

# Version Control with Git



Ben R. Fitzpatrick

PhD Candidate, Statistical Science, Mathematical Sciences School, Queensland University of Technology

0000-0003-1916-0939

[github.com/brfitzpatrick/](https://github.com/brfitzpatrick/)

@benrfitzpatrick

# Download Git for your Operating System

<http://git-scm.com/downloads>



# git

--fast-version-control

**About**

**Documentation**

**Blog**

**Downloads**

GUI Clients

Logos

**Community**

## Downloads



Mac OS X



Windows



Linux



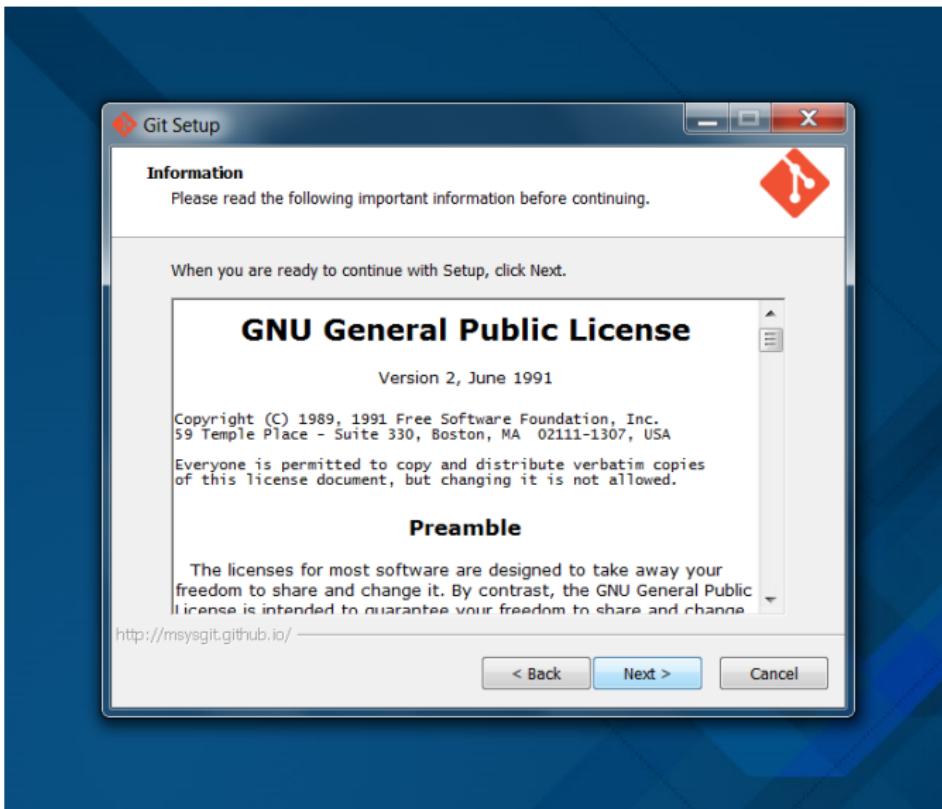
Solaris

Older releases are available and the [Git source repository](#) is on GitHub.

