Git for physicists

Andrzej Kapanowski

Institute of Physics, Jagiellonian University, Krakow, Poland

December 21, 2015

Outline

Motivation

- Scientists need many tools for their work: software for analysing or visualizing data, sharing files, collaborating, writing up papers, publishing, searching the literature. Programming languages are needed to write unique programs or to glue different systems.
- Parallel computing, cloud computing (PLGrid), big data, bioinformatics (genomics data), ...
- Software Carpentry Foundation - teaching researchers basic software skills (from 1998).
- Publishing source code of scientific programs - Nature 467 (2010).
- There are many tools to learn - start from the best ones.

Getting started

- Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
- Git home page: `http://git-scm.com/`.
- Version control is a system that records changes to a set of files over time so that you can recall specific versions later.
- Local VCS: keeping patch sets.
- Centralized VCS: a single server contains all the versioned files.
- Distributed VCS: clients fully mirror the repository.
- Git is available for all major platforms.
- Protocols: HTTP, FTP, rsync, Git protocole.

A short history of Git

- Linus Torvalds (born in 1969) - the creator of the Linux kernel.
- 1991-2002, Linux kernel with patches and archived files.
- 2002-2005, Linux kernel with BitKeeper.
- 2005, Git was born.
- Git = "unpleasant person" (British English slang).
- Git thinks about its data like a stream of snapshots.

What to keep in a Git repo?

Short answer: text-based files.

- Source code of programs written in Shell, Python, C/C++, ...
- Websites (HTML, XHTML, CSS).
- Articles, reports, books (*.txt, *.tex, *.cvs).
- Source code of programs for computer algebra systems (Maxima, Maple, Mathematica).
- Source code of figures in gnuplot, Pyxplot, ...

Use Dropbox (2.5 GB) or Google Drive (15 GB) for binaries.

First-time Git setup

The command line interface provides all Git commands.

```
$ git --version      # check if Git is installed

$ git config --global user.name "Andrzej Kapanowski"

$ git config --global user.email
                     "andrzej.kapanowski@uj.edu.pl"

$ git config --list     # checking settings

$ git help config     # getting help
$ git config --help
$ man git-config
```

Getting a Git repository

Initializing a repo in an existing directory (git init).

```
$ cd myproject

$ git init      # .git subdirectory is created

$ git add -A      # add all files to the staging area

$ git commit -m "First commit."
```

Cloning an existing repo (git clone).

```
$ git clone https://github.com/sympy/sympy.git
```

Checking the status of the repo

```
$ git status
# On branch master
nothing to commit (working directory clean)
$ vim README     # a new file is created
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what
#                              will be committed)
#
#    README
nothing added to commit but untracked files present
                              (use "git add" to track)
```

Viewing the commit history

```
$ git log           # note SHA-1 hashes
commit 48ae450a329c92558038de3b721a029d8eea8c14
Author: Andrzej Kapanowski <andrzej.kapanowski@uj.edu.pl>
Date:   Wed Dec 9 09:54:56 2015 +0100

    UnionFind changed.

commit 64ec8f8edb9ca891b18d94d906d885f91e67e220
Author: Andrzej Kapanowski <andrzej.kapanowski@uj.edu.pl>
Date:   Tue Dec 8 14:12:12 2015 +0100

    Sudoku 6x6 added.

...
```

Viewing the commit history

```
$ git log --pretty=oneline -4
48ae UnionFind changed.
64ec Sudoku 6x6 added.
6355 Rich comparisons.
60a6 lekcja10 changed.

# Repo history:
# ...--60a6--6355--64ec--48ae <-- master <-- HEAD

$ git log --grep=lekcja   # "lekcja" in comments

$ git log --author=Andrzej
```

Viewing the commit history

```
$ git branch   # local branches
* master
$ git checkout 60a6
Note: checking out '60a6'.

You are in 'detached HEAD' state.
...
HEAD is now at 60a68aa... lekcja10 changed.
$ git branch
* (no branch)
  master
$ git checkout master
...
Switched to branch 'master'
```

