Using GIT for software version management

1. Basic Git Commands and Exercises

Author: Taco Walstra

University of Amsterdam

Faculty of Science

January 31, 2018

Text partly based on texts of software carpentry and Learn Git in a month of lunches

Contents

1

Installing Git on your computer

Install git from the following link for your operating system:

Linux: try to use "`sudo apt-get install git-all` for ubuntu or debian.
try to use "`sudo yum install git-all` for centos and fedora/redhat. See for alternative installs: http://git-scm.com/download/linux

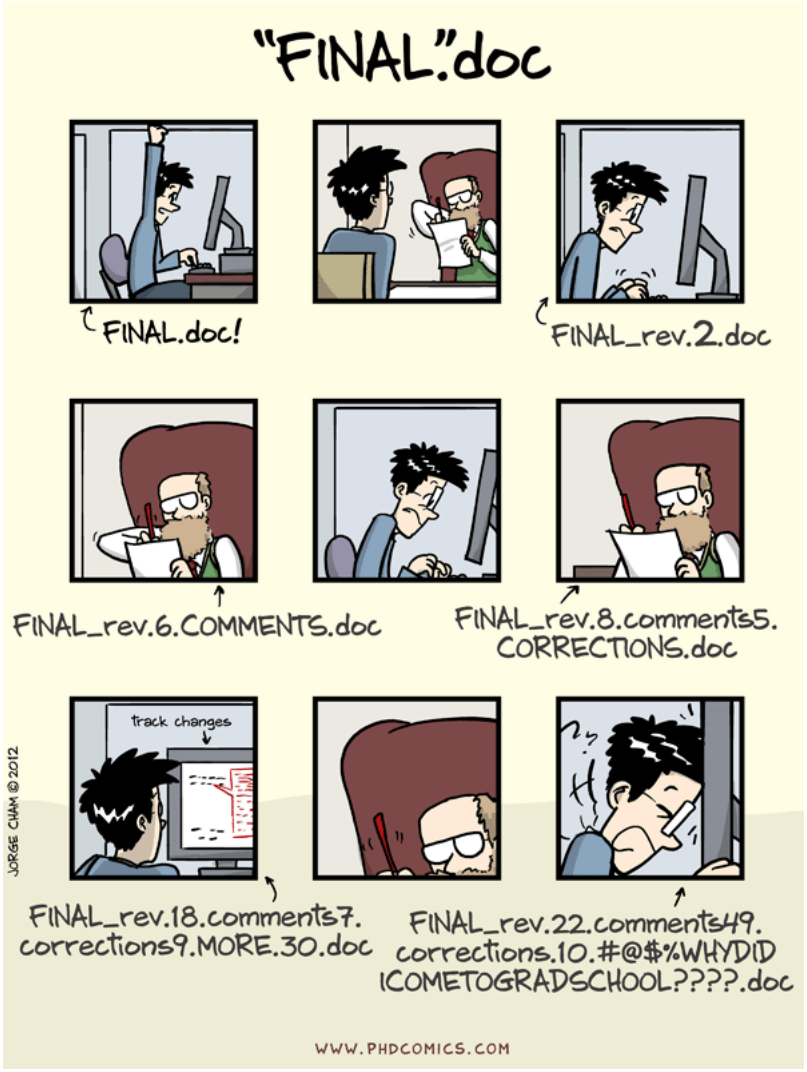
Mac: First try if it's already installed. Open a terminal window and type "`git --version`

If not installed go to http://git-scm.com/download/mac and download the git-xx.xx-intel-universal-mavericks.dmg image and install it as root user.

Windows: Go to http://git-scm.com/download/win and download the git-xx.xx-64-bit.exe file and install it as administrator.