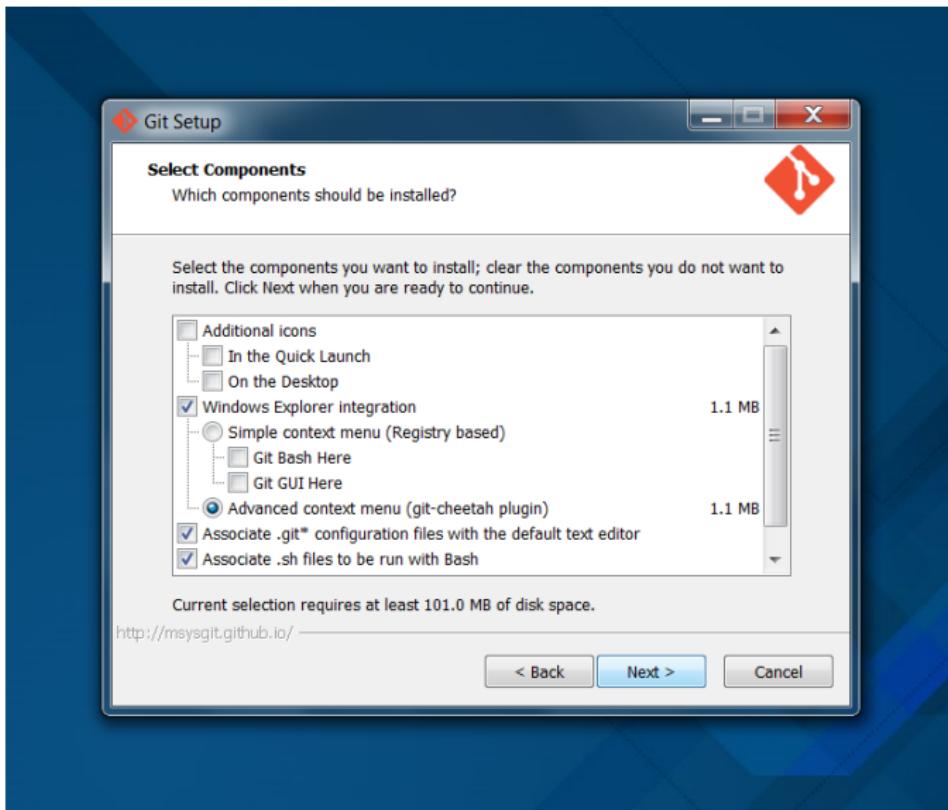
# Installing Git on MS Windows: Step 1



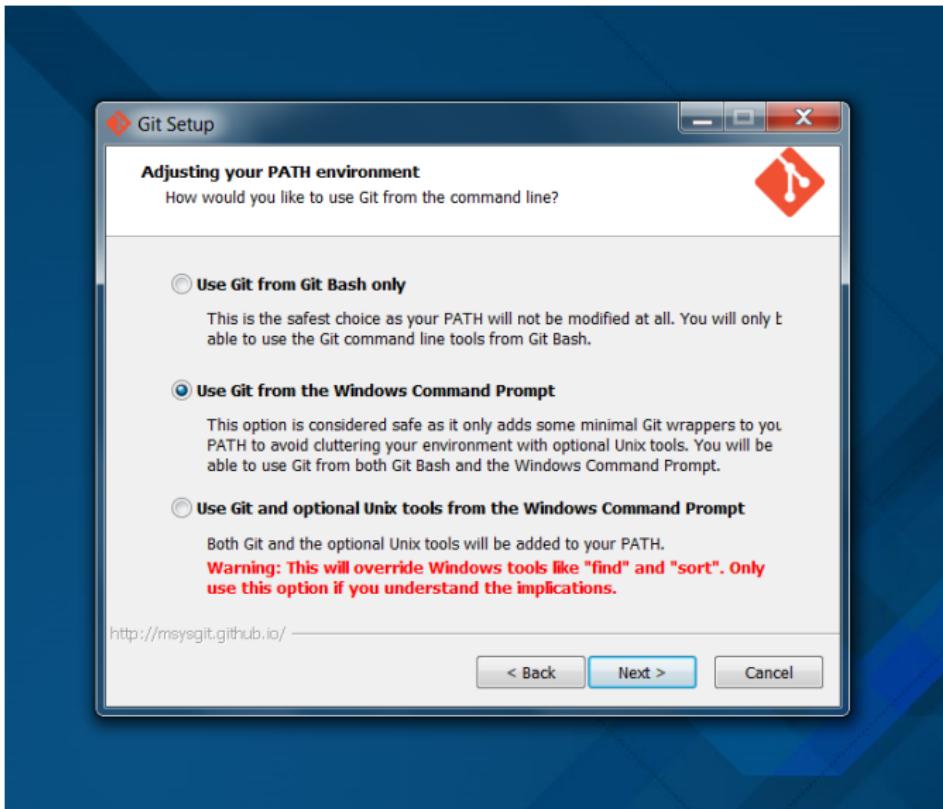
# Installing Git on MS Windows: Step 2



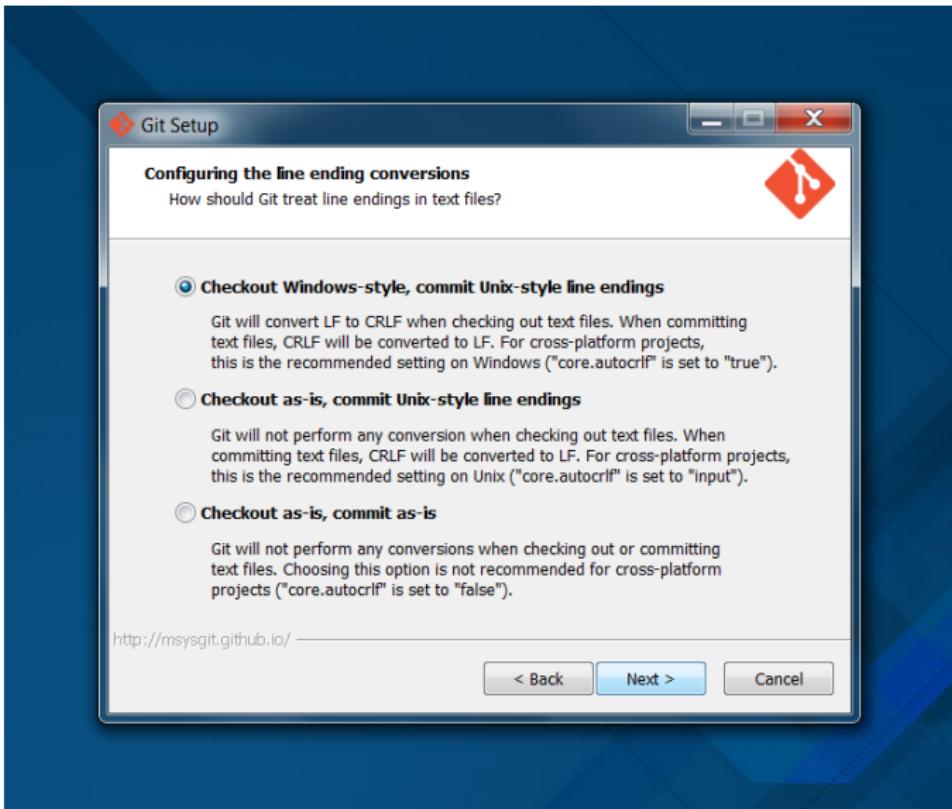
# Installing Git on MS Windows: Step 3



# Installing Git on MS Windows: Step 4

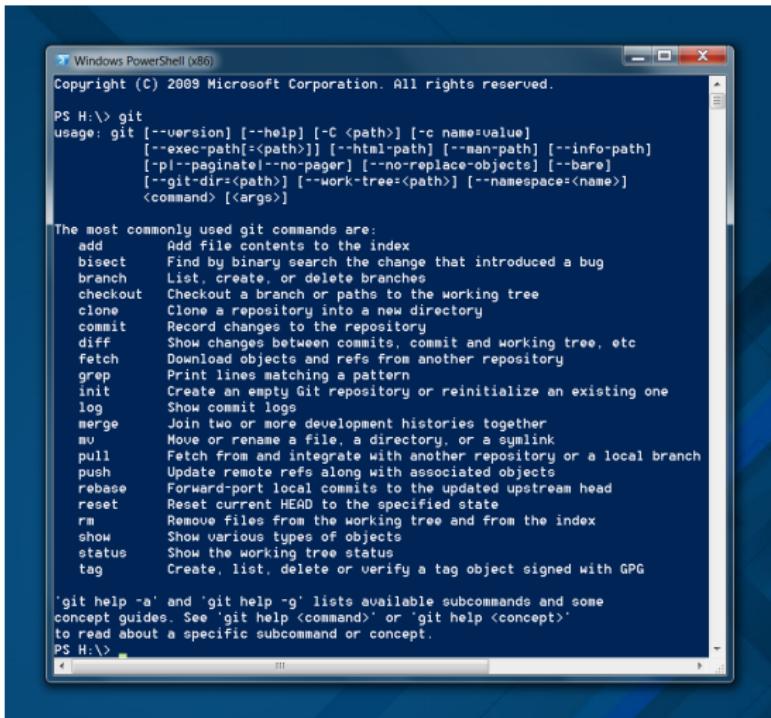


# Installing Git on MS Windows: Step 5



# Check Git is installed on MS Windows

Open a PowerShell and type *git* then press 'Enter' and you should see output like that below:



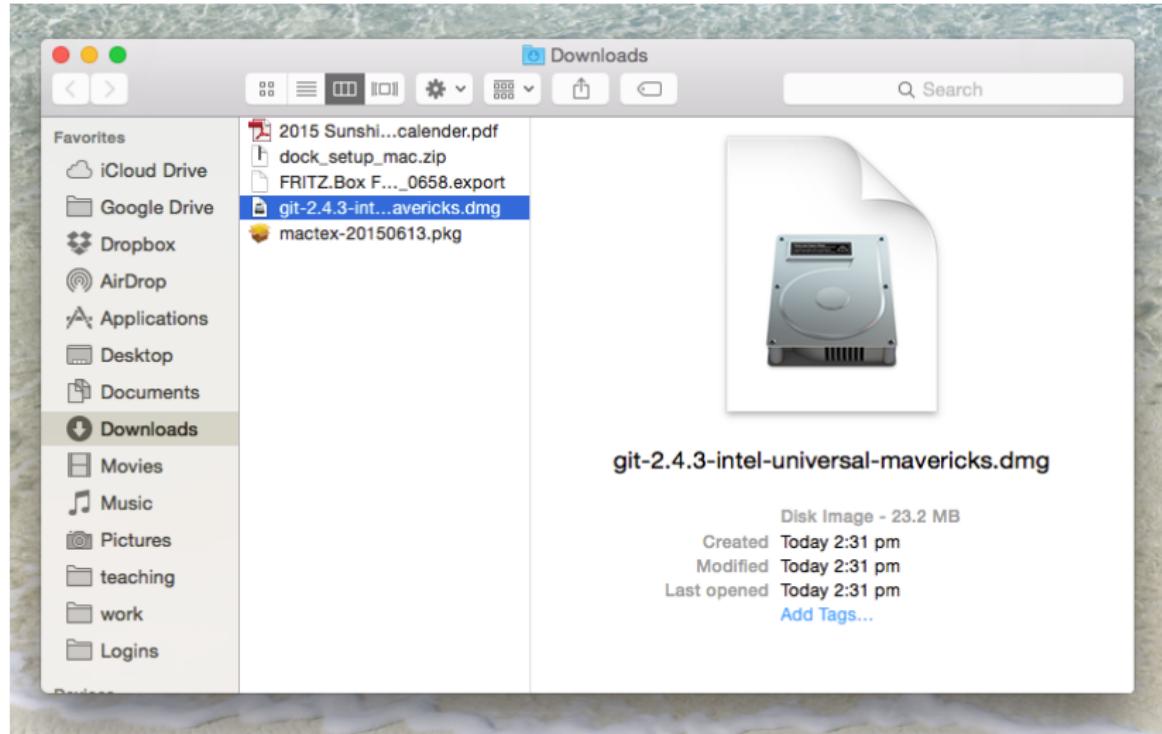
```
Windows PowerShell (x86)
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS H:\> git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[:<path>]] [--html-path] [--man-path] [--info-path]
           [-p|--paginate|--no-pager] [-n--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

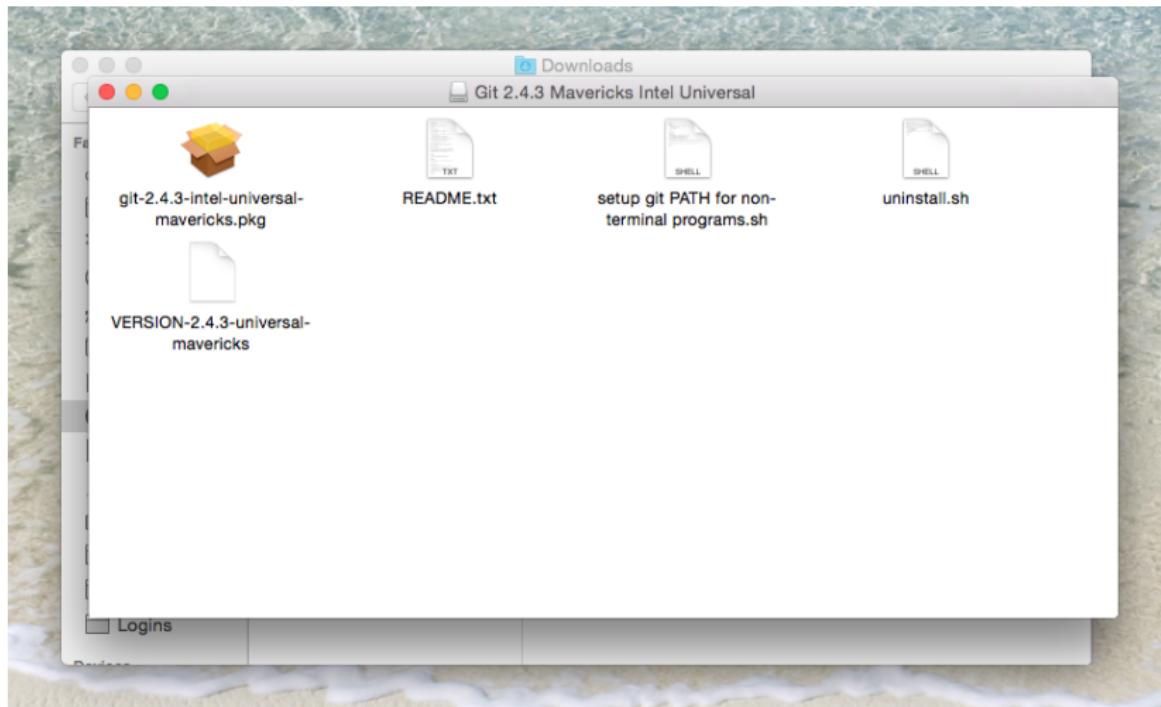
The most commonly used git commands are:
  add          Add file contents to the index
  bisect       Find by binary search the change that introduced a bug
  branch      List, create, or delete branches
  checkout    Checkout a branch or paths to the working tree
  clone       Clone a repository into 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' lists available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
PS H:\>
```

# Installing Git on MacsOS: Step 1



# Installing Git on MacsOS: Step 2



# Installing Git on MacsOS: Step 3



**"git-2.4.3-intel-universal-mavericks.pkg"**  
can't be opened because it is from an  
unidentified developer.

Your security preferences allow installation of only  
apps from the Mac App Store and identified  
developers.

"git-2.4.3-intel-universal-mavericks.pkg" is on the  
disk image "git-2.4.3-intel-universal-mavericks.dmg".  
Google Chrome downloaded this disk image today at  
2:31 pm from sourceforge.net.

