git

Nick Papior @ DTU Computing Center

a short tutorial

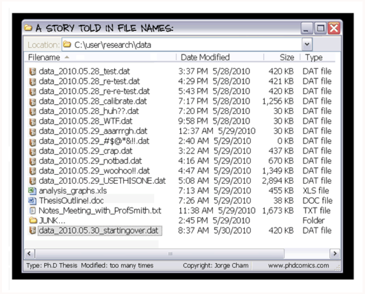
6 April

Outline

Be sure to have a working shell (use ThinLinc if able!)

Code development using Version control system

Local usage



A STORY TOLD IN FILE NAMES:

Location: C:\user\research\data

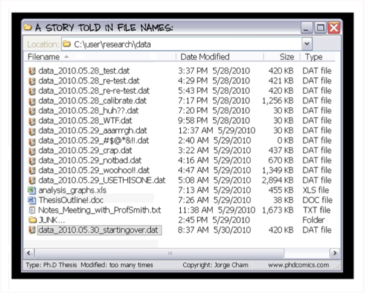
Filename	Date Modified	Size	Type
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	7:20 PM 5/28/2010	30 KB	DAT file
data_2010.05.28_WTF.dat	9:58 PM 5/28/2010	30 KB	DAT file
data_2010.05.29_aaarrrgh.dat	12:37 AM 5/29/2010	30 KB	DAT file
data_2010.05.29_#$@*&!!.dat	2:40 AM 5/29/2010	0 KB	DAT file
data_2010.05.29_crap.dat	3:22 AM 5/29/2010	437 KB	DAT file
data_2010.05.29_notbad.dat	4:16 AM 5/29/2010	670 KB	DAT file
data_2010.05.29_woohoo!!.dat	4:47 AM 5/29/2010	1,349 KB	DAT file
data_2010.05.29_USETHISONE.dat	5:08 AM 5/29/2010	2,894 KB	DAT file
analysis_graphs.xls	7:13 AM 5/29/2010	455 KB	XLS file
ThesisOutline!.doc	7:26 AM 5/29/2010	38 KB	DOC file
Notes_Meeting_with_ProfSmith.txt	11:38 AM 5/29/2010	1,673 KB	TXT file
JUNK...	2:45 PM 5/29/2010		Folder
data_2010.05.30_startingover.dat	8:37 AM 5/30/2010	420 KB	DAT file

Type: Ph.D Thesis Modified: too many times Copyright: Jorge Cham www.phdcomics.com

- Trial-error based approach for testing code
- Easy transfer of projects between machines
- Archival of system settings/documents
- Figure out when a change was made

Code development using Version control system

Local usage



A STORY TOLD IN FILE NAMES:

Location: C:\user\research\data

Filename	Date Modified	Size	Type
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	7:20 PM 5/28/2010	30 KB	DAT file
data_2010.05.28_WTF.dat	9:58 PM 5/28/2010	30 KB	DAT file
data_2010.05.29_aaarrrgh.dat	12:37 AM 5/29/2010	30 KB	DAT file
data_2010.05.29_#$@*&!!.dat	2:40 AM 5/29/2010	0 KB	DAT file
data_2010.05.29_crap.dat	3:22 AM 5/29/2010	437 KB	DAT file
data_2010.05.29_notbad.dat	4:16 AM 5/29/2010	670 KB	DAT file
data_2010.05.29_woohoo!!.dat	4:47 AM 5/29/2010	1,349 KB	DAT file
data_2010.05.29_USETHISONE.dat	5:08 AM 5/29/2010	2,894 KB	DAT file
analysis_graphs.xls	7:13 AM 5/29/2010	455 KB	XLS file
ThesisOutline!.doc	7:26 AM 5/29/2010	38 KB	DOC file
Notes_Meeting_with_ProfSmith.txt	11:38 AM 5/29/2010	1,673 KB	TXT file
JUNK...	2:45 PM 5/29/2010		Folder
data_2010.05.30_startingover.dat	8:37 AM 5/30/2010	420 KB	DAT file

Type: Ph.D Thesis Modified: too many times   Copyright: Jorge Cham   www.phdcomics.com

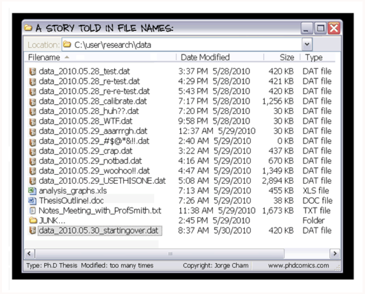
- Trial-error based approach for testing code
- Easy transfer of projects between machines
- Archival of system settings/documents
- Figure out when a change was made

VCS

- Retains all information about advancement of arbitrary data (preferentially text files)
- History is search-able by file, content or both (timestamps)
- Eases collaboration with people located abroad or neighbouring desk
- Allows *testing* new code without destroying developments – and *fast*

Code development using Version control system

Local usage



A STORY TOLD IN FILE NAMES:

Location: C:\user\research\data

Filename	Date Modified	Size	Type
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	7:20 PM 5/28/2010	30 KB	DAT file
data_2010.05.28_WTF.dat	9:58 PM 5/28/2010	30 KB	DAT file
data_2010.05.29_aaarrrgh.dat	12:37 AM 5/29/2010	30 KB	DAT file
data_2010.05.29_#$@*&!!.dat	2:40 AM 5/29/2010	0 KB	DAT file
data_2010.05.29_crap.dat	3:22 AM 5/29/2010	437 KB	DAT file
data_2010.05.29_notbad.dat	4:16 AM 5/29/2010	670 KB	DAT file
data_2010.05.29_woohoo!!.dat	4:47 AM 5/29/2010	1,349 KB	DAT file
data_2010.05.29_USETHISONE.dat	5:08 AM 5/29/2010	2,894 KB	DAT file
analysis_graphs.xls	7:13 AM 5/29/2010	455 KB	XLS file
ThesisOutline!.doc	7:26 AM 5/29/2010	38 KB	DOC file
Notes_Meeting_with_ProfSmith.txt	11:38 AM 5/29/2010	1,673 KB	TXT file
JUNK...	2:45 PM 5/29/2010		Folder
data_2010.05.30_startingover.dat	8:37 AM 5/30/2010	420 KB	DAT file

Type: Ph.D Thesis Modified: too many times   Copyright: Jorge Cham   www.phdcomics.com

- Trial-error based approach for testing code
- Easy transfer of projects between machines
- Archival of system settings/documents
- Figure out when a change was made

VCS

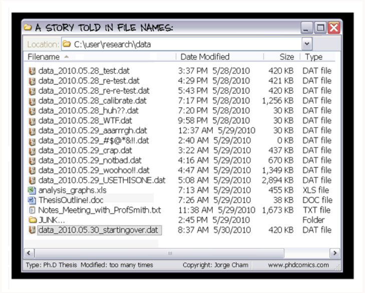
- Retains all information about advancement of arbitrary data (preferentially text files)
- History is search-able by file, content or both (timestamps)
- Eases collaboration with people located abroad or neighbouring desk
- Allows *testing* new code without destroying developments – and *fast*

Where `git` projects can be hosted

- `github.com/gitlab.com`
- your local machine
- *many more ...*

Code development using Version control system

Local usage



A STORY TOLD IN FILE NAMES:

Location: C:\user\research\data

Filename	Date Modified	Size	Type
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	7:20 PM 5/28/2010	30 KB	DAT file
data_2010.05.28_WTF.dat	9:58 PM 5/28/2010	30 KB	DAT file
data_2010.05.29_aaarrrgh.dat	12:37 AM 5/29/2010	30 KB	DAT file
data_2010.05.29_#$@*&!!.dat	2:40 AM 5/29/2010	0 KB	DAT file
data_2010.05.29_crap.dat	3:22 AM 5/29/2010	437 KB	DAT file
data_2010.05.29_notbad.dat	4:16 AM 5/29/2010	670 KB	DAT file
data_2010.05.29_woohoo!!.dat	4:47 AM 5/29/2010	1,349 KB	DAT file
data_2010.05.29_USETHISONE.dat	5:08 AM 5/29/2010	2,894 KB	DAT file
analysis_graphs.xls	7:13 AM 5/29/2010	455 KB	XLS file
ThesisOutline!.doc	7:26 AM 5/29/2010	38 KB	DOC file
Notes_Meeting_with_ProfSmith.txt	11:38 AM 5/29/2010	1,673 KB	TXT file
JUNK...	2:45 PM 5/29/2010		Folder
data_2010.05.30_startingover.dat	8:37 AM 5/30/2010	420 KB	DAT file

Type: Ph.D Thesis Modified: too many times Copyright: Jorge Cham www.phdcomics.com

- Trial-error based approach for testing code
- Easy transfer of projects between machines
- Archival of system settings/documents
- Figure out when a change was made

VCS

- Retains all information about advancement of arbitrary data (preferentially text files)
- History is search-able by file, content or both (timestamps)
- Eases collaboration with people located abroad or neighbouring desk
- Allows *testing* new code without destroying developments – and *fast*

Where `git` projects can be hosted

- `github.com/gitlab.com`
- your local machine
- *many more ...*

Powered by `git`

- `overleaf.com`
- `gitpod.io`
- *many more ...*

Outline

git setup – required

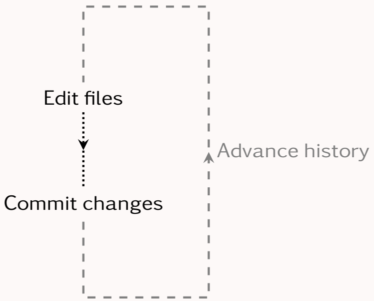
These credentials will be stored in the version control history, global configure file *or* configuration per project.

```
git config --global user.name "Your Name"
git config --global user.email "name@dtu.dk"
git config --global core.editor "nano"
```

or whatever you prefer; ”code –wait –new-window”

git workflow

Basics

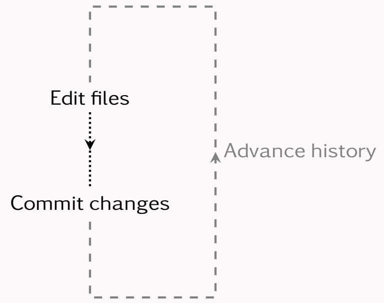


Each commit does:

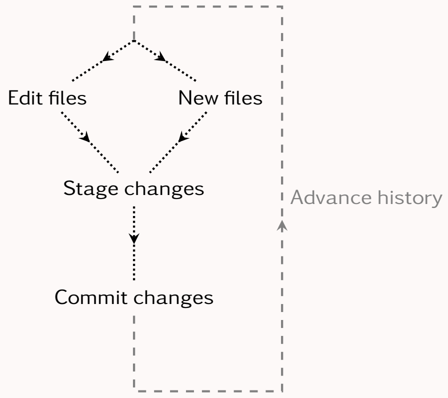
- Save staged changes
- Time stamp it
- Adds a message to the history

