Version Control Sofware: git

PHY1610 – Lecture 6

Ramses van Zon & Marcelo Ponce

SciNet HPC Consortium/Physics Department
University of Toronto

January 28th, 2021

Today's class

Today we will discuss the following topics:

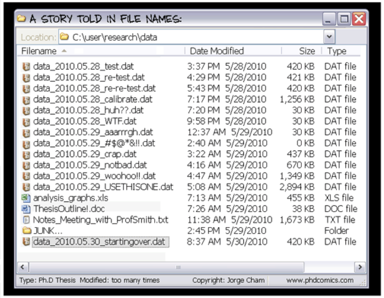
- Version control using git.

Best Practices on Scientific/Professional Software Development

- Modularity
  implementation and header files: .cc/cpp and .h files
- Automation Building Tool
  make
- Version Control
  git
- Defensive Programming
- Unit Testing
  eg. boost/STL

Version Control

- Version Control is a tool for managing changes in a set of files.
- Keeps historical versions for easy tracking.
- It essentially takes a snapshot of the files (code) at a given moment in time.



A STORY TOLD IN FILE NAMES:

Location: C:\user\research\data

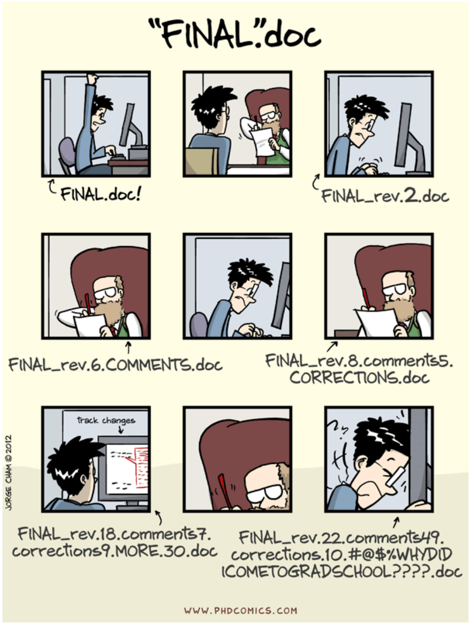
Filename	Date Modified	Size	Type
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	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

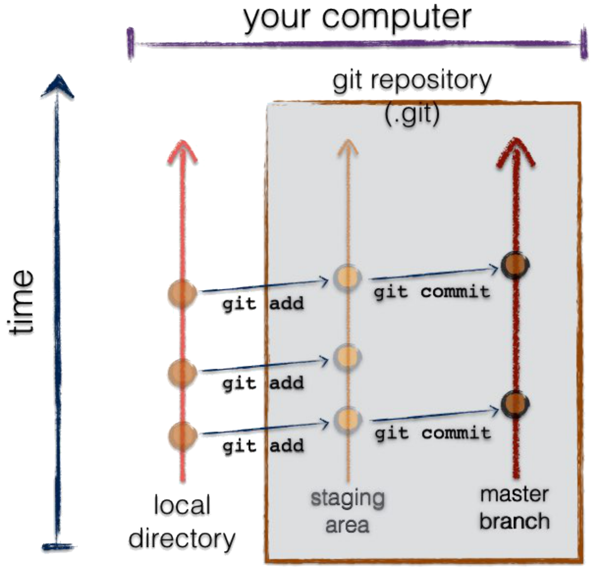
src: PhD Comics

- Why use it?
  - Makes collaborating on code easier/possible/less violent.
  - Helps you stay organized.
  - Allows you to track changes in the code.
  - Allows reproducibility in the code.

