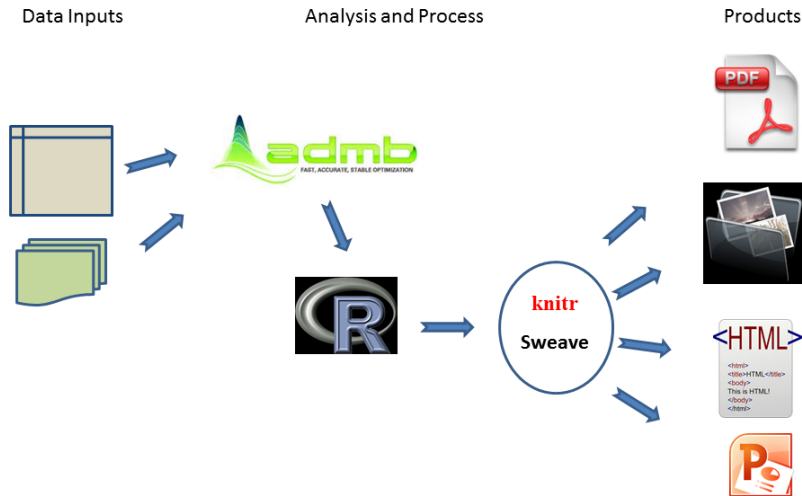


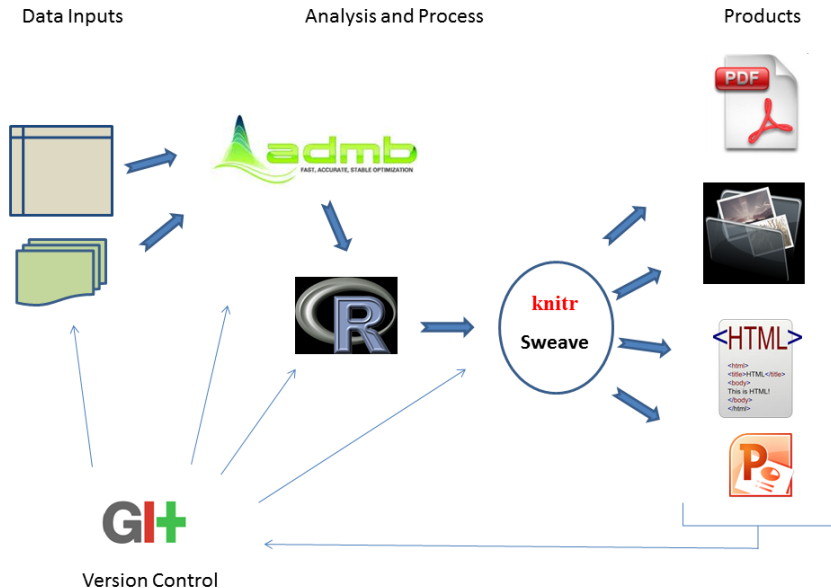
So What is Git??

December 11-12, 2013.

Our Alternative Work-flow



Our Alternative Work-flow



What is Git

- distributed version control software
 - ▶ no central server
 - ▶ each repository is complete and independent
- widely used in computer science and software engineering
- designed to support development of Linux Kernel
 - ▶ to be scalable
 - ▶ to facilitate collaborative coding
- 'save-as' on steroids

Version control allows you to

- easily back-up projects to a server and/or the web
- work effectively on multiple computers
- reset your directory to any previous state
- use branches to safely make changes that might break your code
- work collaboratively with other analysts
 - ▶ sharing whole project
 - ▶ easily integrate their changes or contributions

Version Control Basics

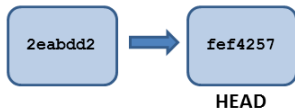
- 'repository' version control database
 - ▶ `.git` directory in project root
- a 'commit' is a snapshot that captures the state of selected files
- each commit has one or more parents
- git allows us to reset the directory to the state of any existing commit