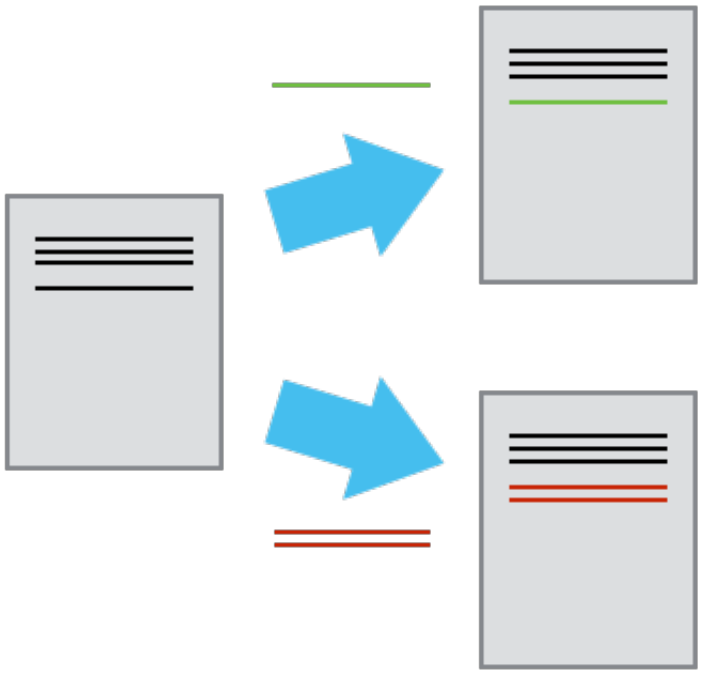
2

Setting up Git

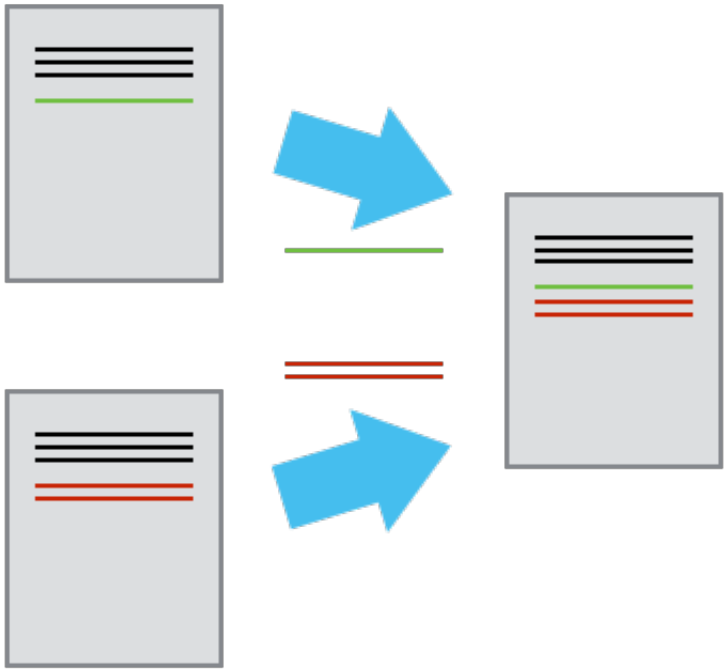


We have seen it all: word documents which are almost identical and difficult to compare. Word has the 'track changes' and Google Docs the version history tools to help with this. Software version management is more or less like this.

Once you have made a change to a file you can think of this as different sets of changes onto a base document and getting different versions of the document. See the picture below.



Which version is the correct one? It could be that the additions are both valid and can be merged into one new document



If multiple versions are merged we didn't encounter a problem, but often 2 people will create a conflict when two versions are merged. In such cases we need to resolve the conflict first. We will see this later.

2.1 Distributed version control

A version control system is a tool that keeps track of these changes for us and helps us version and merge our files. A system like Git is designed to keep multiple sets of versions and changes in sync across different computers or servers. This is called a distributed system.

A *repository* is the set of files that we want to keep under version control.

With Git, every user who wants to make changes to a repository has their own copy of the files in the repository, along with their own copy of the changes (the commits) that have been made to those files. Git keeps the commits in a hidden directory along with the copies of the files.

2.2 Setting up Git

```
$ git config --global user.name "Vlad Dracula"
$ git config --global user.email "vlad@tran.sylvan.ia"
$ git config --global color.ui "auto"
```

There are more config settings, but we will limited to these 3. You can find more options with the command
git config –list

2.3 Creating a Repository

Please note that git is a distributed version management system. We can create git repositories on a server but also on our local system. We start with a local system. The following commands create a repository using command line tools. I recommend to use these tools, even on Windows and to be sceptic on fancy Gui tools. Linux and Mac will make things easier but the Windows installation will also give you a shell (command line) application.

```
$ mkdir planets
$ cd planets
$ git init
$ ls -a
```

The last command will show a hidden directory with the name .git. This is were git will save all the changes to the files including git specific settings for your repository.