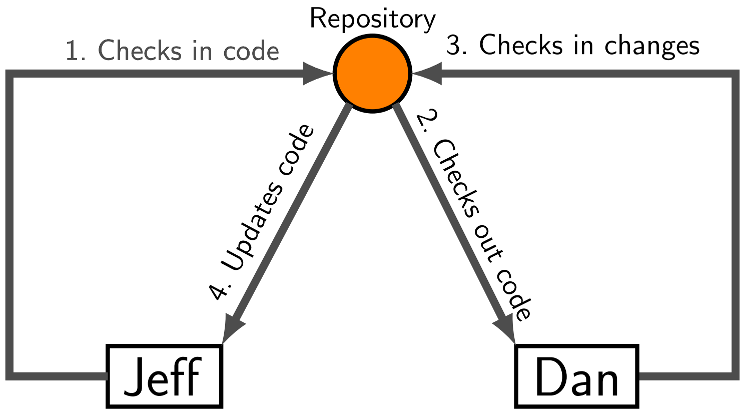
And when something goes wrong, you can back up to the last working version.



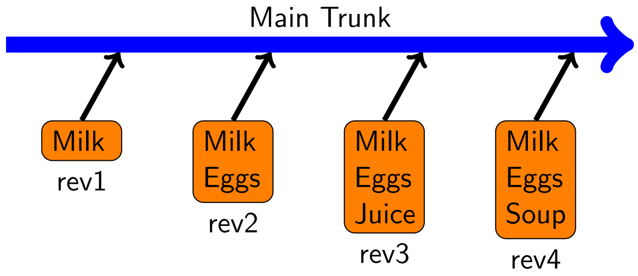
src: `https://git-scm.com`


your computer
git repository
(.git)
time
git add
git add
git add
git commit
git commit
local
directory
staging
area
master
branch

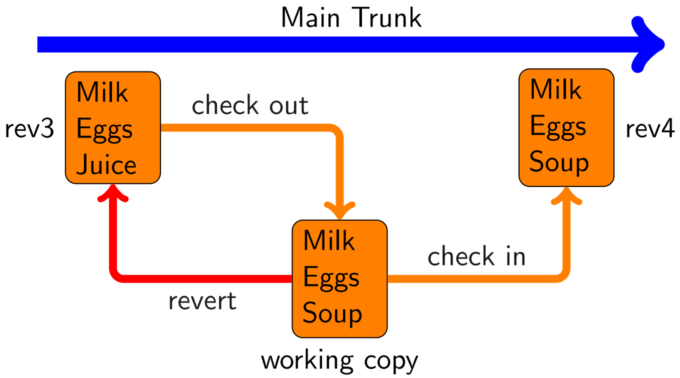
How does version control work?



Basic Checkins



Checkout and edit



Version Control: git

There are many types and approaches to version control. Here we will introduce one implementation: git.

There are four main things you need to know how to do to get started with git:

* Setup git on your computer.
- Initialize a git repository.
- Commit files to the repository.
- Delete files from the repository.
- Where to find more information.

* Linux
  > `yum/.../apt-get install git`
* MacOS
  > Xcode
  > fink/macports/homebrew
  > git OSX installer
* Windows: MobaXterm
  > `apt-get install git`

Version control: setup a *local* repository

The first thing to do is set up a repository for your code.

```
mponce@mycomp:~>  cd code
mponce@mycomp:~/code>  git init
Initialized empty Git repository in /home/s/scinet/mponce/code/.git/
mponce@mycomp:~/code>
```

This creates a .git directory, in the code directory, which contains the repository information.

```
mponce@mycomp:~/code>  ls -a
.      ..      .git
mponce@mycomp:~/code>
```

Version Control: setup your "id"

The first time you might try to use git to commit, it might complain if it can't identify who are you...

```
*** Please tell me who you are.
Run
git config --global user.email ‘‘youremail@example.com’’
git config --global user.name ‘‘FirstName LastName’’
to set your account’s default identity.
Omit --global to set the identity only in this repository.
fatal:  empty indent name (for <(null)>) not allowed
```

Or, just check in advance:

```
mponce@mycomp:~/code>  git config user.name
mponce@mycomp:~/code>  git config user.email
```