Initialize Repository

First Commit



Second Commit



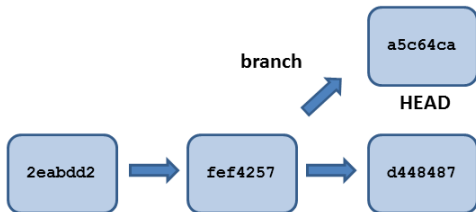
Third Commit



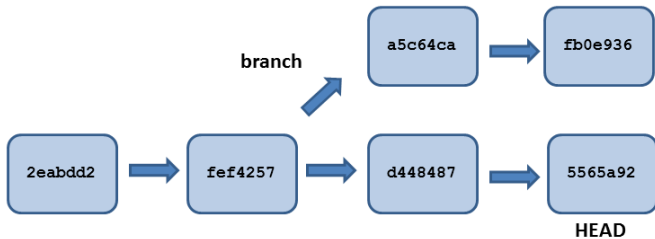
Checkout Commit



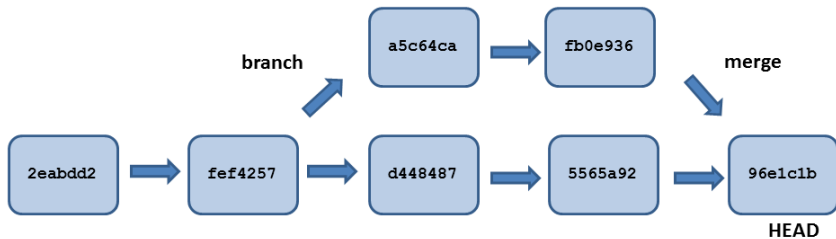
Branch



Checkout Branch



Merge



Setting up git

from a command prompt:

```
> git config --global user.name "<your name>"  
> git config --global user.email "you@there.com"
```

verify:

```
> git config --list
```

Your First Repository

- navigate to one of the example directories (e.g. - /examples/9_amdb_sweave)
- alternatively copy one to a convenient location

from a command prompt:

```
> cd <project root directory>
> git init
> dir /a
```


emacs and git: magit

- git command line tool
- several GUI's available
- most widely used plug-in for emacs is 'magit'

from emacs

- open any file the in project directory
- type C-c C-g to open *magit* buffer

What to commit - .gitignore

- only source files need to be checked into version control
 - ▶ .r, .rnw
 - ▶ .dat, .pin, .tpl
- by default, git will want to commit everything
 - ▶ .gitignore text file that contains file and directory names that should **NOT** be listed
- an example for admb projects found in
~/workshop/utils/.gitignore

Staging

- committing files in git is two step process
- first they need to be 'staged'
 - ▶ placed in the queue
- then they can be committed
 - ▶ during the commit, all staged files are added to the repository
- if you change file after it is staged, it needs to be 'unstaged' and staged again for changes to be reflected in commit

Staging files in *magit*:

- place your cursor beside each file and type s
- each file will move from Untracked Files to Staged Files

Your First Commit

- Commit

- ▶ if you're happy with the staged files type `c` to commit
- ▶ emacs will open a `*magit-edit-log*` buffer
- ▶ type your commit message and then type `C-c C-c` to finalize commit

Commit Messages

- each commit is accompanied by message
 - ▶ first line - treated as title
 - ▶ subsequent lines/paragraphs form body
 - ▶ good messages are succinct and to the point, but accurately capture differences between previous commit
 - ▶ git has a number of tools to search for commits based on message content

Your Second Commit

- make some changes to any of files included in the first commit.
- in the `*magit*` buffer type `g` to refresh it contents
- the files you changed should appear under Changes in the `*magit*` buffer
- to actually see the changes you just made, type `d` in the `*magit*` buffer followed by `<return>`

Git Diff

- tool to display line-by-line difference between commits
- by default it shows the difference between latest commit and current directory contents
- commit numbers and/or file names can be used as arguments
 - ▶ possible to create recipes or cookbooks
- parts of each changed file are shown for context
- new lines are green and prefixed with '+'
- removed line red and prefixed with '-'

