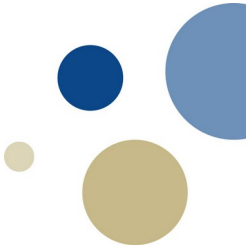
NTNU

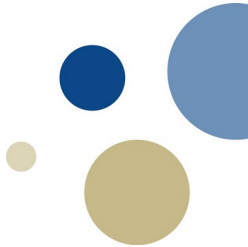
Norwegian University of
Science and Technology



git

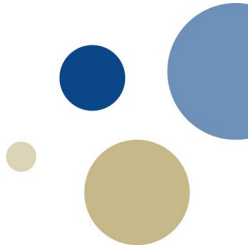
”the information manager from hell”

git is ...



a distributed (open source) version control system that tracks changes in a set of computer files, ...

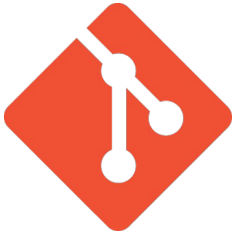
git is a ...



"Global information tracker":
you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.

"Goddamn idiotic truckload of sh*t":
when it breaks.

(github.com/git)



git:

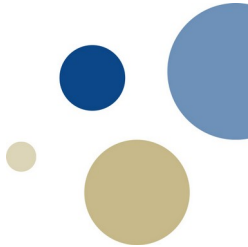
Version control system

github &

gitlab:

Providers for git-based servers to synchronise and backup git projects
+ Webfrontend

Why use git?



Single User:

- Identify changes to files
- Preserve previous versions / no negative progress
- Keep track of changes, understand why you made them

Multiple Users:

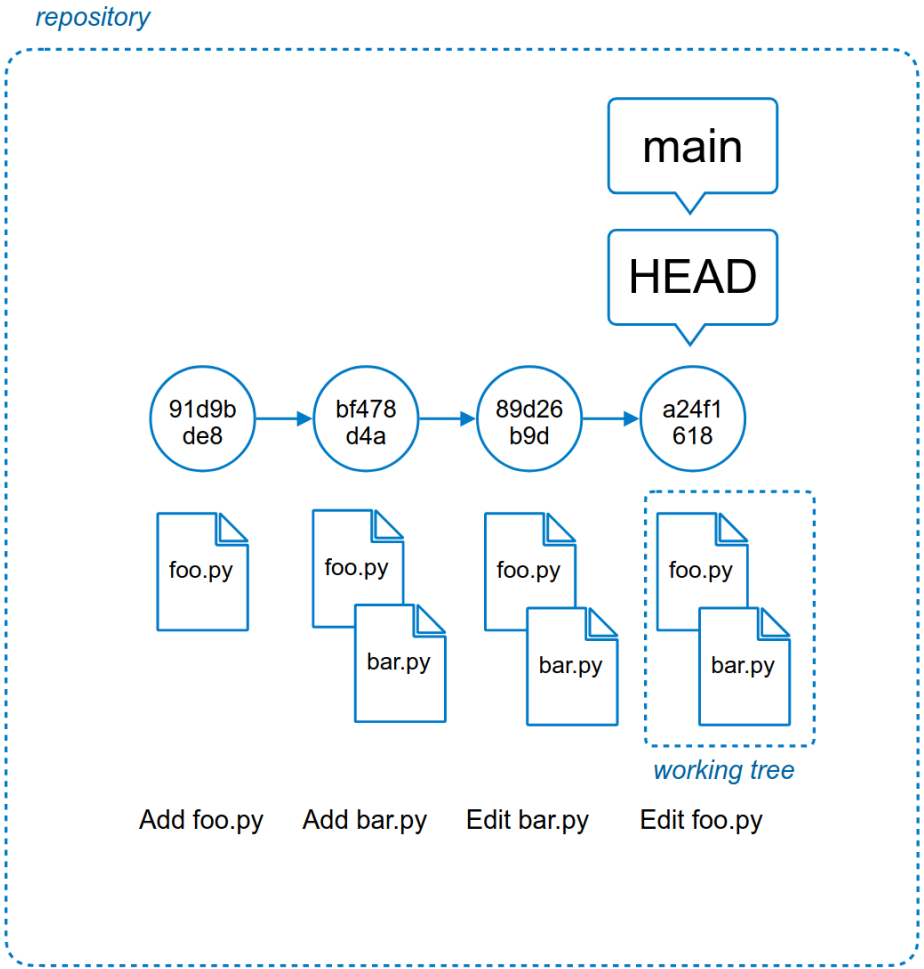
- Compare files
- Collaborate on a (coding) project

