main` is a local branch reffering to the `main` branch in the `origin` repository
- Can not be moved manually

- new local commit
  - `origin/feat` stays
- `git push`
  - send new commits



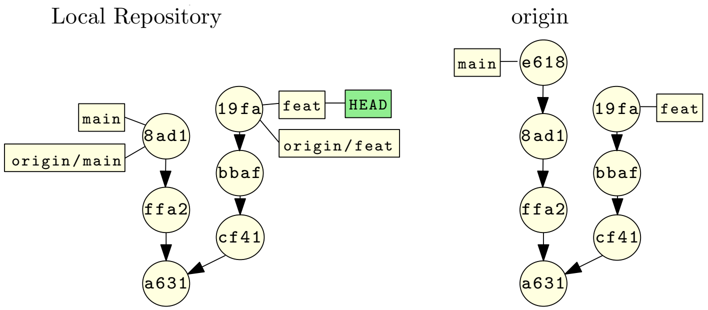
Remote Repositories

Definition: Remote-tracking branch

A **remote-tracking branch** is a *local* branch referencing the state of a branch on a remote repository (called *upstream branch*).

- e.g. `origin/main` is a local branch reffering to the `main` branch in the `origin` repository
- Can not be moved manually

- new local commit
  - `origin/feat` stays
- `git push`
  - send new commits
- new remote commit
  - nothing changes



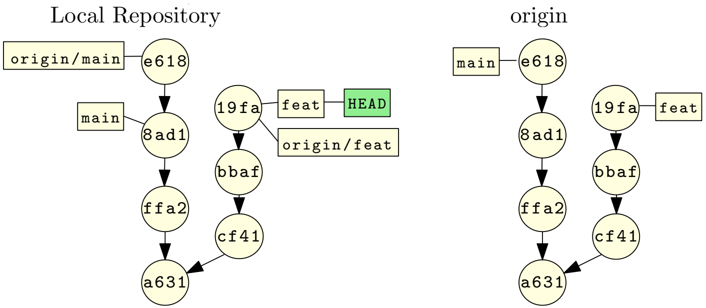
Remote Repositories

Definition: Remote-tracking branch

A **remote-tracking branch** is a *local* branch referencing the state of a branch on a remote repository (called *upstream branch*).

- e.g. `origin/main` is a local branch reffering to the `main` branch in the `origin` repository
- Can not be moved manually

- new local commit
  - `origin/feat` stays
- `git push`
  - send new commits
- new remote commit
  - nothing changes
- `git fetch`
  - get new commits
  - move remote-tracking branches



Rules of thumb

- `git status` a lot!
- Separate changes: Commit (many) small logical steps
  - Write meaningful commit messages
- Use a long running branches (`main`) + small topic branches (`idea`)
- Prefer `git rebase` over `git merge`
- Prefer `git fetch` over `git pull`
- Keep it simple!
- Avoid evil merges:
  - Do not introduce new changes while resolving conflicts
- Never modified published commits
- Do not store large data within git
- Never commit secrets

Where to go next

- Official Docs (git-scm.com/doc)
  - git-scm.com/docs/gittutorial (basic usage)
  - git-scm.com/docs/gittutorial-2 (internals)
  - git-scm.com/docs/giteveryday (most common commands)
  - git-scm.com/docs/gitglossary (all git terms)
- Cheat Sheets
  - ndpsoftware.com/git-cheatsheet.html
  - store.git-init.com
- Games
  - ohmygit.org
  - learngitbranching.js.org
- Other
  - think-like-a-git.net
  - stevelosh.com/blog/2013/04/git-koans
  - xkcd.com/1597/