Git Diff

```
emacs@OND4C00803339
File Edit Options Buffers Tools Magit YASnippet Help

@@ -241,8 +235,8 @@ PARAMETER_SECTION
  init_bounded_number lnM(-5,5,1)

// LOG-SCALE DEVIATIONS BETWEEN OBS AND EFFECTIVE EFFORT
- init_bounded_vector effort_devsT(fyear+1,lyear,-1.,1.,3)
- init_bounded_vector effort_devsG(fyear+1,lyear,-1.,1.,3)
+ init_bounded_vector effort_devsT(fyear+1,lyear,-1.,1.,4)
+ init_bounded_vector effort_devsG(fyear+1,lyear,-1.,1.,4)

// log-scale catchability
  init_bounded_number log_qT(-20,20,1)
@@ -261,18 +255,18 @@ PARAMETER_SECTION
  // init_bounded_number logselT_p4(-3,3,2) // slope second curve trap

//double logistic parameters of size (ie length)
- init_bounded_number logselG_p1(5.8,6.5,2) // inflection for first curve - gill
+ init_bounded_number logselG_p1(5.8,6.5,3) // inflection for first curve - gill
  init_bounded_number logselG_p2(-4,4,3) // slope first curve - gill
  init_bounded_number logselG_p3(6.2,6.7,-3) // inflection for second curve - gill
  init_bounded_number logselG_p4(-4,4,-3) // slope second curve gill
- init_bounded_number logselT_p1(5.8,6.5,2) // inflection for first curve - trap
+ init_bounded_number logselT_p1(5.8,6.5,3) // inflection for first curve - trap
  init_bounded_number logselT_p2(-4,4,3) // slope first curve - trap
  init_bounded_number logselT_p3(6.2,6.7,-3) // inflection for second curve - trap
  init_bounded_number logselT_p4(-10,4,-3) // slope second curve trap

// parameters for random walk for p1
- init_bounded_vector logdevG_p1(fyear+1,lyear,-1,1,3);
- init_bounded_vector logdevT_p1(fyear+1,lyear,-1,1,3);
+ init_bounded_vector logdevG_p1(fyear+1,lyear,-1,1,4);
+ init_bounded_vector logdevT_p1(fyear+1,lyear,-1,1,4);

//// the survey selectivity will be modelled as a normal curve
// init_bounded_number log_s1_sel_p1(-5,5,2); //mean of selectivity curve
@@ -287,7 +281,7 @@ PARAMETER_SECTION

// LOG-SCALE DEVIATIONS FROM OVERALL SCALE
// FIRST "years" VALUES ARE RECRUITS, REST INIT POP
1\%- *magit-diff* 77% (267,0) (Magit Diff WS yas)
```

Your Second Commit (cont'd)

- if you are happy with status of files
- stage each of the files as before
- type `c` to open the commit buffer
- provide a brief commit message and finalize the commit by typing `C-c C-c`
- the `*magit*` buffer will be reset with a new commit hash

What is a hash?

- git uses 'hashes' to track commits
- a hash is generated by an algorithm run on the content of the commit
- hashes are unique to the commit (1.2×10^{24})
- small changes in content result in wildly different hashes - probability of collisions extremely small.
- why hashes?
 - ▶ distributed nature means that git can't use sequential commit numbers

What is a hash? (cont'd)

- R can generate hashes using the digest library.

For example try:

```
> library(digest)

> digest('QFC_workshop', algo='sha1')

> digest('QFC workshop', algo='sha1')
```

Reviewing Previous Commits - Git Log

- git log provides a history of changes that lead to current state
- multiple options to control output and format

from a command prompt in your working directory try:

```
> git log  
> git log --oneline
```

or equivalently in emacs with magit

- C-c C-g l L
- C-c C-g l l

When to commit

- commit early and often
- especially if tests pass or model converges
- immediately before reporting

Reverting to a Previous Commit

- best done from command line
- don't forget the '.'

Basic Syntax:

```
git checkout <hash> .
```

Creating Branches

- easy to create branches
- git documentation encourages branch creation
- branches allow for independent parallel development without disrupting existing code
- Create a branch if you are:
 - ▶ making changes that might break something
 - ▶ fixing a bug
 - ▶ adding a feature

command line:

```
> git branch <branch name>
```

in `*magit*`:

```
b c
```

Switching between branches

- switching branches moves HEAD
- resets working directory to match state associated with commit

command line:

```
> git checkout <branch name>
```

```
in *magit*:
```

```
b b
```

Merging

- merging is complementary to branching
- merge commits have more than one parent
- changes in each branch are 'automagically' integrated by git
- merge conflict only occur if same lines changed in both commits

Merging

- first checkout your main branch
- then merge target branch with main

```
> git checkout master  
> git merge <branchname>
```


Merge Conflicts

- occur when the same line has been changed in both branches

Merge Conflict:

```
<<<<<< HEAD:sweave\_admb.rnw
\bibliographystyle{cjfas}
=====
\bibliographystyle{plain}
>>>>>> testbranch:sweave\_admb.rnw
```

Dealing with merge conflicts:

- manually edit files
 - ▶ replace code between lines delineated by <<<<<< and >>>>>> with code that will be committed
- individually stage (add) resolved files
- commit to finish merge

Tags

- like bookmarks for commits
- git supports two type of tags:
 - ▶ lightweight
 - ▶ annotated (recommended)
- must be manually pushed to remote repositories
 - ▶ `> git push usb --tags`
- generally associated with significant commits
 - ▶ software release
 - ▶ manuscript submissions
 - ▶ final reports ('Quota Setting 2013').