We can get the status of our git project with the 'git status' command (you would never have thought about that one).

2.4 Using git to track changes

Now we will create a few files and add them to the repository:

```
$ touch jupiter.txt
$ touch saturn.txt
$ git status
#on branch master
#
# Initial commit
#
# Untracked files:
# (use "git add <file> ..." to include in what will be committed)
#
# jupiter.txt saturn.txt
nothing added to commit but untracked files present (use "git add" to track)

$ git add jupiter.txt saturn.txt
$ git status
#on branch master
#
# Initial commit
#
# Changed to be committed:
# (use "git rm --cached  <file> ..." to unstage)
#
# new file: jupiter.txt saturn.txt

$git commit -m"initial commit of 2 files"

[master (root-commit) f22b25e] Initital commit of 2 files
 2 files changed, 2 insertions(+)
 create mode 100644 jupiter.txt saturn.txt
```

Note that the addition will *stage* the files but not commit them into the repository. Only after we have entered the commit command they will be put in the repository.

Note also that you can *unstage* added files

Note also that a commit is used with a comment line. Use this because it will help you track what has been changed.

Now we will change the contents of one file, let's say we modify jupiter.txt. Use some editor to do this (not a word processor of course). Enter again the git status command.

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   jupiter.txt
#
no changes added to commit (use "git add" and/or "git commit -a")

$git diff
diff --git a/jupiter.txt b/jupiter.txt
index df0654a..315bf3a 100644
--- a/jupiter.txt
+++ b/jupiter.txt
@@ -1 +1,2 @@
 +some new added text here
```

This is a bit cryptic but is actually a difference between two versions with a unique identifier (df0654a and 315bf3a).

Now we stage and commit the change. We can do this also in one command line

```
$ git commit -a -m"added text to jupiter"
[master 34961b1] added text to jupiter
 1 file changed, 1 insertion(+)
```

2.5 Moving back to earlier versions

Suppose we change again jupiter.txt but we accidently delete a large part of our text (or code) and want to move back to the earlier version. We can do this as follows:

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   jupiter.txt
#
no changes added to commit (use "git add" and/or "git commit -a")

$ git checkout HEAD jupiter.txt
(or we could also move to some specific version:)
$ git checkout f22b25e jupiter.txt
```

As you see in the picture below git uses a stack of versions with unique identifiers. You can find these identifiers with the **'git log'** command. This will return all versions of all files. As you see above the f22b25e is only a small part of the version tag, you do not need to use the huge hexadecimal commit string. Git log has additional features:

```
git log --pretty=oneline  This will give you a log of versions and
                          the commit message for each file
on one line

git log --pretty=format:"%h - %an, %ar:%s"  will give you a short version, name,
                                            how long ago and message.

The huge output of some logs can be shortened:
git log --since=2.weeks
git log --until="2017-01-15"
git log --until="2 years 1 day 3 minutes ago"
git log --after="2016-02-03"
git log --before="2016-02-04"
git log --author="mr hacker"
git log --grep="some string in the message"
```

2.6 Ignore certain file types

In the programming world your directory will often clutter with all kind of generated files line object files (from compilers) .pyc (byte compiled python files) etc. You don't want these files in your repository and we do this by creating a .gitignore file

```
$ nano .gitignore
$ cat .gitignore
*.dat
*.pyc
results/

$git add .gitignore
$git commit "adding ignore file"
```

From now on a file with extension .dat or .pyc will not be added staged. You can override it with the -f extension (git add -f mydata.dat)

3

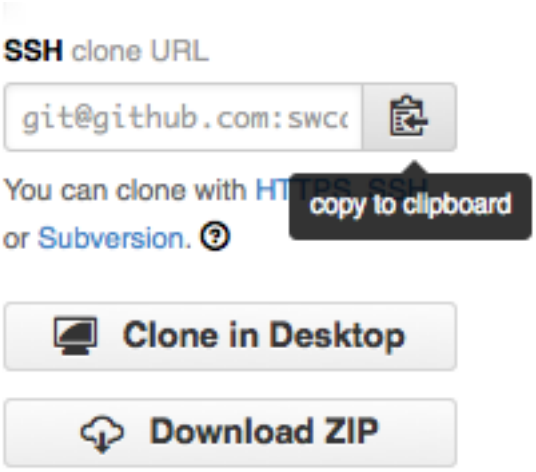
Using remote servers

We have now used a local repository, but you normally want to backup your work to a server. The distributed model of git is intended to have version management locally during development and incomplete code. You normally push to a server when you have tested your code in order to give a collaborating developer a working version of your code. Please note that you can have multiple servers added to your local repository. When we want to save our work to a remote location we use the *git push* command.