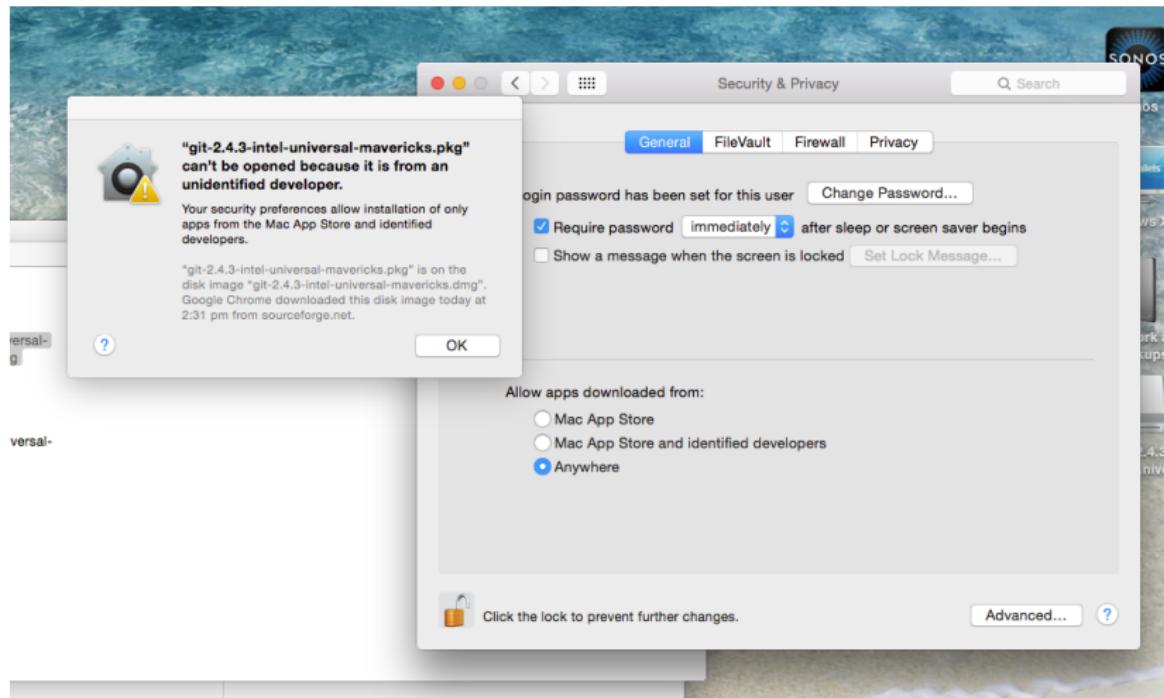


OK

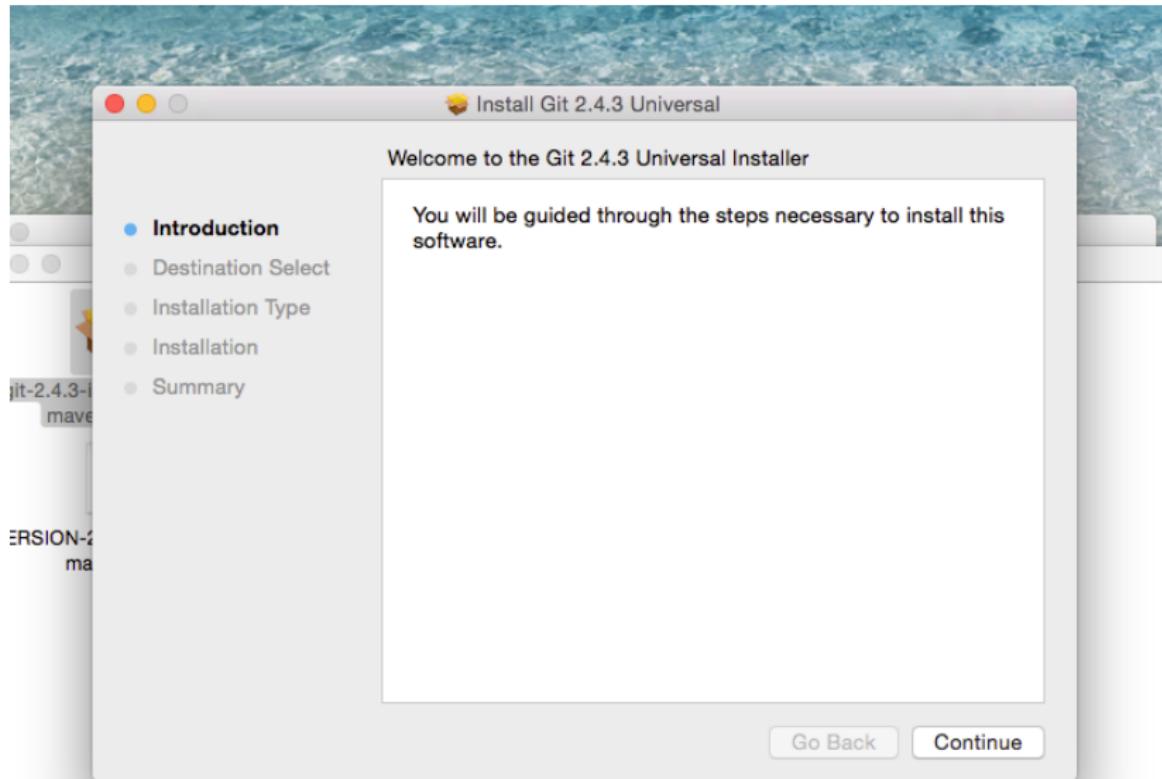
# Installing Git on MacsOS: Step 4

You may need to relax your security setting briefly

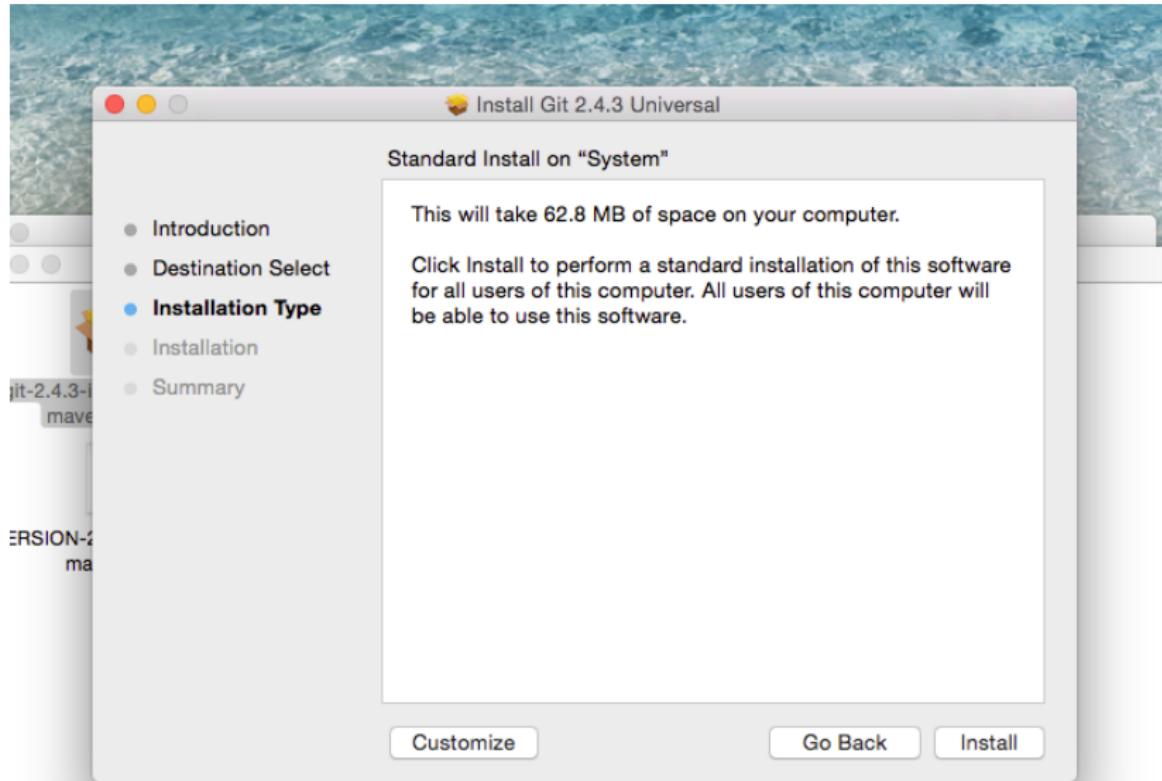
If this makes you uncomfortable you're welcome to watch the session rather than participating