An easy start – Single-User Git

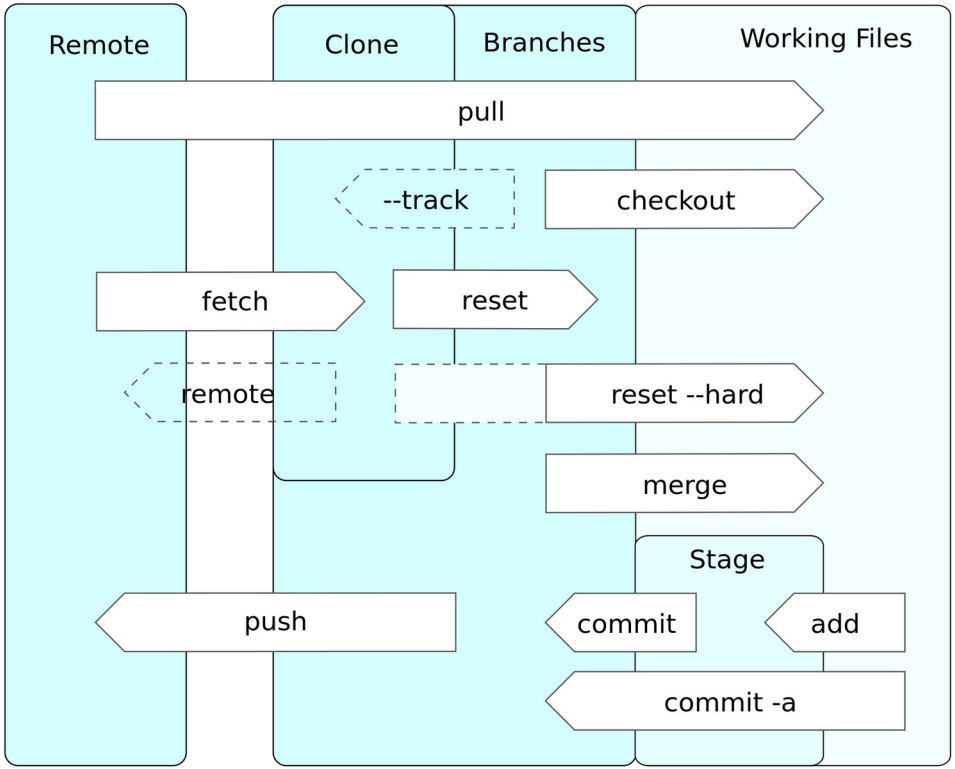
Terminology:

3 pillars of git

- **Repository:** set of files and their histories managed by git
- **Working tree / directory:** list of all current files (tracked and untracked)
- **Commit:** full copy of the tracked files at some particular time, including a timestamp, log message and link to ≥1 parent commits. Identified by a unique 160-bit SHA1 hash.
- **Index:** staging area for changes before they are committed. Sits between working directory and immutable object database of all previous commits.
- **HEAD:** reference for the currently checked-out commit.
- **Checkout:** replace working tree with files from a particular commit.
- **Branch:** named pointer to a particular commit. Making a commit automatically advances the active branch.


repository
main
HEAD
91d9b
de8
bf478
d4a
89d26
b9d
a24f1
618
foo.py
foo.py
bar.py
foo.py
bar.py
foo.py
bar.py
working tree
Add foo.py
Add bar.py
Edit bar.py
Edit foo.py

Typical Workflow



en.wikipedia.org/wiki/Git

Typical Workflow

1. Figure our what has changed since the last commit

2. Stage some / all of the changes

3. Commit changes, including a short note

(4. Push changes to remote)

Step 0: Starting a git repo

git init <project directory>

Initialise a new git repository (folder).

git clone <repo address>

“Copy” an existing repo from a remote server.

```
git clone https://git.overleaf.com/62ea85525a816e7158cf8049
```

Step 1: What happened here?

git status

Let’s you know:

- Currently active branch
- Connection to remote-tracking branches
- Unstaged, modified files
- Untracked files in working tree

```
❯ git status
On branch ntnu
Your branch is up to date with 'ntnu_gitlab/ntnu'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be
committed)
  (use "git restore <file>..." to discard changes
in working directory)
        modified:   analysis_basics.py
        modified:   compare_spectrum.py
        modified:   snippets.ipynb

Untracked files:
  (use "git add <file>..." to include in what will
 be committed)
        .gitignore
        Makefile
        adaptive-example.ipynb
        chain.h5
        helloworld.c
        helloworld.cpython-39-x86_64-linux-gnu.so
        helloworld.pyx
        helper_scripts_sanitised.ipynb
        hidden_params.mp4
        lib_spectrum.so
        open_issues.md
        particles_to_spectrum.cpp
        range_rescaling.nb
        submit_scan.sh
        test.cpp
        uhecr_scan.sh

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

Tip:

Modify your .gitignore file to hide irrelevant files from git status.

```
❯ vim .gitignore
*.gz
*.py
*.zip
*.csv
*.png
*.txt
*.svg
*.pdf
*.npy
*.npz
*/*
rust_components/
```

Step 1: What happened here?

git diff <hash1> <hash2>

Shows the difference between two commits.

git diff <filename>

Shows the changes made to a file since HEAD.

git diff --stat

Shows number of changed lines per file.

Pick a good text editor!

```
diff --git a/compare_spectrum.py b/compare_spectrum.py
index b89720b..cbf2aee 100644
--- a/compare_spectrum.py
+++ b/compare_spectrum.py
@@ -49,7 +49,7 @@ def main(exec='script', x0=[0,0,0,0], ipc=0, config='m'):
     zmin        = 0.001     # minimum source distance for analysis
     metal_ratio = 1         # metallicity rescaling factor
     param       = 'Heitler' # parametrisation scheme for hadronic showers
-    model       = 'EPOS-LHC' # model for hadronic interaction in air showers
+    model       = 'Sibyll23' # model for hadronic interaction in air showers
     phi_src     = 'exp'     # high-R spectral cutoff at the sources
     dpdRmax     = 'delta'   # shape of Rmax distribution
     IRFlag      = 4     # IR Model for em propagation
```

w/o arguments: returns the difference between HEAD and the working tree.

Step 2: Stage modified files

git add <filename>

Stages a single file for inclusion in the next commit.

git add -u

Stages ALL MODIFIED FILES for inclusion in the next commit.

Note: Additional changes to a staged file must be staged again.

```
❯ git status
On branch ntnu
Your branch is up to date with 'ntnu_gitlab/ntnu'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   compare_spectrum.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committ
ed)
  (use "git restore <file>..." to discard changes in work
ing directory)
        modified:   analysis_basics.py
        modified:   snippets.ipynb

Untracked files:
  (use "git add <file>..." to include in what will be com
mitted)
        .gitignore
        Makefile
        adaptive-example.ipynb
        chain.h5
        helloworld.c
        helloworld.cpython-39-x86_64-linux-gnu.so
        helloworld.pyx
        helper_scripts_sanitised.ipynb
        hidden_params.mp4
        lib_spectrum.so
        open_issues.md
        particles_to_spectrum.cpp
        range_rescaling.nb
        submit_scan.sh
        test.cpp
        uhecr_scan.sh
```

Step 3: Create a new commit

git commit -m ‘<Commit Message>’

Move changes from index (staging area or “cache”) to the currently active local branch. Advances branch pointer and HEAD to latest commit.

git commit --amend

Apply additional, newly-staged changes to the previous commit.

Use only for minor modifications!
Avoid amending pushed/public commits!

Aside: View the history

git log --graph

See commit history, including hash, branches (local+remote), Author + Email, timestamp and commit message.

git show <hash>

Show all changes made in that particular commit.