Remote (git) servers are often github.com and bitbucket.org where you can store code public or private. For confidential sourcecode it's of course recommended not to use such servers, but to use a trusted local server. At FNWI the FEIOG group can install a virtual linux system very cheap and help with installing the git server and apache webserver applications. You can also ask me for help.

On github create first the planets repository using your webbrowser. After creation you will see the commands you will need to type in order to push your local repository into the github.

Please note that github supports public repositories without payment, but for a personal storage which is not searchable you will need to pay. Bitbucket.org is a better choice because it allows personal repositories and gives more storage for universities, but there are also restrictions of course when you use only a free account. Also note that you need to store a ssh key in your profile to be able to push you work.



Copy that string from the browser and run this command:

```
$git remote add origin https://github.com/twalstra/planets
$git remote -v
origin   https://github.com/twalstra/planets.git (push)
origin   https://github.com/twalstra/planets.git (fetch)
$git push origin master
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 821 bytes, done.
Total 9 (delta 2), reused 0 (delta 0)
To https://github.com/twalstra/planets
 * [new branch]      master -> master
 Branch master set up to track remote branch master from origin.
```

We can also *pull* changes (from others) from the remote repository into our own repository to synchronize our work with others:

```
$ git pull origin master
From https://github.com/twalstra/planets
 * branch            master     -> FETCH_HEAD
 Already up-to-date.
```

Please note the two words at the end of the git pull: origin and master. These are not fixed but more or less standards in the git world. The first refers to the remote server the second to a branch which is called by default master. The master branch is like a root of a development tree. You can have several development branches (for example with different or new development directions of your project). We will see this in following chapters.

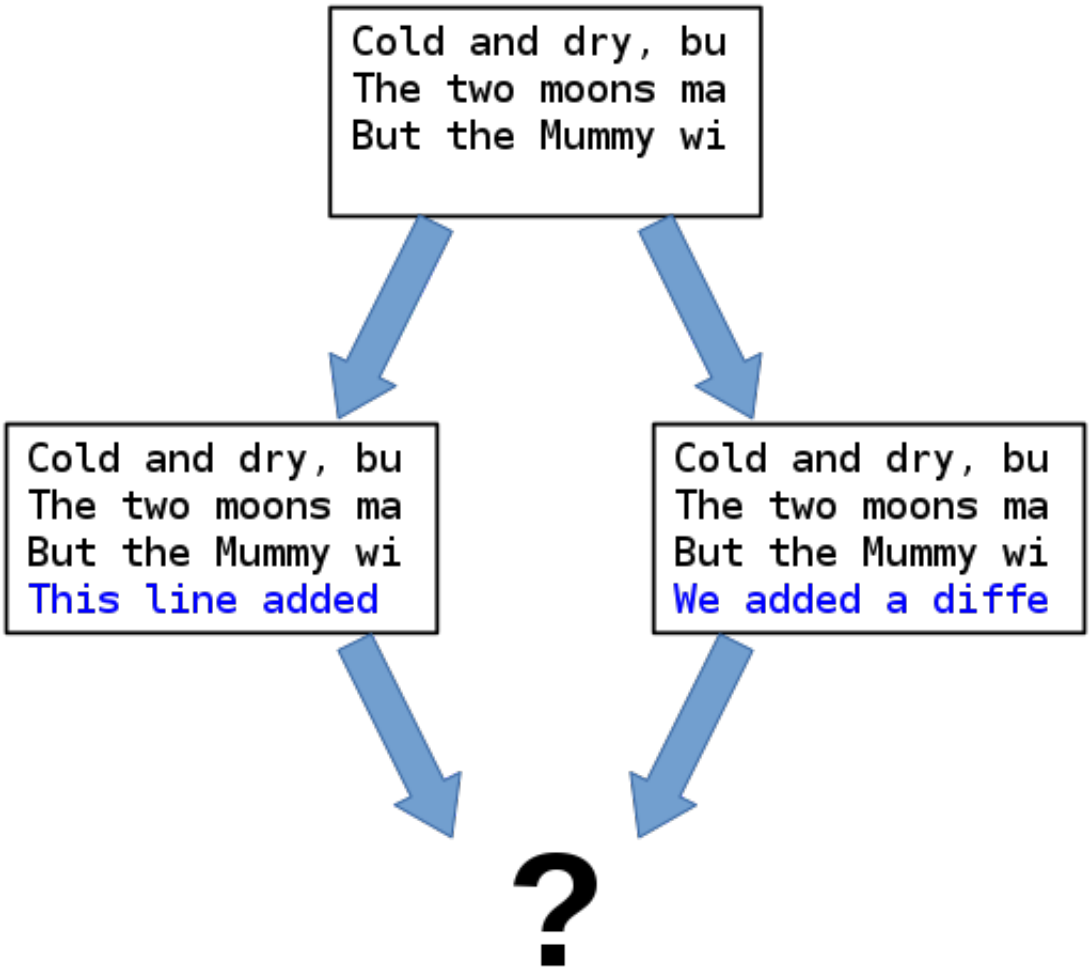
When you want to have a copy of your github repository on a different machine you do the following:

```
$git clone https://github.com/twalstra/planets
Cloning into 'planets'...
remote: counting objects: 7, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 7 (delta 0), resused 7 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.
checking connectivity... done.
```