`git` workflow

Basics



Basics +



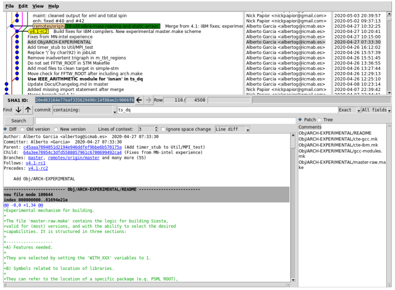
Each commit does:

- Save staged changes
- Time stamp it
- Adds a message to the history

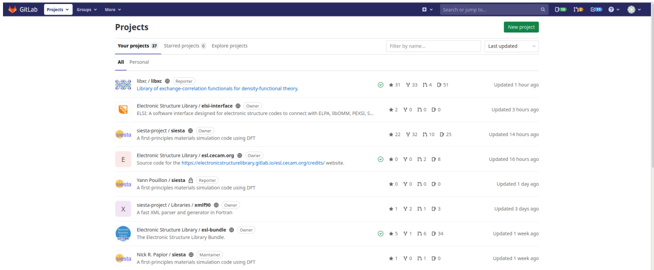
GUI’s and more

While `git` is mostly a terminal based program it may be viewed through many GUI variants. These GUI’s are mainly for history searches[1] and finding particular diffs fast.

- `gitk` (Linux/MacOS)
- `Git GUI` (Windows)



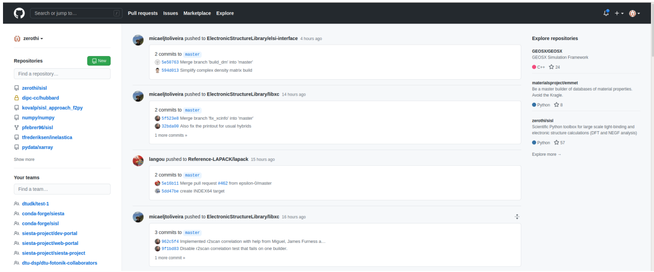
```
maint: cleaned output for xml and total spin                 Nick Papior <nickpapior@gmail.com>   2020-05-03 20:39:57
enh: fixed #40 and #42                                       Nick Papior <nickpapior@gmail.com>   2020-05-02 09:37:13
remotes/origin/39-util-vibra-mass-reading-and-static-arrays  Merge from 4.1: IBM fixes; experimer   Alberto Garcia <albertog@icmab.es>   2020-04-27 10:32:25
v4.1-rc2  Build fixes for IBM compilers. New experimental master.make scheme   Alberto Garcia <albertog@icmab.es>   2020-04-27 10:20:41
Fixes from MN-intel experience                               Alberto Garcia <albertog@icmab.es>   2020-04-27 10:15:00
Add Obj/ARCH-EXPERIMENTAL                                    Alberto Garcia <albertog@icmab.es>   2020-04-27 07:33:30
Add timer_stub to Util/MPI_test                              Alberto Garcia <albertog@icmab.es>   2020-04-26 16:12:02
Replace '\' by char(92) in jobList                           Alberto Garcia <albertog@icmab.es>   2020-04-26 15:57:39
Remove inadvertent trigraph in m_tbt_regions                 Alberto Garcia <albertog@icmab.es>   2020-04-26 15:51:45
Do not set FFTW_ROOT in STM Makefile                         Alberto Garcia <albertog@icmab.es>   2020-04-26 13:36:55
Add mod files to clean target in simple-stm                  Alberto Garcia <albertog@icmab.es>   2020-04-26 13:27:44
Move check for FFTW_ROOT after including arch.make           Alberto Garcia <albertog@icmab.es>   2020-04-26 12:29:13
Use IEEE_ARITHMETIC module for 'isnan' in ts_dq              Alberto Garcia <albertog@icmab.es>   2020-04-26 12:25:10
Update Docs/Changelog.md in master                           Alberto Garcia <albertog@icmab.es>   2020-04-08 10:23:14
Added missing import statement after merge                   Nick Papior <nickpapior@gmail.com>   2020-04-07 22:39:42

SHA1 ID: 20ed63164e77eaf335629490c14f88ae2c9066f6   Row 116 / 4508
Find  commit containing: ts_dq   Exact   All fields
Diff  Old version  New version   Lines of context: 3   Ignore space change   Line diff

Author: Alberto Garcia <albertog@icmab.es>  2020-04-27 07:33:30
Committer: Alberto <Garcia>  2020-04-27 07:33:30
Parent: c45aaa7094051d2194e946ddfef9bbe6b570175a (Add timer_stub to Util/MPI_test)
Child:  64a3ee70954c3dfd5588857961c6700699492ca4 (Fixes from MN-intel experience)
Branches: master, remotes/origin/master and many more (55)
Follows: v4.1-rc1
Precedes: v4.1-rc2

    Add Obj/ARCH-EXPERIMENTAL

------------------------ Obj/ARCH-EXPERIMENTAL/README ------------------------
new file mode 100644
index 000000000..81694e21e
@@ -0,0 +1,34 @@
+Experimental mechanism for building.
+
+The file 'master-raw.make' contains the logic for building Siesta,
+valid for (most) versions, and with the ability to select the desired
+capabilities. It is structured in three sections:
+
+------------------
+A) Features needed.
+
+They are selected by setting the 'WITH_XXX' variables to 1.
+
+B) Symbols related to location of libraries.
+
+They can refer to the location of a specific package (e.g. PSML ROOT),

Patch  Tree
Comments
Obj/ARCH-EXPERIMENTAL/README
Obj/ARCH-EXPERIMENTAL/cte-gcc.mk
Obj/ARCH-EXPERIMENTAL/cte-ibm.mk
Obj/ARCH-EXPERIMENTAL/gcc-modules.mk
Obj/ARCH-EXPERIMENTAL/master-raw.make
```