```
* commit 82bea07 (HEAD -> ntnu, origin/ntnu, ntnu_gitlab/ntnu)
| Author: Domenik Ehlert <domenik.ehlert@desy.de>
| Date:   Mon Apr 3 16:29:02 2023 +0200
|
|     Updated calculation of multimessenger chi2 penalty. Only use penalty
 from an upper-limit point if N_pred > N_obs.
|
* commit 413c163
| Author: Domenik Ehlert <domenik.ehlert@desy.de>
| Date:   Mon Mar 20 17:22:49 2023 +0100
|
|     Switched lnA-calculations to TM approach. Fixed transfer matrices fo
r neutrinos and photons.
|
* commit 332f3a0
| Author: Domenik Ehlert <domenik.ehlert@desy.de>
| Date:   Fri Mar 10 09:57:54 2023 +0100
|
|     Implemented propagation matrix (TM) approach; includes complete over
haul of analysis code structure and rewrite of the total flux normalisatio
n. Moved more information from separate variables into source_model class.
|
| * commit ba172f3 (ntnu_gitlab/legacy)
|/  Author: Domenik Ehlert <domenik.ehlert@desy.de>
|   Date:   Tue Mar 14 09:21:13 2023 +0100
|
|       Provide compatibility with pre- transfer matrix analysis setup.
|
* commit 321ebf8
| Author: Domenik Ehlert <domenik.ehlert@desy.de>
| Date:   Mon Feb 20 14:39:25 2023 +0100
|
|     Re-fix total emssivity scaling. Separate normalisation point and min
imum energy integration threshold. Surface plot for R0 and gamma parameter
s, if beta1 x beta2 scan.
```

Adding a remote server

git remote add <name> <address>

```
❯ git remote add ntnu_gitlab git@gitlab.com:ntnu_nv_ify/app/crpropa-analysis.git
```

Verifying remote servers

git remote -v