➠ **Fix:** follow git instructions...

```
mponce@mycomp:~/code>  git config --global user.email ‘‘mponce@scinet.utoronto.ca’’
mponce@mycomp:~/code>  git config --global user.name ‘‘Marcelo Ponce’’
```

Version control: adding repository files

Adding files to the repo – First you must **add** the files to the 'staging' area, then you **commit**:

```
mponce@mycomp:~/code>  echo "some data" > temp.txt
mponce@mycomp:~/code>  cp temp.txt temp2.txt
mponce@mycomp:~/code>  ls
temp2.txt     temp.txt
mponce@mycomp:~/code>  git add .  # include all files in the commit.
mponce@mycomp:~/code>  git commit -m "First commit for my repository."
[master (root-commit) f60c07d] First commit for my repository.
2 files changed, 2 insertions(+), 0 deletions(-)
create mode 100644 temp.txt
create mode 100644 temp2.txt
mponce@mycomp:~/code>
```

Notice that you must always 'stage' (`git add`) the files before actually commiting them (`git commit -m ''...''`).

Version Control: comparing file versions

Let's update some data and see how can we compare it with the already commited files...

```
mponce@mycomp:~/code>  echo "some more data" >> temp.txt
```

```
mponce@mycomp:~/code>  git diff temp.txt
diff --git a/temp.txt b/temp.txt
index 4268632..fdd9353 100644
--- a/temp.txt
+++ b/temp.txt
@@ -1 +1,2 @@
some data
+some more data
```

```
mponce@mycomp:~/code>  git add temp.txt
mponce@mycomp:~/code>  git commit -m ''updating data due to ...''
```

Version Control: recovering file versions

Revisiting what it has been done in the repo: logs

```
mponce@mycomp:~/code>  git log
commit 5e3e51323391e550c42920f60c07da5e36c9dcd5
Author: Contributor #1 <user1@scinet.utoronto.ca>
Date: Wed Oct 18 16:34:31 2016 -0500

updating data due to ...

commit f60c07da5e36c9dcd55e3e51323391e550c42920
Author: Contributor #1 <user1@scinet.utoronto.ca>
Date: Wed Oct 18 14:34:31 2016 -0500

First commit for my repository.
```

Recover an specific version:

```
mponce@mycomp:~/code>  git checkout f60c07da5e36c9dcd55e3e51323391e550c42920
```

Version Control: Reverting changes – Reset, Checkout, and Revert

Command	Scope	Common use cases
`git reset`	Commit-level	Discard commits in a private branch or throw away uncommited changes
`git reset`	File-level	Unstage a file
`git checkout`	Commit-level	Switch between branches or inspect old snapshots
`git checkout`	File-level	Discard changes in the working directory
`git revert`	Commit-level	Undo commits in a public branch
`git revert`	File-level	(N/A)

More details at `https://www.atlassian.com/git/tutorials`

Version control: removing repository files

Let's look at what we've done so far.

```
mponce@mycomp:~/code>  git log
commit f60c07da5e36c9dcd55e3e51323391e550c42920
Author: Marcelo Ponce <mponce@scinet.utoronto.ca>
Date: Wed Jan 8 14:34:31 2014 -0500

First commit for my repository.
```

But suppose you want to delete a file?

```
mponce@mycomp:~/code>  git rm temp2.txt
rm 'temp2.txt'
```

```
mponce@mycomp:~/code>  git add .
```

```
mponce@mycomp:~/code>  git commmit -m "Remove temp2.txt."
[master 95c1ef3] Remove temp2.txt
1 files changed, 0 insertions(+), 1 deletions(-)
delete mode 100644 temp2.txt
```

Summary: Setting up a GIT-repo

➡ download and install GIT (eg. `https://git-scm.com/download`)

only once, at the beggining of a new project:

1. start your project, eg. create a new folder or directory
   1.A) you may need to set up your identity (`git config --global user.name/user.email`)
2. set up the repo: `git init`

as many times as needed, as the project evolves and grows...