Projects

Your projects 37 · Starred projects 6 · Explore projects — All · Personal — Last updated

Project	Description	Stars	Forks	MRs	Issues	Updated
libxc / **libxc** (Reporter)	Library of exchange-correlation functionals for density-functional theory.	31	33	4	51	Updated 1 hour ago
Electronic Structure Library / **elsi-interface** (Owner)	ELSI: A software interface designed for electronic structure codes to connect with ELPA, libOMM, PEXSI, S...	2	0	0	0	Updated 3 hours ago
siesta-project / **siesta** (Owner)	A first-principles materials simulation code using DFT	22	32	10	25	Updated 14 hours ago
Electronic Structure Library / **esl.cecam.org** (Owner)	Source code for the https://electronicstructurelibrary.gitlab.io/esl.cecam.org/credits/ website.	0	0	2	8	Updated 16 hours ago
Yann Pouillon / **siesta** (Reporter)	A first-principles materials simulation code using DFT	0	0	0	0	Updated 1 day ago
siesta-project / Libraries / **xmlf90** (Owner)	A fast XML parser and generator in Fortran	1	2	1	3	Updated 3 days ago
Electronic Structure Library / **esl-bundle** (Owner)	The Electronic Structure Library Bundle.	5	1	6	34	Updated 1 week ago
Nick R. Papior / **siesta** (Maintainer)	A first-principles materials simulation code using DFT	1	0	1	0	Updated 1 week ago



zerothi

Repositories: zerothi/sisl · dipc-cc/hubbard · kovalp/sisl_approach_f2py · numpy/numpy · pfebrer96/sisl · tfrederiksen/inelastica · pydata/xarray · Show more

Your teams: dtudk/test-1 · conda-forge/siesta · conda-forge/sisl · siesta-project/dev-portal · siesta-project/web-portal · siesta-project/siesta-project · dtu-dsp/dtu-fotonik-collaborators

micaeljtoliveira pushed to **ElectronicStructureLibrary/elsi-interface** 4 hours ago
2 commits to `master`
`5e50763` Merge branch 'build_dm' into 'master'
`594d013` Simplify complex density matrix build

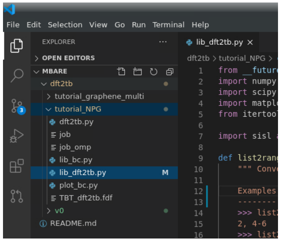
micaeljtoliveira pushed to **ElectronicStructureLibrary/libxc** 14 hours ago
2 commits to `master`
`5f523e8` Merge branch 'fix_xcinfo' into 'master'
`32bda00` Also fix the printout for usual hybrids
1 more commits »

langou pushed to **Reference-LAPACK/lapack** 15 hours ago
2 commits to `master`
`5e16b11` Merge pull request #462 from epsilon-0/master
`5dd47be` create INDEX64 target

micaeljtoliveira pushed to **ElectronicStructureLibrary/libxc** 16 hours ago
3 commits to `master`
`962c5f4` Implemented r2scan correlation with help from Miguel, James Furness a...
`9f1bd83` Disable r2scan correlation test that fails on one builder.
1 more commit »

Explore repositories
GEOSX/GEOSX — GEOSX Simulation Framework — C++ ☆ 24
materialsproject/emmet — Be a master builder of databases of material properties. Avoid the Kragle. — Python ☆ 8
zerothi/sisl — Scientific Python toolbox for large scale tight-binding and electronic structure calculations (DFT and NEGF analysis) — Python ☆ 57
Explore more →

VSCode



```
EXPLORER
OPEN EDITORS
MBARE
  dft2tb
    tutorial_graphene_multi
    tutorial_NPG
      dft2tb.py
      job
      job_omp
      lib_bc.py
      lib_dft2tb.py      M
      plot_bc.py
      TBT_dft2tb.fdf
    v0
  README.md
```

```
lib_dft2tb.py ×
dft2tb > tutorial_NPG >
from __futur
import numpy
import scipy
import matpl
from itertoo

import sisl

def list2ran
    """ Conv

    Examples
    --------
    >>> list
    2, 4-6
    >>> list
```

[1] My main usage!

Everyday git commands

`https://education.github.com/git-cheat-sheet-education.pdf`

Initialisation of repository (one time only)

1. `git init <project folder>`
2. `git clone <url>`

```
git init repo
cd repo
git clone https://github.com/LLNL/zfp.git
cd zfp
```

Staging files

1. `git add <new file>`
2. `git add <existing file>`

```
echo "New readme" > README
git add README
echo "Added text" >> README
git add README
```

Committing changes

1. `git commit`
2. `git commit -a`
   stages all tracked files; then commits *use with care*

```
git commit -m "Initial commit"
echo "Further text" >> README
git commit -a -m "Adjusted README"
```

Restoring files

1. `git restore|checkout < |file>.`
   revert unstanged changes
2. `git restore --staged < |file>.`
   unstage staged changes

```
echo "Further text" >> README
git restore README
echo "Further text" >> README
git add README
git restore --staged README
```