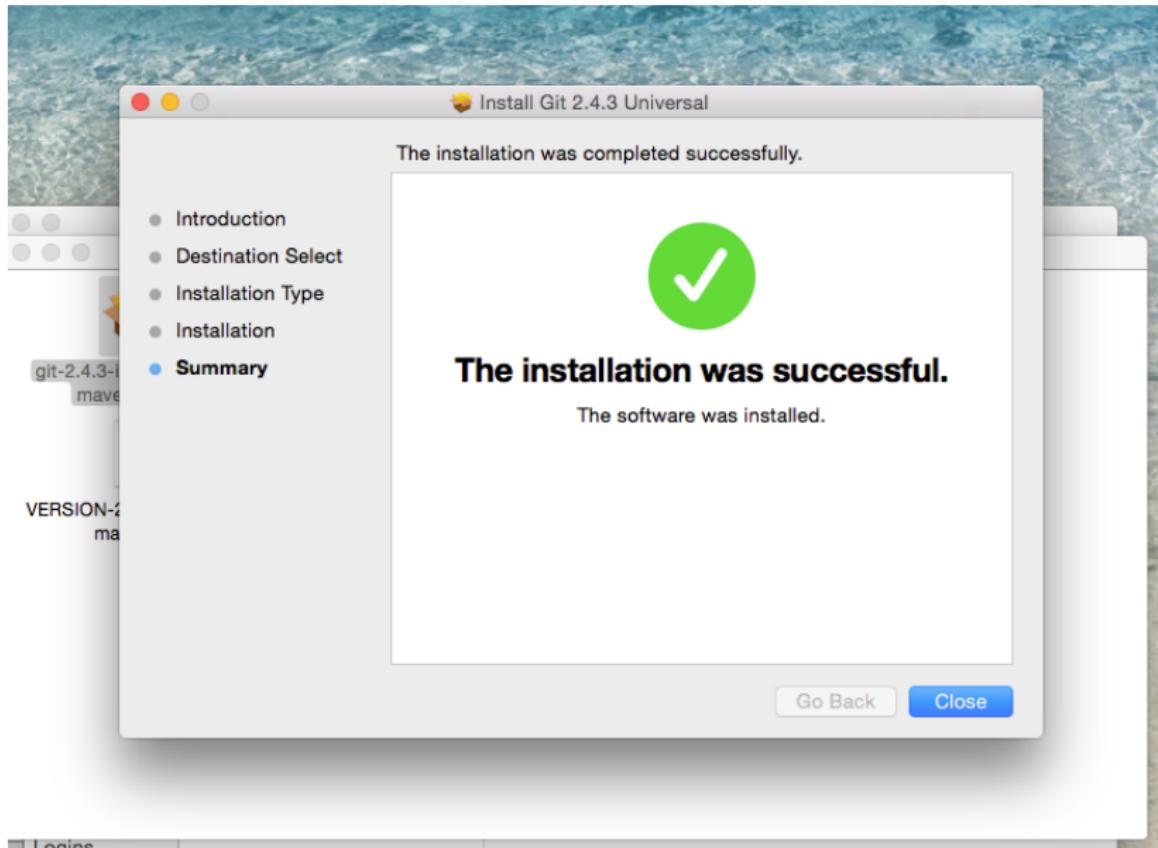
# Installing Git on MacsOS: Step 5



# Installing Git on MacsOS: Step 6



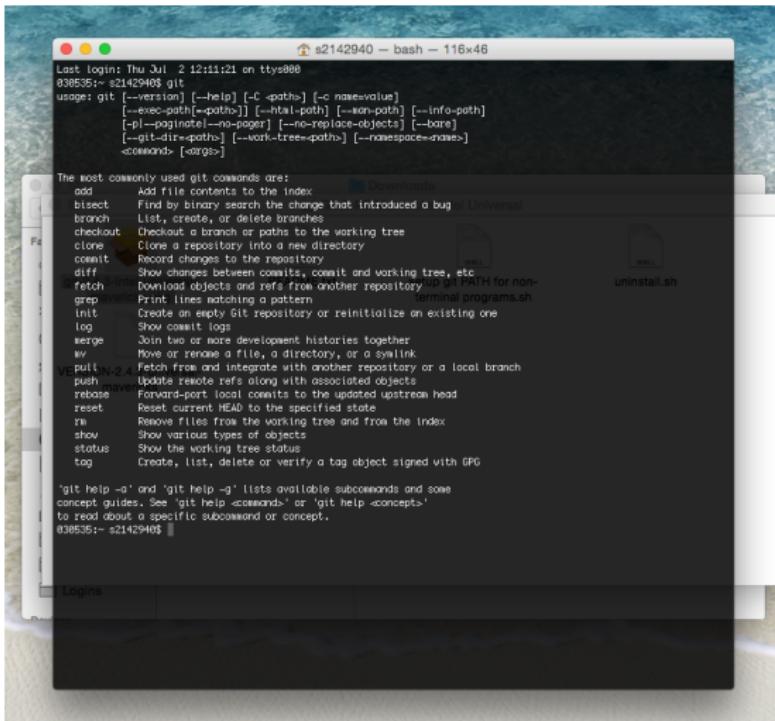
# Installing Git on MacsOS: Step 7



# Installing Git on MacsOS: Step 8

Check Git is installed:

Open a terminal, type *git* and press ‘Return’ and you should see output like that below:



The screenshot shows a terminal window titled "s2142940 — bash — 116x46". The command "git" was entered at the prompt, resulting in the following output:

```
Last login: Thu Jul  2 12:11:21 on ttys000
038535:~ s2142940$ git
usage: git [<-version> [<-help>] [<-c <path>] [<-c name=value>
           [<-e exec-path> [<path>]] [<-html-path>] [<-xon-prompt>] [<-info-path>
           [<-p1>--paginate=<n>]<-no-pager>] [<-no-replace-objects>] [<-bare>]
           [<-git-dir=<path>] [<-work-tree=<path>] [<namespace=<name>>
           [<command> [<args>]]]

The most commonly used git commands are:
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' lists available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
038535:~ s2142940$
```

# Version Control Software

Have you ever put numbers or dates at the end of a file name to keep different versions of it?

- If yes then you have already used version control!

Have you ever used track changes and comments to take turns editing a MS Word document with one or more others?

- If yes then you have collaboratively edited a file!

Version Control Software:

- formalizes these concepts
- can do a lot of the organisational work for you
- makes reverting to previous versions without losing work easy

# Benefits of Using Version Control Software

## Version Control for Working Solo

- Easy to revert to previous versions of a file without losing any of the subsequent revisions
- Simple to create multiple versions of a file and edit them in parallel with the option combine them again at a later date

## Version Control for Collaborating

- Greatly simplifies your life
- Different contributors can edit different versions of the same files concurrently
- Tools exist to systematically combine versions of the same file from different contributors
- time stamped record of all changes from all contributors - if someone breaks something you can revert to when it last worked and work forward to discover the bug

# Why Git?

- Git is free and open source
- Git is available for most operating systems
- Git is Distributed and Fast
- Two major code hosting services (GitHub & BitBucket - both with free plans) support version control with Git

# Git is Distributed Version Control

Distributed Version Control means that:

- Every contributor has a copy of the entire database up to when they copied it (all revisions of all files)
- two or more contributors can work on the files in their copy of the database independently then combine their modifications systematically at a later date

Results is a fast and flexible version control system:

- contributors can work independently offline connecting to the server sporadically to submit their changes
- most operations are performed locally (on individual contributors' computers) so less waiting for the server

# Why Git on the Command Line?

- Makes discrete components of the workflows obvious.