```
❯ git remote -v
ntnu_gitlab     git@gitlab.com:ntnu_nv_ify/app/crpropa-analysis.git (fetch)
ntnu_gitlab     git@gitlab.com:ntnu_nv_ify/app/crpropa-analysis.git (push)
origin  git@gitlab.desy.de:domenik.ehlert/crpropa-analysis.git (fetch)
origin  git@gitlab.desy.de:domenik.ehlert/crpropa-analysis.git (push)
```

Add an ssh key

Our GitLab server is (apparently) set up to require a public-private key pair for access.

> need to generate an ssh key pair via ssh-keygen -C <email>

> upload PUBLIC key to GitLab profile

> add private key to keychain with ssh-add <private key>

> live demo

NB: Can also use ssh key for password-less authentication with HPC!
> copy pubkey into ~/.ssh/authorized_keys on server

Step 4: Push commit to remote

git push <remote_name> <branch>

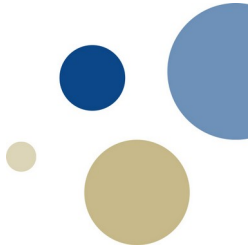
Push a single branch to the remote server.

git push --all

Push ALL branches to the remote server.

git push –all --force

Push ALL branches to the remote server and ignore merge conflicts (CAREFUL!).



Summary:
Getting started with Git & GitLab

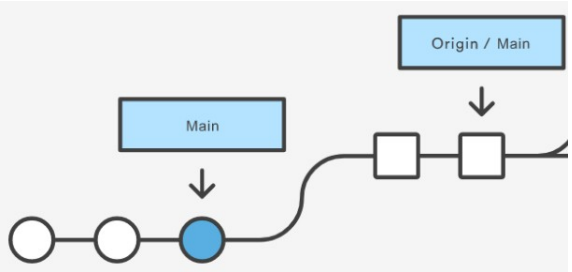
> live demo

Receiving Commits: Strict downstream

Assume a uni-directional workflow

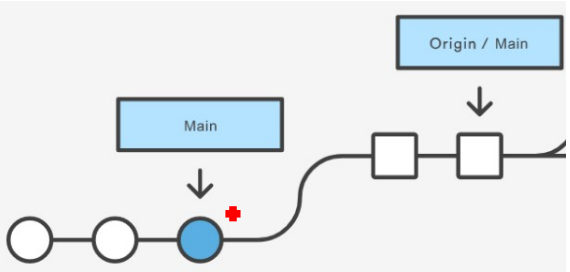
Receiving Commits: Strict downstream

Case 1: No changes on HPC



git fetch origin
git checkout main
git merge origin/main

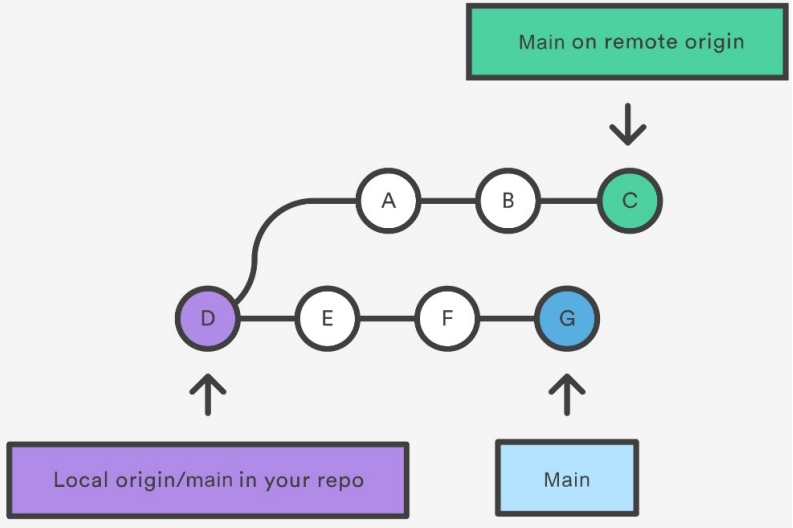
Case 2: Some irrelevant changes on HPC



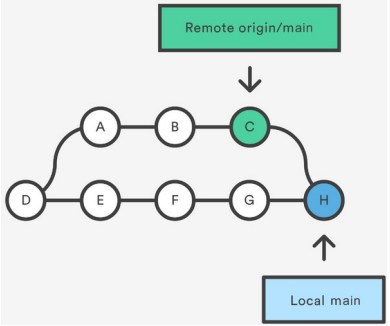
git fetch origin
git checkout main
git reset --hard origin/main

Merging and Rebasing

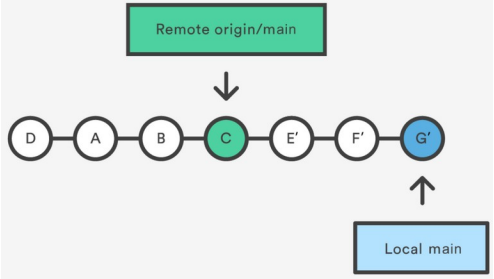
Now we have diverging histories …
(after a git fetch origin)



There are two different solutions:

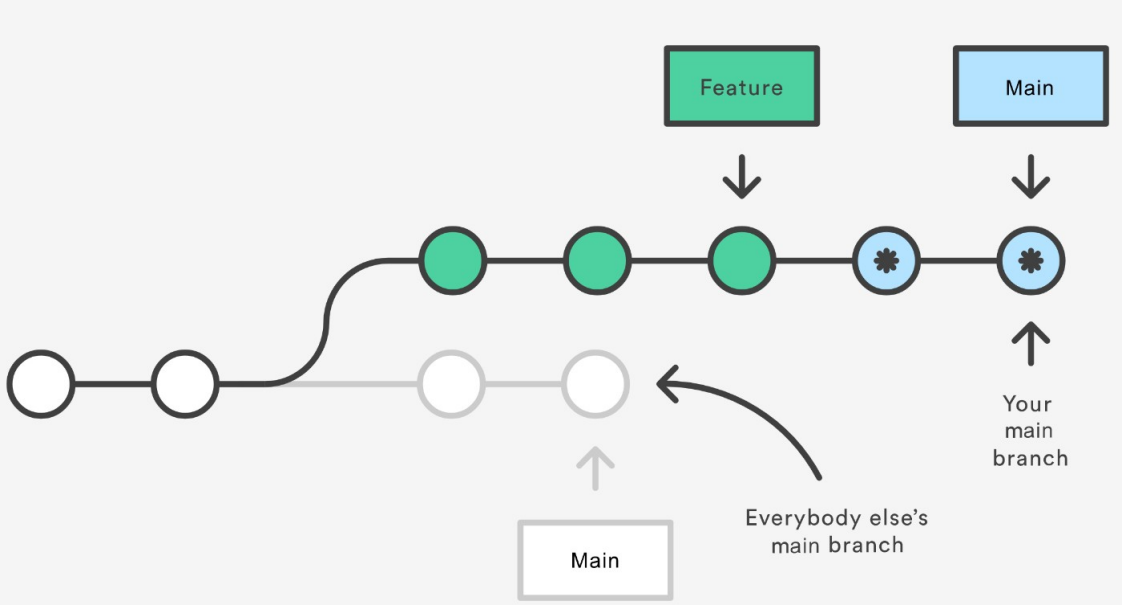


git merge
origin/main



git rebase
origin/main

Rebasing



Golden Rule of Rebasing:
Never rebase public branches!

Rebasing - continued

Pro Tip:
Modify previous commits with interactive rebase.

git rebase -i HEAD~n

n: # of commits to be modified

> git opens a text editor to define the modifications

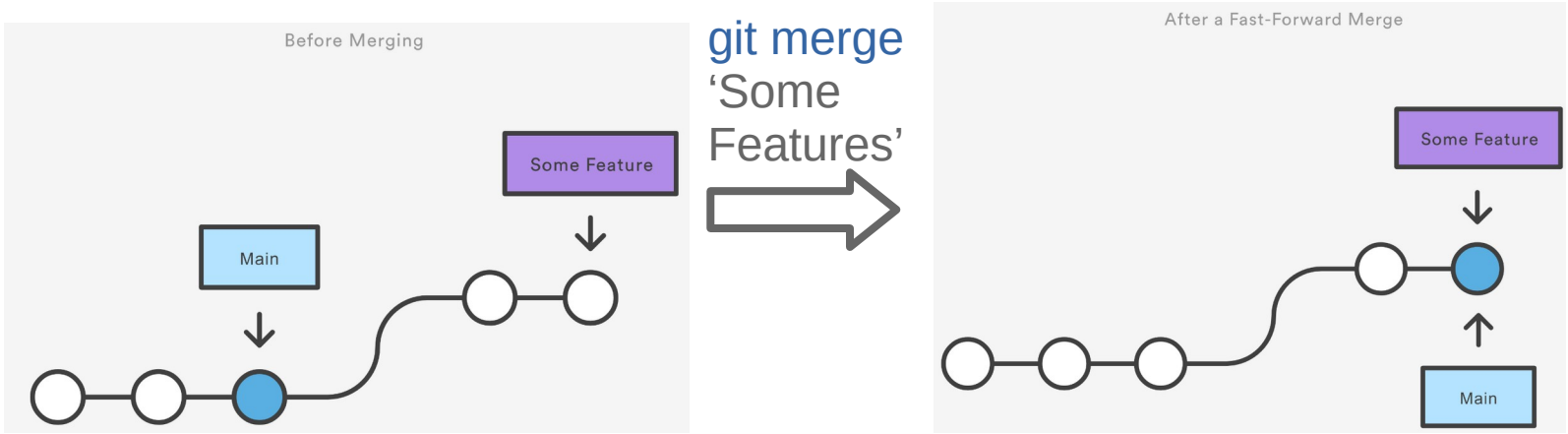
```
pick 332f3a0 Implemented propagation matrix (TM) approach; includes complete overhaul of an
pick 413c163 Switched lnA-calculations to TM approach. Fixed transfer matrices for neutrino
pick 82bea07 Updated calculation of multimessenger chi2 penalty. Only use penalty from an u

