Workshop @Uni Bonn, D. Barbi (dbarbi@uni-bonn.de)

Introduction to git

Table of contents

Introduction to git

1. git basics

-- the stuff you should remember

git command line help

git has a nice command line help:

- ***`git help`***
- ***`git help [config/commit/rm/...]`***

Gives some hints on how to use the commands, on available flags,…

Configuring git – mandatory

git wants to know who you are:

- ***git config --global user.email ***
  dirk.barbi@awi.de
- ***git config --global user.name ***
  "Dirk Barbi"

This info is needed to "sign" your commits.

Create a new (local) git project

- ***`cd ~/my_project`***

  (or wherever your stuff is)

- ***`git init`***

  (creates empty repository and .git-folder)

Cloning a git project

- ***git clone --username dbarbi \ https://url-to-your-project myfolder***

- ***cd myfolder***

Copies whole repository (all revisions) into folder "myfolder", working directory = master/HEAD).

Connecting a local git project to server

- ***`git remote add origin https://url-to-your-project`***

Identifies local repository (all revisions) with remote repository, so that changes can be exchanged.

Staging files / changes

"Staging" means: Mark a change so that it is added to the repository later (***commit***)

- ***git add ****
- ***git add *.f90***
- ***git add src/echam/configure***

git add does NOT add changes to the repository, but STAGES them!

Committing files / changes

- ***git commit –m "Blabla"***

Everything that has been changed AND STAGED becomes part of the new revision, which becomes HEAD-revision of trunk or branch.

- ***git commit –a –m "Blabla"***

Stage and commit all changes (svn-style).

Pushing files / changes to server

- ***git push***

Everything that has committed to master is sent to server (only works if merging isn't needed).
If remote repository is empty (master doesn't exist yet):

- ***git push –all*** or
- ***git push –u origin master***

Getting files / changes from server

- ***`git pull`***

Gets all changes to master from server and starts to merge (if necessary)

Status of your repository

- ***git status***

  Gives information on:
  - untracked files
  - new files
  - modified files
  - deleted files

- ***git status –s (short)***

Remember these:

- ***git help***

- ***(git init/clone)*** (ONCE per project)
- ***git add***
- ***git commit (-a)***
- ***git pull***
- ***git push***
- ***git status (-s)***

→ ***At least 95% of all you need!***

Introduction to git

2. Which files get into the repository?

--Adding, removing and excluding files

Stage a new file

- ***`git add blabla.txt`***

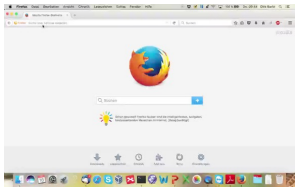
Stages the file blabla.txt. Next commit will actually add it to repository.

Committing files / changes

- ***`git commit –m “Blabla“`***

Everything that has been changed AND STAGED becomes part of the new revision, which becomes HEAD-revision of trunk or branch.

- ***`git commit –a –m “Blabla“`***



Stage and commit all changes (svn-style).

Remove a file

- ***rm blabla.txt***
- ***git rm blabla.txt***

Stages the file blabla.txt for removal. Next commit will remove it from repository AND YOUR WORKING COPY!!!
If blabla.txt already has staged changes, you need to type

- ***git rm –f blabla.txt***

Remove a file ONLY from repo

- ***`git rm --cached blabla.txt`***

Stages the file blabla.txt for removal. Next commit will remove it ONLY from repository.
(For example if you don't want to commit your TODO.txt, compiled files, input data,…)

Remove several files at once

- ***rm *.txt***
- ***git rm *.txt***

"*" needs to be escaped with a "\", because the expansion (replacement) of "*" is done by git, not the shell.

Move a file around

- ***`git mv blabla.txt important.txt`***

Does exactly what you think it does. Could be replaced by ***`mv`***, ***`git rm`***, ***`git add`***; but ***`git mv`*** keeps the history of the file intact.

Exclude files from repo

You can write a ***.gitignore***-file in the same folder that contains the ***.git***-folder and specify files NEVER to be added to the repository, e.g.:

```
# no .a files
*.a

# but do track lib.a
!lib.a

# ignore all .pdf files in the doc
# directory and subfolders
doc/**/*.pdf
```

Introduction to git

3. What have I done???

-- More information

What have I changed, but not staged yet?

- ***`git diff`***

Gives "diff-style" information on anything that has been done, but is not staged to be committed yet.

What have I changed and staged?

- ***`git diff --staged (or cached)`***

Gives "diff-style" information on anything that has been done, and that is staged already (so about to be committed).

Nicer diffs…

- ***`git difftool (--staged)`***

Does the same as git diff, but displays the result in a diff viewer like opendiff, vimdiff…

What has been done at all?

- ***git log***

Shows list of all commits, including author, date, and message. Try these too:

- ***git log -2***
- ***git log –p***
- ***git log --stat***
- ***git log --pretty=oneline --graph***
- ***git log --since=2.weeks***

Introduction to git

4. That's not how I want it

-- configuring git

Some nice settings

- ***git config --global core.editor ***
  vim
- ***git config --global diff.tool ***
  vimdiff
- ***git config –global merge.tool ***
  vimdiff

- ***git config --global alias.mylog ***
  "log --oneline --decorate --graph"

Introduction to git

5. Getting in sync

-- synchronizing with a server

Clone a server-based repository

- ***git clone --username dbarbi https://url-of-your-project myfolder***

Copies whole repository (all revisions) into working cpoy, working directory = master/HEAD.

Servers are called “remotes”.