- add/modify/... files within your repo
  1. update changes (`git add`)
  2. commit your changes (`git commit -m ''...''`)

* check your commits and file status, by using `git log` and `git status`

[ADV] Version Control - git: branches

Version Control, Star Trek Style
revision 1
revision 2
branch 2a
branch 2b
Aieee! Roll back! Roll back!

```
#shows current branch
    git branch
#shows all branches
    git branch -a
#shows all remote branches
    git branch -r
# creates the branch "myNEWbranch"
    git branch myNEWbranch
# switch to the branch "myNEWbranch"
    git checkout branchname
```

```
git branch [--color[=<when>] | --no-color] [-r | -a] [--list] [-v [--abbrev=<length> | --no-
abbrev]] [--column[=<options>] | --no-column] [(--merged | --no-merged | --contains) [<commit>]] [-
-sort=<key>] [--points-at <object>] [<pattern>...]
git branch [--set-upstream | --track | --no-track] [-l] [-f] <branchname> [<start-point>]
git branch (--set-upstream-to=<upstream> | -u <upstream>) [<branchname>]
git branch --unset-upstream [<branchname>]
```

[ADV] Version Control - git: remote repos...

* "remote" machine (sever):

```
ssh USR@myServer.somewhere.IP
mkdir my_project.git
cd my_project.git
git init --bare
git update-server-info # If
planning to serve via HTTP
exit
```

* local machine:

```
cd my_project
git init
git add *
git commit -m "My initial commit message"
git remote add origin
git@example.com:my_project.git
git push -u origin master
```

- other copies:

```
git clone USR@myServer.somewhere.IP:my_project.git
```

* pulling & committing changes to/from the repo...

```
#check for diffs between local version
and commited version in the repo...
  git diff
#retrieve changes from the repo
  git pull
```

```
#commit local changes to the repo
  git add ...
  git commit -m "..."
  git push
```

Version Control: git ...

```
mponce@mycomp:~/code>  man git
```

```
mponce@mycomp:~/code>  git --help
```

```
add        Add file contents to the index
bisect     Find by binary search the change that introduced a bug
branch     List, create, or delete branches
checkout   Checkout a branch or paths to the working tree
clone      Clone a repository into a new directory
commit     Record changes to the repository
diff       Show changes between commits, commit and working tree, etc
fetch      Download objects and refs from another repository
grep       Print lines matching a pattern
init       Create an empty Git repository or reinitialize an existing one
log        Show commit logs
merge      Join two or more development histories together
mv         Move or rename a file, a directory, or a symlink
pull       Fetch from and integrate with another repository or a local branch
push       Update remote refs along with associated objects
rebase     Forward-port local commits to the updated upstream head
reset      Reset current HEAD to the specified state
rm         Remove files from the working tree and from the index
show       Show various types of objects
status     Show the working tree status
tag        Create, list, delete or verify a tag object signed with GPG

'git help -a' and 'git help -g' list available subcommands and some concept guides.  See 'git
help <command>' or 'git help <concep>' to read about a specific subcommand or concept.
```

Version Control: a few tips

- Use it, will save you trouble.
- Commit often.
- Include sensible comment messages.
- Do not commit derivative stuff (eg log files, executables, compiled modules, ...)
- can be used for several different kind of projects: code development, collaborations, papers, ...

▶ Various different VC systems: git, hg, svn, cvs, ...

Version control: more information

There are many other things that can be done with git:

- Review differences between files in different commits.
- Go back to a previous version of the code.
- Branch the code to add new and wonderful features.
- Reconcile different branches of the code.

For a very extensive tutorial, go here:
`http://www.vogella.com/tutorials/Git/article.html`

Web-based options:

- GitHub: `https://github.com/`
- Bitbucket: `https://bitbucket.org`

* BitBucket: GIT Basics & Tutorials

`https://www.atlassian.com/git`
`https://www.atlassian.com/git/tutorials`