Everyday git commands

git reset < |revision|path>

Unstage files or uncommit commits, either retaining changes or not

```
# uncommit latest 2 commits, keep changes
git reset HEAD~2
# delete latest 2 commits, changes are lost
git reset --hard HEAD~2
```

---

git log < |revision|path>

History log for specific revisions/path
`--format=oneline`
`--abbrev-commit`

```
f41bc094d (HEAD -> main) doc: added new publication using sisl
1e580e42d (origin/main, origin/HEAD) travis: added py3.9 tests
50029fcbf bug: fixed wrong path vs. module names
004bc6908 maint: joined single line
a66b8783e bug: fixed a2o in-place changing numpy-arrays fix #280
```

---

git status < |path>

Status with respect to HEAD

```
On branch main
Your branch is ahead of 'remotes/origin/main' by 1 commit.

Changes not staged for commit:
modified:   README.md

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

---

git diff < |file|branch>

Difficult command to master and extremely helpful

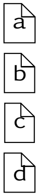
```
diff --git a/README b/README
index 67bb9529c..78a874799 100644
--- a/README
+++ b/README
@@ -1 +1,2 @@
 New readme
+Added text
```

Staging – committing

Preparing a commit

Selectively decide which changes goes to which commit

- no change
- an edit has been made
- file staged for next commit



Output of `git status`:

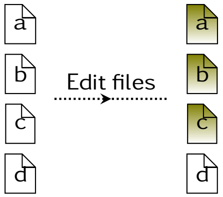
```
On branch main
nothing to commit (use -u to show untracked files)
```

Staging – committing

Preparing a commit

Selectively decide which changes goes to which commit

- 🗋 no change
- 🗋 an edit has been made
- 🗋 file staged for next commit



Output of `git status`:

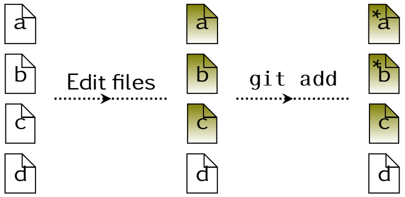
```
On branch main
Changes not staged for commit:

modified: a
modified: b
modified: c

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

Staging – committing

Preparing a commit

Selectively decide which changes goes to which commit

- 🗋 no change
- 🗋 an edit has been made
- 🗋 file staged for next commit



Output of `git status`:

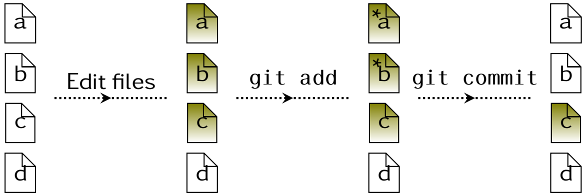
```
On branch main
Changes to be committed:

modified: a
modified: b

Changes not staged for commit:

modified: c
```

Staging – committing

Preparing a commit

Selectively decide which changes goes to which commit

- 🗋 no change
- 🗋 an edit has been made
- 🗋 file staged for next commit



Output of `git status`:

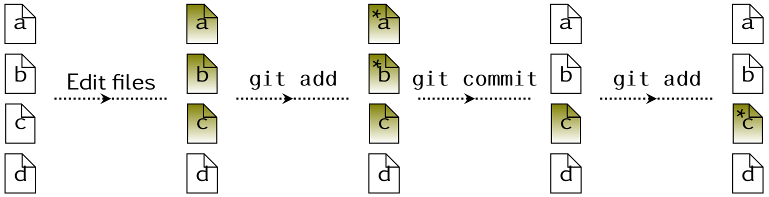
```
On branch main
Changes not staged for commit:

modified: c

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

Staging – committing

Preparing a commit

Selectively decide which changes goes to which commit

- 🗋 no change
- 🗋 an edit has been made
- 🗋 file staged for next commit



Output of `git status`:

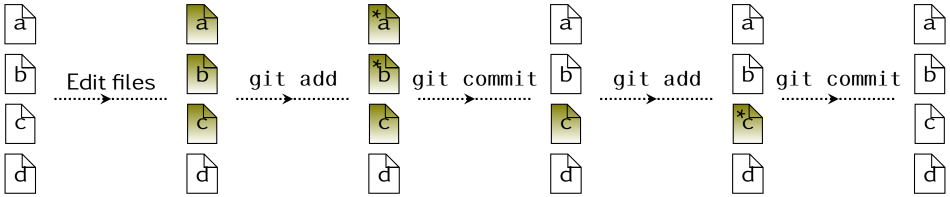
```
On branch main
Changes to be committed:

modified: c
```

Staging – committing

Preparing a commit

Selectively decide which changes goes to which commit

- 🗋 no change
- 🗋 an edit has been made
- 🗋 file staged for next commit



Output of `git status`:

```
On branch main
nothing to commit (use -u to show untracked files)
```

Tutorial 1

Task 1

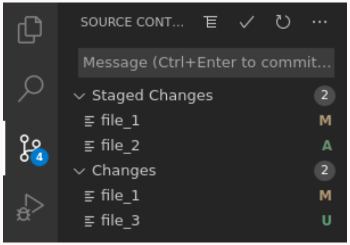
1. Create an empty git repository
2. Create and add a file to the repository
   - use `git diff/status/log` between each command
3. Create and add a second file to the repository
4. Use `git log` and check how it looks

Task 1 – advanced users

1. Add a file with at least 6 lines (including white-space between)
2. Now change at least 3 different lines (not next to each other)
   - Figure out `add -p/-u`
   - Figure out `commit --patch` and play with files for log messages (play with `<-e> --file FILE`
   - Figure out what `git status -v` and `git status -vv` does.

Task 2 – obtain a `status` like this / VS Code:

```
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
modified:   file_1
new file:   file_2

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:   file_1

Untracked files:
  (use "git add <file>..." to include in what will be committed)
file_3
```



Tutorial 1 — continued

Task 3 – obtain a `status` much like Task 2:

```
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
modified:   file_1
new file:   file_2

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:   file_1
```

1. Revert the not staged changes in `file_1` obtain a status like this:

```
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
modified:   file_1
new file:   file_2
```

2. Revert the staged changes in `file_1` (requires two commands)

```
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
new file:   file_2
```

Learned Objectives

- How to initialize a `git` repository
- How to add and commit files
- How to perceive the staging area
- How to query the status of a repository
- How to see the log of a repository

Learned Objectives

- How to initialize a `git` repository
- How to add and commit files
- How to perceive the staging area
- How to query the status of a repository
- How to see the log of a repository
- `git status` reminded us of `git restore`

`git restore|checkout`

Older versions of `git` used `git checkout` for restoring files and switching branches. Since 2.23 these are now separate commands (`git restore` and `git switch`) for clarity.
When searching for these things you will typically find the `git checkout` variant since that is what long-time users are used to.

Outline

Collaboration

Simultaneous development

Common workflow[2]

- Hosting Git repository at GitHub/GitLab/*any remote*

[2] This is e.g. how you are editing Overleaf documents

Collaboration

Simultaneous development

Common workflow[2]

- Hosting Git repository at GitHub/GitLab/*any remote*
- Several people have local copies and do simultaneous development

[2] This is e.g. how you are editing Overleaf documents

Collaboration
Simultaneous development

Common workflow[2]

- Hosting Git repository at GitHub/GitLab/*any remote*
- Several people have local copies and do simultaneous development
- People committing on the `main` branch

[2] This is e.g. how you are editing Overleaf documents

Collaboration

Simultaneous development

Common workflow[2]

- Hosting Git repository at GitHub/GitLab/*any remote*
- Several people have local copies and do simultaneous development
- People committing on the `main` branch
- Pushing development to hosting site

[2]This is e.g. how you are editing Overleaf documents

Collaboration

Simultaneous development

Common workflow[2]

- Hosting Git repository at GitHub/GitLab/*any remote*
- Several people have local copies and do simultaneous development
- People committing on the `main` branch
- Pushing development to hosting site

Remotes, copies and branches

- Generally a *single* remote is the reference repository (GitHub/GitLab/...)
- Each developer's repository is a copy of everything
- Each repository can have multiple branches
- Each branch starts at a specific commit and is a separate development entity

[2]This is e.g. how you are editing Overleaf documents

Collaboration

Simultaneous development

Common workflow[2]

- Hosting Git repository at GitHub/GitLab/*any remote*
- Several people have local copies and do simultaneous development
- People committing on the `main` branch
- Pushing development to hosting site

Remotes, copies and branches

- Generally a *single* remote is the reference repository (GitHub/GitLab/...)
- Each developer's repository is a copy of everything
- Each repository can have multiple branches
- Each branch starts at a specific commit and is a separate development entity

Working with remotes

- `git clone` to initialize a repository
- `git pull` to update your *local* copy with changes *upstream*
- `git push` update the remote host with local changes

Things generally work well if collaborators do not do the same thing at the same time.

[2] This is e.g. how you are editing Overleaf documents

Branches

Separating development

Idea (`git branch <new branchname>`)

- Allows retaining a functioning branch (`main`) while developing
- Big changes require a lot of incremental changes
- Develop on several branches simultaneously
- Merging branches is done automatically via incremental patches for each commit

Branches

Separating development

Idea (`git branch <new branchname>`)

- Allows retaining a functioning branch (`main`) while developing
- Big changes require a lot of incremental changes
- Develop on several branches simultaneously
- Merging branches is done automatically via incremental patches for each commit

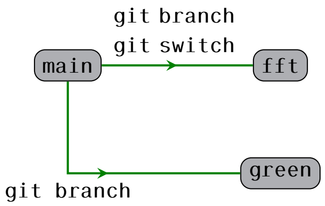
Branches

Separating development

Idea (`git branch <new branchname>`)

- Allows retaining a functioning branch (`main`) while developing
- Big changes require a lot of incremental changes
- Develop on several branches simultaneously
- Merging branches is done automatically via incremental patches for each commit

1. Create branch `git branch new-branch`
2. Switch to branch `git switch new-branch`



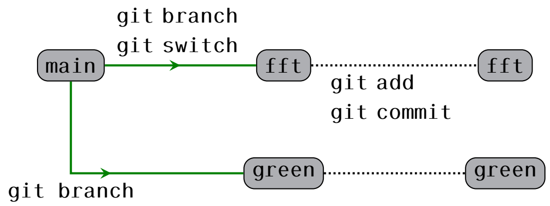
Branches

Separating development

Idea (`git branch <new branchname>`)

- Allows retaining a functioning branch (`main`) while developing
- Big changes require a lot of incremental changes
- Develop on several branches simultaneously
- Merging branches is done automatically via incremental patches for each commit

1. Create branch `git branch new-branch`
2. Switch to branch `git switch new-branch`
3. Make commits on this branch



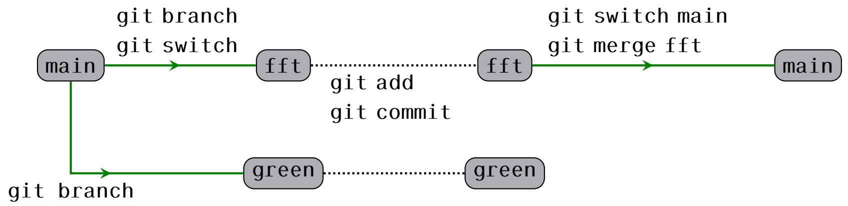
Branches

Separating development

Idea (`git branch <new branchname>`)

- Allows retaining a functioning branch (`main`) while developing
- Big changes require a lot of incremental changes
- Develop on several branches simultaneously
- Merging branches is done automatically via incremental patches for each commit

1. Create branch `git branch new-branch`
2. Switch to branch `git switch new-branch`
3. Make commits on this branch
4. Switch to main branch `git switch main`



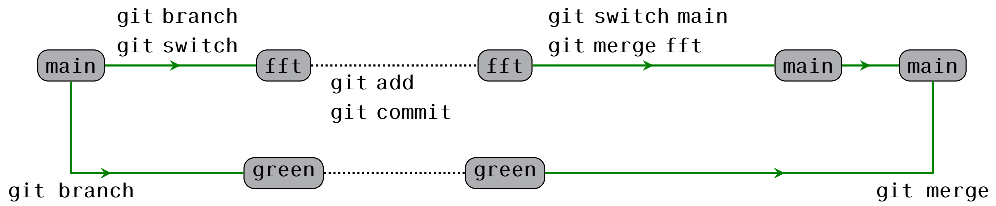
Branches

Separating development

Idea (`git branch <new branchname>`)

- Allows retaining a functioning branch (`main`) while developing
- Big changes require a lot of incremental changes
- Develop on several branches simultaneously
- Merging branches is done automatically via incremental patches for each commit

1. Create branch `git branch new-branch`
2. Switch to branch `git switch new-branch`