# Rebase 321ebf8..82bea07 onto 321ebf8 (3 commands)
#
# 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 [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# 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
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
#         create a merge commit using the original merge commit's
#         message (or the oneline, if no original merge commit was
#         specified); use -c <commit> to reword the commit message
# u, update-ref <ref> = track a placeholder for the <ref> to be updated
#                       to this position in the new commits. The <ref> is
#                       updated at the end of the rebase
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
```

Resolving Merge Conflicts

Make sure HEAD is at the receiving branch!

(git checkout main)

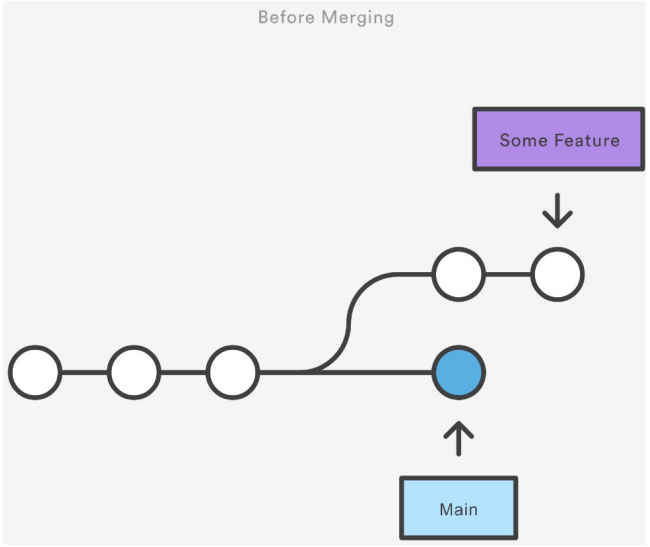
I. Fast Forward Merge – if one branch is strictly ahead



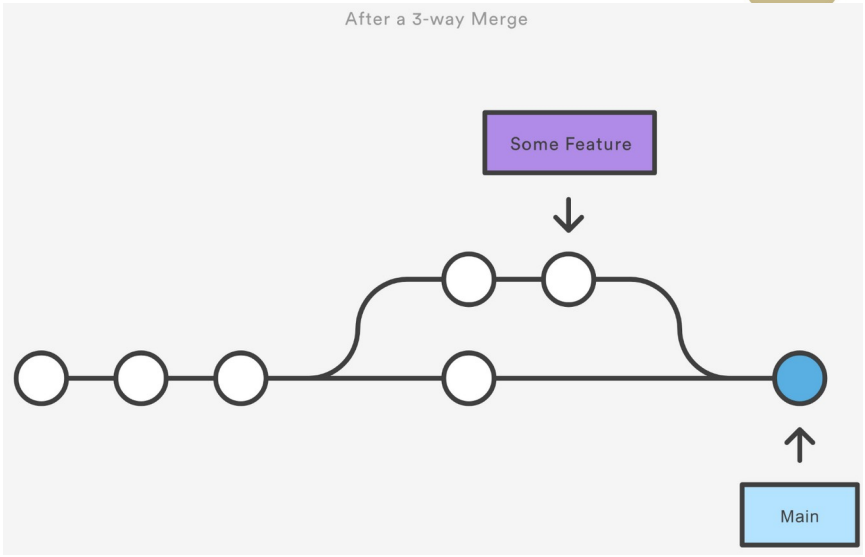
Git simply integrates all of the changes from the 'Some Features' branch into 'Main'.

Resolving Merge Conflicts

II. Three-way Merge – for diverging histories



git merge ‘Some Features’



Merge Conflict: If the same lines where changed in both files.

Resolution depends on merge strategy.

Resolving Merge Conflicts

II. Three-way Merge – for diverging histories

Standard merge strategy: ort (“Ostensibly Recursive’s Twin”)

Define merge option: git merge -X <option> <branch>

In case of conflict

ours	favour current version
theirs	favour other version
<none>	must resolve manually

Resolving Merge Conflicts

II. Three-way Merge – for diverging histories