- This in turn will enable you to migrate easily to one of the many GUIs at a later date should you so desire.
- The workflows we will practise today function identically at MS Windows, MacOS and GNU+Linux command line interfaces.

# Plan for this Module

## Part 1

### Version Control for Working Alone

- ① Set up Git on your laptop to communicate with the GitHub servers
- ② Create a local **clone** of a **remote repository**
- ③ Make some changes to the files in your local copy of the repository
- ④ **commit** these changes to your copy of the history of revisions of these files
- ⑤ **push** these changes to your local copy of the repository to the remote copy on the GitHub servers
- ⑥ make some more changes
- ⑦ revert to a previous version of a file in this repository safely by **checking out** an previous **commit** to a new **branch** of the revision history of that file

# Plan for this Module

## Part 2

### Collaborating with Git & GitHub

- ① fork the repository of a fellow course attendee
- ② clone your fork of their repository creating a local copy on your hard drive
- ③ edit one of their files in your local copy of their repository
- ④ commit your changes to your local copy of their repository
- ⑤ submit a pull request - requesting the repository owner merge your changes with their branch of the repository
- ⑥ watch the merge process on your fellow course attendee's computer
- ⑦ switch roles and repeat

# Creating a Repository on the GitHub Servers

## Log into the GitHub Web Service & Create a New Repository

The screenshot shows a GitHub user profile for Ben R. Fitzpatrick. At the top, there's a search bar and navigation links for Pull requests, Issues, and Gist. Below the search bar, there's a large profile picture of a person's face, followed by the user's name and handle: Ben R. Fitzpatrick and brfitzpatrick. To the right of the profile picture, there's a 'Repositories' section with a 'New' button highlighted by a green circle and labeled '1.'. Further down, there's a repository named 'fork\_for\_ben' with a green circle around its 'New' button and a large green number '2.' next to it. The 'fork\_for\_ben' repository has a description: 'Materials for an Introductory Short Course on the R Language and Environment for Statistical Computing and Graphics.', updated 2 hours ago, and 1 star. Below this, there's another repository named 'fork\_this\_repo' with a description: 'practise collaborating with GitHub', updated 4 days ago, and 0 stars.

Search GitHub

Pull requests Issues Gist

Contribution Repositories Public activity

Edit profile

New

Find a repository Search All Public Private Sources Forks Mirrors

Intro to R

Materials for an Introductory Short Course on the R Language and Environment for Statistical Computing and Graphics.

Updated 2 hours ago

R ★ 1 ▾ 0

1.

fork\_for\_ben

Forked from sk8aus/fork\_for\_ben

Updated 2 days ago

R ★ 0 ▾ 2

fork\_this\_repo

practise collaborating with GitHub

Updated 4 days ago

Followers 0 Starred 0 Following 0

# Creating a Repository on the GitHub Servers

## Complete the Details

Search GitHub Pull requests Issues Gist

Owner Repository name  
 brfitzpatrick / my\_new\_repo ✓

Great repository names are short and memorable. Need inspiration? How about [scaling-octo-ironman](#).