3. Make commits on this branch
4. Switch to main branch `git switch main`
5. Merge changes from `new-branch` `git merge new-branch`

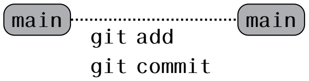


Now `main` has all the changes of the two separate development branches `fft` and `green`

Common branch problems

Common branch problems

Local (un)commited changes and a remote with new commits leads to divergence between branches.

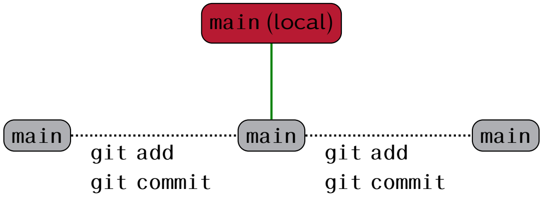


Nearly all problems are related to diverging histories/commits in different repositories. The more people that work together, the harder it will be to avoid problems. Coordinating efforts and where people work is the most effective way to steer clear of `git merge` problems.

Common branch problems

Common branch problems

Local (un)commited changes and a remote with new commits leads to divergence between branches.

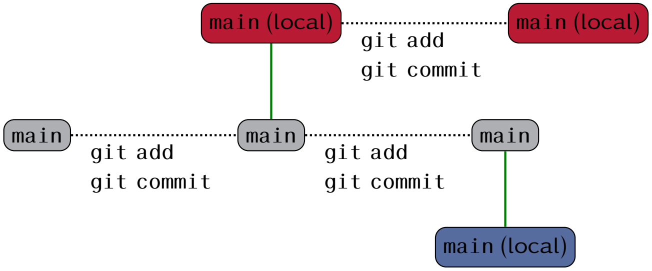


Nearly all problems are related to diverging histories/commits in different repositories. The more people that work together, the harder it will be to avoid problems. Coordinating efforts and where people work is the most effective way to steer clear of `git merge` problems.

Common branch problems

Common branch problems

Local (un)commited changes and a remote with new commits leads to divergence between branches.

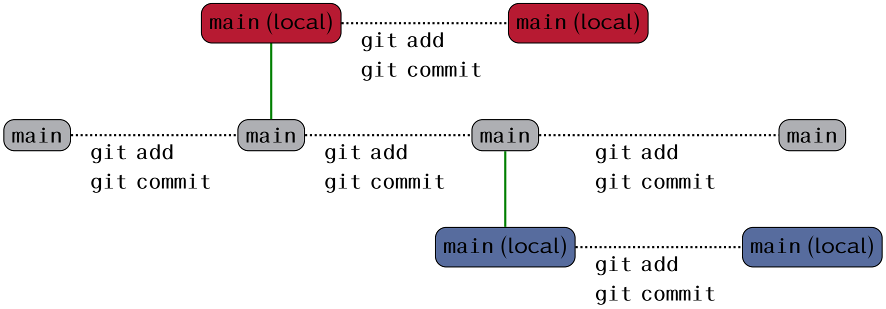


Nearly all problems are related to diverging histories/commits in different repositories. The more people that work together, the harder it will be to avoid problems. Coordinating efforts and where people work is the most effective way to steer clear of `git merge` problems.

Common branch problems

Common branch problems

Local (un)commited changes and a remote with new commits leads to divergence between branches.



Nearly all problems are related to diverging histories/commits in different repositories. The more people that work together, the harder it will be to avoid problems. Coordinating efforts and where people work is the most effective way to steer clear of `git merge` problems.

Tutorial 2

Playing with remote/branch problems

Be sure to use `git log/diff/status` commands OFTEN

Task 4 – create a remote at `https://gitlab.gbar.dtu.dk`

- Create a new project on the DTU GitLab server and follow instructions there to make your first commit.
- Make a few more commits and `git push` them as they are committed

Task 5 – `pull` problems

Trying to `pull` from a remote where history have diverged.

New `git` users

1. Clone the repository `https://gitlab.gbar.dtu.dk/02466F22/reprod_research.git`
2. Run the shell script `sh setup.sh 1`
3. Do a `git pull host` and try and understand the error message.
4. Check the history and try and recreate something similar

Prior `git` users

1. Ensure your remote (GitLab) repository from Task 4 is fully updated (`push`/`pull`)
2. Make a commit on your local repository, but do not push it
3. Clone your repository (from GitLab) from Task 4 into a new directory and create some commits that you push to the remote
4. Go back to your original local repository and do a `git pull`. Do you fully grasp the error message?

Repeat the task by editing a local file but not committing it.
Fixing the problem: reset, store changes, pull and apply changes; new branch, update, merge; rebasing at pull (advanced)

Tutorial 2

Playing with remote/branch problems

Be sure to use `git log`/`diff`/`status` commands OFTEN

Task 6 – push problems

Same as Task 5, but push to try and update the remote.

New `git` users

1. Clone the repository `https://gitlab.gbar.dtu.dk/02466F22/reprod_research.git`
2. Run the shell script `sh setup.sh 2`
3. Do a `git push host` and try and understand the error message.
4. Check the history and try and recreate something similar

Prior `git` users

1. Ensure your remote (GitLab) repository from Task 4 is fully updated (`push`/`pull`)
2. Clone your repository (from GitLab) from Task 4 into a new directory and create some commits that you push to the remote
3. Go back to your original local repository; do a commit.
4. Do a `git push` and try and understand the error message.

Fixing the problem: reset, store changes, pull, apply changes, push; new branch, update, merge/rebase, push.

Learned Objectives

- Always start with `git pull`: aligns local repository with remote
- Always end with `git push`: aligns remote repository with local

Learned objectives

- How to work with remotes
- Use `git push/pull` *OFTEN!*
- Forced common merge conflicts
- Each and every remote acts exactly like a separate branch, you are encouraged to do developments in `branches`

`git switch|checkout`

Older versions of `git` used `git checkout` for restoring files and switching branches. Since 2.23 these are now separate commands (`git restore` and `git switch`) for clarity.
When searching for these things you will typically find the `git checkout` variant since that is what long-time users are used to.

Outline

Permanently ignore specific files

Why ignoring files

Files not shown when querying the status of the repository (`git status`)

TeX	`log`, `aux`, etc.
C	`o`
FORTRAN	`o`, `mod`
Python	`pyc`

Permanently ignore specific files

Why ignoring files

Files not shown when querying the status of the repository (`git status`)

TeX	`log`, `aux`, etc.
C	`o`
FORTRAN	`o`, `mod`
Python	`pyc`

`.gitignore`

`~/.gitignore`	Globally for all projects (not recommended)
`.gitignore`	Locally for one project (recommended!)

Every line is a file name or regexp

filename	File `filename` will be disregarded
*.o	All files with extension `.o` will be disregarded
*.mod	All files with extension `.mod` will be disregarded
path/	entire directory

Permanently ignore specific files

Why ignoring files

Files not shown when querying the status of the repository (`git status`)

TeX	`log`, `aux`, etc.
C	`o`
FORTRAN	`o`, `mod`
Python	`pyc`

`.gitignore`

`~/.gitignore`	Globally for all projects (not recommended)
`.gitignore`	Locally for one project (recommended!)

Every line is a file name or regexp

filename	File `filename` will be disregarded
*.o	All files with extension `.o` will be disregarded
*.mod	All files with extension `.mod` will be disregarded
path/	entire directory

Task 7 – trying `.gitignore`

Do this in VS Code or your favourite code editor and see how its `git` interface behaves for changed files/ignored files etc.

1. Create some files with specific extensions, say `.ext`
2. Run `git status`
3. Add `*.ext` to the local `.gitignore` file
4. Run `git status` again, has anything changed?
5. Run `git status --ignored`, has anything changed?

Used commands