```
here is some content not affected by the conflict
<<<<<<< main
this is conflicted text from main
=======
this is conflicted text from feature branch
>>>>>>> feature branch;
```

atlassian.com/git/tutorials

git edits the files and highlights the conflicts.

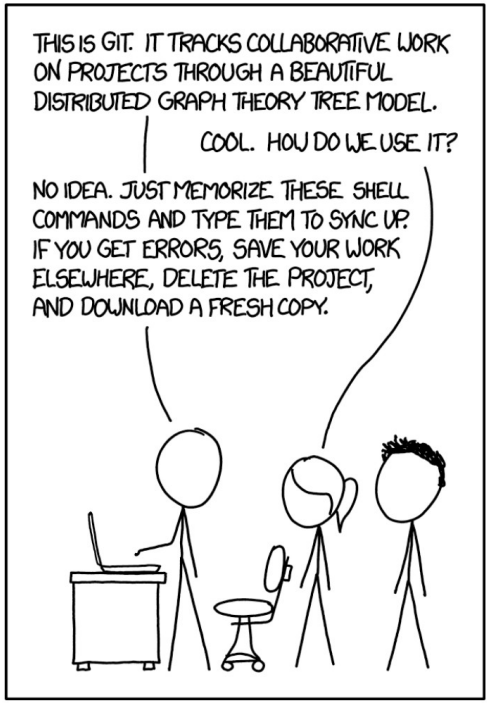
Pick a good text editor!

Resolve conflicts manually, then git add the changed files and git commit as usual.

Note: A successful merge does NOT guarantee that the code makes sense.

Best practices

- Commit regularly
- Write useful commit messages (what, why & how)
- Avoid complicated merges by keeping branches short-lived



Additional Links

Git Tutorial
Git Reference
SSH Keys and GitLab
Oh My Git!