Description (optional)

 Public Anyone can see this repository. You choose who can commit.

 Private You choose who can see and commit to this repository.

Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** Add a license: **GNU General Public License v3.0** ⓘ

**Create repository**

# Creating a Repository on the GitHub Servers

## View your New Repository

The screenshot shows a GitHub repository page for the user 'brfritzpatrick' named 'my\_new\_repo'. The top navigation bar includes links for 'This repository', 'Search', 'Pull requests', 'Issues', and 'Gist'. The repository name 'brfritzpatrick / my\_new\_repo' is displayed prominently. On the right side, there are buttons for 'Unwatch', 'Star', 'Fork', and a dropdown menu. Below the repository name, there are fields for 'Description' (containing 'Short description of this repository') and 'Website' (containing 'Website for this repository (optional)'). A 'Save' button and a 'Cancel' link are also present. To the right of these fields, there are summary statistics: 1 commit, 1 branch, 0 releases, and 1 contributor. The main content area shows the repository's history with one commit from 'brfritzpatrick' authored just now. It also lists files like 'LICENSE' and 'README.md' with their respective commit details. On the far right, there is a sidebar with links for 'Code', 'Issues' (0), 'Pull requests' (0), 'Wiki', 'Pulse', 'Graphs', and 'Settings'. At the bottom, there is an 'SSH clone URL' field containing 'git@github.com:brf...'. It also states that you can clone with 'HTTPS, SSH, or Subversion' and provides a 'Download ZIP' button.

This repository

Search

Pull requests Issues Gist

brfritzpatrick / my\_new\_repo

Unwatch 1 Star 0 Fork 0

Description

Short description of this repository

Website

Website for this repository (optional)

Save or Cancel

1 commit 1 branch 0 releases 1 contributor

branch: master my\_new\_repo / +

Initial commit

brfritzpatrick authored just now latest commit 4930beee35

LICENSE Initial commit just now

README.md Initial commit just now

README.md

my\_new\_repo

Code

Issues 0

Pull requests 0

Wiki

Pulse

Graphs

Settings

SSH clone URL

git@github.com:brf...

You can clone with HTTPS, SSH, or Subversion.

Download ZIP

# Cloning your new Repository to your Hard Drive

First, create a folder on your Hard Drive in which to store your Git Repositories.

We are now going to use the Git command line application to 'clone' your new repository from the GitHub servers to your hard drive.

Git uses distributed version control so a Git 'clone' is a complete copy of the entity of the repository i.e.

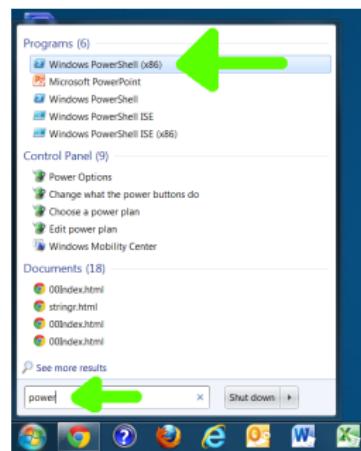
- all the files
- the entire history of snapshots of files states created each time you commit modifications to the repository

Being a distributed version control system Git doesn't require you to be connected to the GitHub servers to work on your files and commit them to your local clone of the repository.

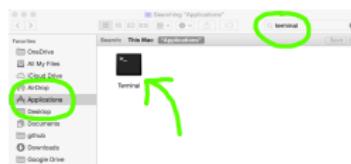
# Git on the Command Line

Accessing a command line interface in your OS of choice

MS Windows  
Open a 'PowerShell'



Mac OS X & later  
Open a 'Terminal'

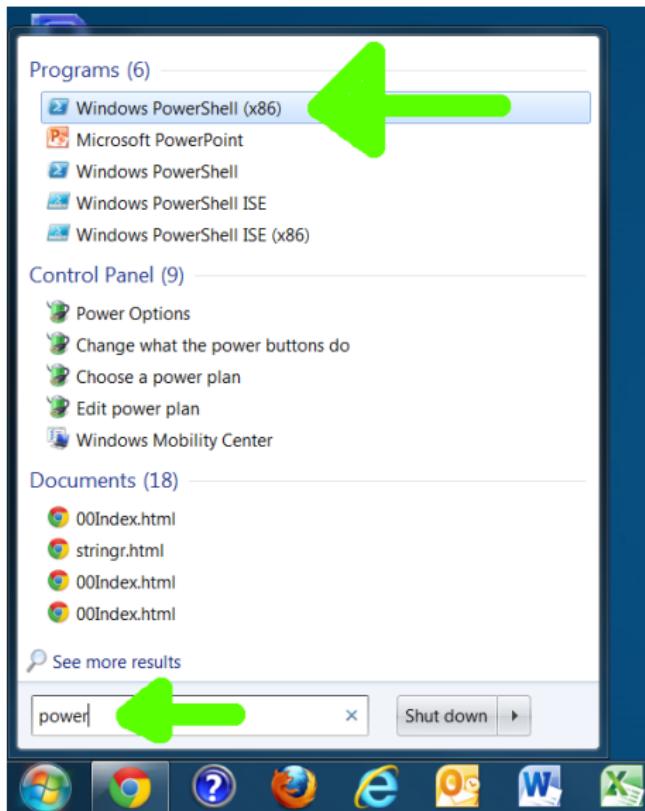


GNU+Linux  
Open a 'Terminal'



