E.T.
EFFICIENT TOOLS for RESEARCH

VERSION CONTROL with GIT
~ accelerated tutorial for busy academics ~
Version Control with Git

accelerated tutorial
for busy academics

Pierre Haessig
CentraleSupélec Rennes, January 24, 2018
Why Git?

1) Version Control system:
keep track of changes (history)
   • Git = most popular VCS since 2010
   • Alternatives: Subversion (svn), Mercurial

2) Collaboration
   • Publish and on the web
     - including the history
     - always up-to-date
   • Collaborate
     manage asynchronous contributions,
     with potential conflicts.
Outline of the training

1) **Personal work**: single user on its local computer
   
   *week-long break*

2) **Publish work** on the Internet: hosted Git services (GitHub, GitLab)

3) **Collaborative work**: keep in sync, manage conflicts
1) Personal work

single user on its local computer

(no leaks on a “cloud” 😊, but no backup either 😞)
Personal work (local computer)

- Setting up Git: install and configure (once per machine)
- Initialize an empty git repository,
- Track changes: add changes to the staging area, create commits
- Compare versions (diff) and explore the history (log)

Practice 1
based on
http://swcarpentry.github.io/git-novice/
Setting up Git
(once per machine)

- Install Git. Instructions for Windows:
     On page [https://git-scm.com/download/win](https://git-scm.com/download/win), download should start automatically.
  2) then I suggest TortoiseGit as a convenient graphical tool [https://tortoisegit.org/](https://tortoisegit.org/)
- Create empty folder and open “Git Bash”
- Configure identity: `git config (name & email)`

http://swcarpentry.github.io/git-novice/02-setup/
Creating an (empty) Repository

- `git init`
- → Observe new “.git” directory (unhide hidden files and folders)

http://swcarpentry.github.io/git-novice/03-create/
Tracking changes

Actions:

1) Put changed files to the staging area: `git add`
2) Save staged content as a new commit: `git commit`

Check the Status of the repository: `git status`
Exploring History

• Compare versions (diff): `git diff`
  
  - Ex: `git diff HEAD~1 script.m`

• Explore the history (the graph of all commits): `git log`

(Easier with graphical tools, c.f. next)
Places

Workspace
Index/Staging area
Local Repository

Commands

`git ...`

config
init
add
commit
status
diff
log
checkout
Graphical Git tools

• (Too) many to choose!
• Windows shell: TortoiseGit
• IDE integration: e.g. Matlab
TortoiseGit

- https://tortoisegit.org/
- A “Windows Shell Interface”, i.e., in the file explorer: “right click” → context menu
Git in Matlab

- Integration in the “Current Folder” panel of Matlab Desktop
  - e.g. handling of binary files like Simulink’s .slx
Week-long break
2) **Publish** work on the Internet

Hosted Git services:
- GitLab (at CentraleSupélec)
- GitHub
GitLab

GitLab instance hosted at CS: https://gitlab.centralesupelec.fr/

Login with your usual LDAP username
Publish work

Based on the previously created local Git repository (Practice 1)

→ On CentraleSupélec GitLab website: create a new repository.
→ On local computer, add a remote, then push the local commits.
→ On GitLab: explore the web interface, look at the commits.

Practice 2

based on
http://swcarpentry.github.io/git-novice/07-github/ (with GitLab instead)
3) Collaborative work

“How to keep in sync?”

**Clone** an existing repository

**Pull** (fetch & merge) fresh changes

Manage potential **conflicts**
Collaborate

Based on the previously created online (GitLab) Git repository (Practice 2)

➔ make pairs: “Owner” and “Collaborator”
➔ Collab. clones the GL repo of Owner (needs O to give rights to C)
➔ Collab. makes local changes, commit and push
➔ Owner pulls those changes

Practice 3

based on
http://swcarpentry.github.io/git-novice/08-collab/
The “ping-pong” Git workflow

Notice: Git provides **no lock mechanism**. Work is asynchronous.

**No conflicts**, if people work in very different time zones 🌍!
Manage merge conflicts

Again in pairs “Owner” and “Collaborator”

➔ Collab clones the GL repo of Owner (needs O to give rights to C)

➔ Owner AND Collab make local changes, commit

➔ Both attempt to push

➔ The first to push is fine 😊

➔ The second gets an error (“local is behind remote”)
   → needs to pulls the first changes, and merge its own changes (automatic or needs manual conflict resolution)

➔ Then second creates a merge commit and finally makes a push

http://swcarpentry.github.io/git-novice/09-conflict/
Merge commit process

1) Everybody synced
2) Local commits
3) Collab pushes first
   → Owner’s view of remote is outdated
4) Owner pulls and make a merge commit
5) Owner pushes
   Collab is outdated
6) Collab pulls
   → everybody synced

http://swcarpentry.github.io/git-novice/09-conflict/
See also Git Branching - Branches in a Nutshell from “Pro Git” book
Review: typical workflows

• Locally:
  1) make changes
  2) add to staging
  3) commit to local repo

• With a remote:
  1) pull from remote (before making changes if possible)
  2) make local commits
  3) push to remote. If error (local behind remote), pull to merge.
Ressources

Gentle introduction:

- “Version Control with Git” lesson from Software Carpentry
  http://swcarpentry.github.io/git-novice/
- Interactive tutorial https://try.github.io

Comprehensive book:

  free to read online https://git-scm.com/book/
Quick References (Git “Cheatsheets”)

From Software Carpentry’s lesson Quick Reference page:

- Printable PDF (EN & FR) https://services.github.com/on-demand/resources/cheatsheets/
- Interactive webpage http://ndpsoftware.com/git-cheatsheet.html