Remote repositories

- github.com - over 30.2 million repositories, issue tracking, code reviews, syntax highlighted code, markdown for formatting text, paid plans. [GitHub Enterprise]
- bitbucket.org - Atlassian (JIRA, Confluence), pull request, branch permissions, free private repos for up to 5 users. [Bitbucket Server]
- gitlab.com - private repos, code reviews, issue tracking, wikis. [GitLab Community Edition and Enterprise Edition]

Working with remotes

```
# git remote add [shortname] [url]

$ git remote add origin
https://ufkapano@github.com/ufkapano/myproject.git

$ git push -u origin master   # only first time

$ git remote   # show shortnames of remotes
origin
```

Simple workflow

```
$ git pull

# Changing files ...

$ git add -A

$ git commit -m "Comments."

$ git push
```

Branches in Git

- A branch is a lightweight movable pointer to a commit.
- Creating, deleting, and modifying branches is quick and easy.
- The default branch name is master. This branch points to the last commit we made. It moves forward automatically after every commit.
- Switching branches changes files in the working directory.

Local branches

```
# git branch [branch_name] [sha1]

$ git branch testing
# A--B--C <-- master <-- HEAD
#       ^
#       |
#       testing
$ git checkout testing
Switched to branch 'testing'
# A--B--C <-- master
#       ^
#       |
#       testing <-- HEAD
```

Fast forward

```
# A--B--C <-- master
#        \
#         D--E <-- testing <-- HEAD

$ git checkout master   # it changes files!
Switched to branch 'master'
$ git merge testing
Updating cccc..eeee
Fast-forward
...
# A--B--C--D--E <-- master <-- HEAD
#               ^
#               |
#               testing
```

Removing branches

```
# A--B--C <-- master
#        \
#         D--E <-- testing <-- HEAD

$ git checkout master   # it changes files!
Switched to branch 'master'

$ git branch -d testing   # removing 'testing'

# A--B--C <-- master <-- HEAD
#        \
#         D--E (dangling commits)
```

Merging

```
# A--B--D <-- testing
#     \
#      C <-- master <-- HEAD

$ git merge testing      # edit comments
Merge made by the 'recursive' strategy.
...
# A--B--D <-- testing
#     \  \
#      C--E <-- master <-- HEAD

$ git log --pretty=oneline -1
eeee Merge branch 'testing'
```

Merging with conflicts

```
# A--B--D <-- testing
#     \
#      C <-- master <-- HEAD

$ git merge testing
Auto-merging a1.txt
CONFLICT (content): Merge conflict in a1.txt
Automatic merge failed;
fix conflicts and then commit the result.
```

Merging with conflicts

```
$ cat a1.txt
file a1
<<<<<<< HEAD
a1 changed in master
=======
a1 changed in testing
>>>>>>> testing
```

Merging with conflicts

```
$ git mergetool
merge tool candidates: meld opendiff kdiff3 tkdiff
    xxdiff tortoisemerge gvimdiff diffuse ecmerge
    p4merge araxis bc3 emerge vimdiff
Merging:
a1.txt

Normal merge conflict for 'a1.txt':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (gvimdiff):
```

Merging with conflicts

```
$ git status
# On branch master
# Changes to be committed:
#
#    modified:   a1.txt
#
# Untracked files:
#   (use "git add <file>..."
#   to include in what will be committed)
#
#    a1.txt.orig
```

Merging with conflicts

```
$ rm a1.txt.orig

$ git add -A

$ git commit -m "Merge branch testing."
```

Remote branches

```
$ git branch --all          # list all branches
* master                    # local branch
  remotes/origin/master     # remote branch
```

git pull = git fetch + git merge

```
$ git push
...
 ! [rejected]          master -> master (non-fast-forward)
...

# git fetch [remote_name] [branch_name]
$ git fetch origin master

$ git merge origin/master   # conflicts are possible ...

$ git add -A
$ git commit -m "Merge branch origin/master."
$ git push
```

Summary

- Use Git to handle small and large projects like creating software, writing scientific articles, books, websites.
- Use remote repositories to collaborate with other persons, to publish your software or data, for backups.

Thank you for your attention