Remote Repositories

- remote repositories often original source of code
- also serve as backup and mobile repositories

create a remote repository:

```
> dir F:
> mkdir gitrepos
> cd gitrepos
> git init --bare
> cd <your original repo>
> git add remote usb F:/gitrepos
> git remote -v
```

Pushing and Pulling to Remote Repositories

Synchronize Remote:

of the form: `git push <remote> <branch>`

```
> git push usb --all
```

Synchronize Local:

```
> git fetch usb
```

```
> git merge usb master
```

OR

```
> git pull usb master
```

Remote Repository on the go

- example of workflow using usb repository and two disconnected computers

Clone Existing Repository

- cloning a repository give you an exact copy of an existing repository
- clone from websites such as bitbucket or github
- or from other sources such as usb, ftp site or cloud service
- cloned repository will automatically have remote
 - ▶ named 'origin' by convention

Example:

```
> git clone https://github.com/AdamCottrill/QFC_Workshop.git
```

Hooks

- files that run on when specific actions occur
- git has numerous hooks available
- need to be manually activated in each repository
 - ▶ not created during clone

Make your research reproducible

- hooks are used write a file that contains commit number:
 - ▶ on each commit
 - ▶ on each merge
 - ▶ on each checkout
- L^AT_EX package gitinfo to integrate commit number into all pdf reports

changes to *.rnw

```
\usepackage{gitinfo}           % in preamble
...
Commit Number: \gitAbbrevHash  % somewhere in document
```


Hooks

post-checkout hook

```
prefixes="."
echo $GIT_DIR
for pref in $prefixes
do
  git log -1 --date=short \
    --pretty=format:"\usepackage[%
      shash={%h},
      lhash={%H},
      authname={%an},
      authemail={%ae},
      authsdate={%ad},
      authidate={%ai},
    ...
      commdate={%at},
      refnames={%d}
    ]{gitsetinfo}" HEAD > $pref/gitHeadInfo.gin
done
```

Hooks

results in gitHeadInfo.gin:

```
\usepackage[%  
    shash={dabb2eb},  
    lhash={dabb2eb433a5d14bc45a8dae8aad7f43208d990},  
    authname={Adam Cottrill},  
    authemail={adam.cottrill@ontario.ca},  
    authsdate={2013-10-07},  
    authidate={2013-10-07 10:52:12 -0400},  
    authudate={1381157532},  
    commname={Adam Cottrill},  
    commemail={adam.cottrill@ontario.ca},  
    commsdate={2013-10-07},  
    commidate={2013-10-07 10:52:12 -0400},  
    commudate={1381157532},  
    refnames={ (HEAD, master)}  
{gitsetinfo}
```

Git - Gotchas

- pdf reports must be generated **after** committing working directory in order for commit number to reflect actual state of directory
- be careful with dropbox
 - ▶ don't use dropbox as working directory with git
 - ▶ dropbox folders are great as remote repositories

Recap

- git distributed version control system
- designed for collaborative use
- magit emacs plugin for git
- hooks extend functionality

Further Reading and References

- Software Carpentry
 - ▶ <http://software-carpentry.org/>
- Pro Git:
 - ▶ <http://git-scm.com/book>
- excellent introductory book:
 - ▶ Version Control with Git
- An introductory talk by the author of Pro Git:
 - ▶ <http://www.youtube.com/watch?v=ZDR433b0HJY>
- An excellent intermediate talk about git:
 - ▶ <http://www.youtube.com/watch?v=ig5E8CcdM9g>
- git and github youtube channel:
 - ▶ <https://www.youtube.com/user/GitHubGuides/videos>
- A recent blog post about using magit:
 - ▶ [~/introduction-magit-emacs-mode-git/](#)