Connecting a local git project to server

- ***`git remote add origin https://url-to-your-project`***

Identifies local repository (all revisions) with remote repository, so that changes can be exchanged.

Fetching changes from remote

- ***`git fetch`***

Connects to server, gathers all changes on all branches (including master) that are tracked from this remote, but doesn't merge with your stuff -> working directory remains unchanged.

Merge fetched changes

- ***`git merge FETCH_HEAD`***

Takes the changes downloaded by the last ***`git fetch`*** and tries to merge them into your working directory.

Pulling changes from remote

- ***git pull***

Connects to server, gathers all changes on all branches and master, and tries to merge with your stuff -> working directory will probably change!

→ ***git pull = git fetch; git merge FETCH_HEAD***

Pushing changes to remote

- ***`git push`***

Connects to server, sends all changes. Only works if you are have already the newest version of your branch!

(Otherwise you would need to ***`pull`***, then ***`push`***.)

Pushing changes to empty remote

If „master“ doesn‘t yet exist on the remote server, you need to use

- ***`git push –all`***

once. That sets up „master“ on the remote and pushes your working copy to it.

Show the remotes of your repo

- ***`git remote -v`***

Shows list of all remote servers that are known to git you can use to sync your project (yes, there can be more than one).

In our case, it's only swrepo1.awi.de, named "origin".

What goes on on the remotes?

- ***`git remote show`***

Gives address of remote, and what ***`git pull`*** and ***`git push`*** would try to do (can be quite helpful).

- ***`git branch –a`***

Introduction to git

6. More than one truth

-- branching and merging

Creating a new branch

- ***`git branch testing`***

Creates a new branch (“testing”), but doesn‘t switch to that branch.
A branch is actually only a POINTER to a revision! Other pointers are HEAD, master, FETCH_HEAD,…

Switching to another branch

- ***`git checkout testing`***

Switches to branch testing, so afterwards: HEAD → testing.

In general, that will change your files!!!

- ***`git checkout –b testing2`***

Creates branch testing2, and change to it.

Which branch am I on?

- ***`git branch`***

Shows all branches and indicates active one.

Merging one branch into another

- ***`git checkout testing`***
- ***`git merge testing2`***

Changes to branch testing, then merges (or tries to merge) all changes from testing2 into testing.

Sharing a branch on a server

- ***`git push origin testing`***

Push a branch to a server. Doesn't happen automatically – you can keep private branches for temporary work.

- ***`git push –u origin testing`***

Even better – local and remote branch keep connected for further ***`pull`***s.

What is "upstream"?

When remote and local branches are "linked", the remote branch is called upstream.

- ***git checkout testing*** → often automatic

Otherwise:

- ***git push –u***

Or:

- ***git branch --set-upstream-to master origin/master***
- ***git branch --set-upstream-to myname origin/blabla***

Deleting a branch

- ***git branch –d testing2***

Removes branch testing2 (that is: the pointer; the revisions stay.)

- ***git push origin --delete testing2***

Removes branch testing2 from server origin.

List all (local) branches

- ***`git branch –v`***
  List of all local branches

- ***`git branch –vv`***
  List of all tracking branches

- ***`git branch –a`***
  List of all branches,
  including remote ones

Introduction to git

7. Give mayor versions a name

-- tagging revisions

Create an (annotated) tag

- ***git tag -a v1.0 –m "version 1.0"***

Creates the tag "v1.0" together with message, author and date and adds it to the active revision.

Tag an older version

- ***git log --pretty=oneline***
- ***git tag -a v0.9 abcdef –m "blabla"***

Creates the tag "v0.9" and adds it to the revision with a revision number starting with abcdef.

Sharing tags

- ***`git push v1.0`***

Push tag “v1.0” to server.

- ***`git push --tags`***

Push all tags to server.

„Check out“ tag

- ***`git checkout -b version1 v1.0`***

Make a new branch (“version1”) and initialize it to the revision that is tagged as v1.0.

List tags

- ***`git tag`***

Shows a list of all defined tags known to your database.

- ***`git tag –l “v1.*”`***

List all tags starting with v1.

Show tag info

- ***`git show v1.0`***

Shows available information on tag v1.0, such as: author, date, tag description, revision number, last committing author, …

Introduction to git

8. Uuups...

-- undoing stuff

… I accidently removed / changed a file …

- ***git checkout -- myfile***

Gets file "myfile" from the repository the way it was at the last commit.
Also:

- ***git checkout c684v6 myfile***
- ***git checkout "c684v6~1" myfile***
- ***git checkout v1.0 myfile***
- ***git checkout mybranch -- myfile***

… I shoudn't have staged THAT…

- ***`git reset HEAD myfile2`***

Unstages file "myfile2", but leaves it unchanged in your working directory.

… much less committed…

Just keep on working, prepare your working directory as you want to commit it, then:

- ***`git commit --amend`***

Replaces the last commit (mostly needed when you messed up the commit message).

Introduction to git

Last words

Where to find information

- Pro Git book (free pdf version online)
- Cheat sheets
- Lots of online resources

- Get this document:

```
git clone https://USERNAME@gitlab.dkrz.de/esm-tools/git-workshop.git
```

git vs svn

git	svn
git clone	svn checkout
git pull	svn update
git push	svn commit
git diff	svn log
git add -A	svn import
git checkout -- file	svn revert file
git reset --hard HEAD	svn revert . -R

...again, remember these:

- ***git help***

- ***(git init/clone)*** (ONCE per project)
- ***git add***
- ***git commit (-a)***
- ***git pull***
- ***git push***
- ***git status (-s)***

→ At least 95% of all you need!