```
git init <name or .>          # one time only
git clone <url>               # one time only
git add <files or folders>
git commit
git diff
git status
git log
git branch <new branchname>
git switch <branchname>
git merge <branchname>
git restore <files>
git checkout
git push
git pull
git remote <>                 # one time only
git blame
```

Tips & Tricks

Further help – without internet

`man git <command>` are must reads once in a while

```
man gittutorial
man gittutorial-2
man giteveryday
man gitworkflows
```

Retain a functioning `main`

Always try to have one branch fully functional, development in separate branches

No development is too small for a branch

Aliases – shorthands

```
git config --global user.name "Your Name"
git config --global user.email "your@email.dk"
git config --global --list
```

Sites of interest

google git cheatsheets pdf.
Plenty of cheatsheets around, find one that suits you!

`https://github.com/jlord/git-it-electron`
A complete `git` tutorial using GitHub

`https://swcarpentry.github.io/git-novice/10-open/index.html`
A simple and elaborate Git tutorial

`https://ndpsoftware.com/git-cheatsheet.html#loc=local_repo`
Interactive overview cheat-sheet

`https://github.com/pcottle/learnGitBranching`
Sandbox interactive Git branching tutorial

`https://nvie.com/posts/a-successful-git-branching-model/`
Git flow ; collaboration and branch models for workflow practices
please note his Reflection!