4

Solving conflicts

As soon as people can work in parallel, someones going to step on someone elses toes. This will even happen with a single person: if we are working on a piece of software on both our laptop and a server in the lab, we could make different changes to each copy. Version control helps us manage these conflicts by giving us tools to resolve overlapping changes. Suppose 2 persons, let's name them John and Mariska, work on jupiter.txt and John has pushed his changes to the remote server. Mariska has not yet pulled his changes but extended jupiter.txt with a different text at exactly the same location as John has done. This will lead to problems when Mariska tries to push her changes to the remote server. The server will try to merge the changes but will detect a conflict which has to be resolved. See the picture below with an example we try to solve:




```
$ git pull origin master
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1)
Unpacking objects: 100% (3/3), done.
From https://github.com/twalstra/planets
 * branch            master     -> FETCH_HEAD
 Auto-merging mars.txt
 CONFLICT (content): Merge conflict in jupiter.txt
 Automatic merge failed; fix conflicts and then commit the result.
```

Now our jupiter.txt file has some additional lines which show the conflict:

```
$cat jupiter.txt

Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
<<<<<<< HEAD
We added a different line in the other copy
=======
This line added to Wolfman's copy
>>>>>>> dabb4c8c450e8475aee9b14b4383acc99f42af1d
```

Our change is in the "HEAD" the server version is below the dash line. Mariska now edits this file (remove the dash etc.) into something which should be right and where John hopefully agrees with. She stages again the file and commits into her local repo, and pushes it afterwards to the remote server. A pull afterwards take care that our local repo is in sync with the server. Suppose Mariska changes the jupiter file into the following:

```
$ cat jupiter.txt

Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
We removed the conflict on this line
```

```
$ git add jupiter.txt
$ git status

# On branch master
# All conflicts fixed but you are still merging.
#   (use "git commit" to conclude merge)
#
# Changes to be committed:
#
#   modified:   jupiter.txt
#
$ git commit -m "Merging changes from GitHub"
[master 2abf2b1] Merging changes from GitHub
$ git push origin master

Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 697 bytes, done.
Total 6 (delta 2), reused 0 (delta 0)
To https://github.com/twalstra/planets.git
   dabb4c8..2abf2b1  master -> master

$ git pull origin master

remote: Counting objects: 10, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 2), reused 6 (delta 2)
Unpacking objects: 100% (6/6), done.
From https://github.com/twalstra/planets
 * branch            master     -> FETCH_HEAD
 Updating dabb4c8..2abf2b1
 Fast-forward
  jupiter.txt | 2 +-
   1 file changed, 1 insertion(+), 1 deletion(-)
```

And now the file looks like:

```
$ cat jupiter.txt

Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
We removed the conflict on this line
```

5

Links

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git