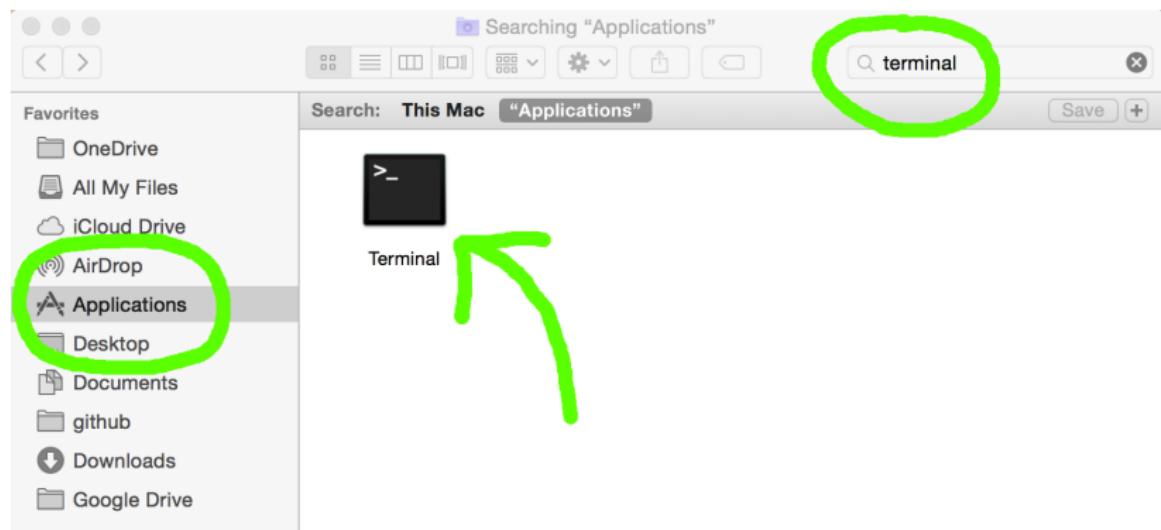
# Accessing a command line interface in MS Windows

Open a 'PowerShell'



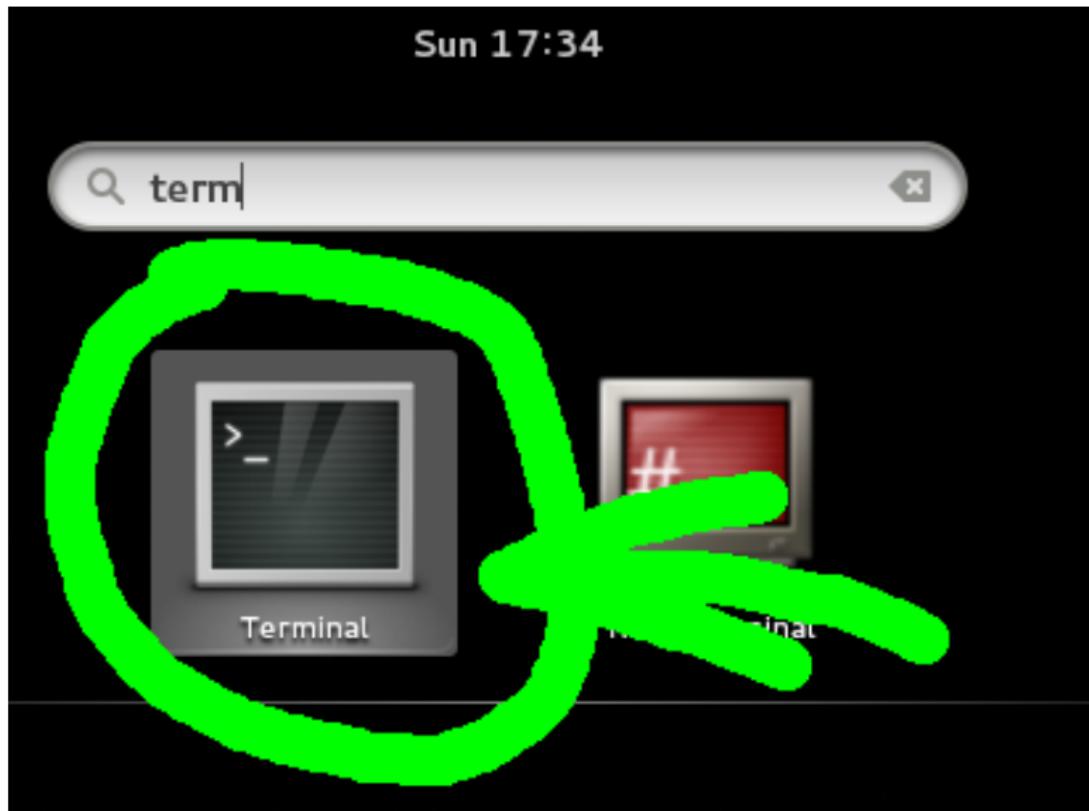
# Accessing a command line interface in MacOS

Open a 'Terminal'



# Accessing a command line interface from GNU+Linux

Open a 'Terminal'



# Configuring Git for the first time

## All Users

```
> git config --global user.name "Your Name"  
> git config --global user.email you@mail.com
```

# Configuring Git for the first time

## Choosing a Global Editor

The global editor is used to write your commit messages.

Default is Vim but Vim is a little heavy on keyboard short cuts for some...

Other Options:

- Notepad (available on all Windows PCs)
- Atom: <https://atom.io/docs/v1.0.0/getting-started-installing-atom>
- Sublime: <http://www.sublimetext.com/2>
- TextMate: <http://macromates.com/download>
- Emacs:  
`sudo apt-get install emacs`

# Configuring Git for the first time

<https://help.github.com/articles/associating-text-editors-with-git/>

## Set a Text Editor of your choice as the global editor

You'll need relevant the text editor installed for the below to work:

```
# Notepad:  
> git config --global core.editor notepad.exe  
# Atom:  
> git config --global core.editor "atom --wait"  
# Sublime:  
> git config --global core.editor "subl -n -w"  
# TextMate:  
> git config --global core.editor "mate -w"  
# Emacs:  
> git config --global core.editor "emacs"
```

# Cloning Your Repository from the GitHub Servers

Set the working directory to the location on your hard drive to which you would like the repository cloned then list the files and folders present at that location:

## Powershell Users

```
> C:  
> cd C:\Users\username\Documents\GitHub_Repos\  
> dir
```

## Terminal Users

```
> cd /home/username/Documents/GitHub_Repos  
> ls
```

# Copy HTTPS URL to use when cloning repository

The screenshot shows a GitHub repository page for 'brfitzpatrick / my\_new\_repo'. The page includes fields for 'Description' and 'Website', and displays metrics like 1 commit, 1 branch, 0 releases, and 1 contributor. A green arrow points from the bottom right towards the 'HTTPS clone URL' field, which contains the URL `https://github.com/brfitzpatrick/my_new_repo`. There is also a 'Copy to clipboard' button next to the URL.

This repository

Search

Pull requests Issues Gist

Unwatch 1 Star 0 Fork 0

brfitzpatrick / my\_new\_repo

Description Website

Short description of this repository Website for this repository (optional)

Save or Cancel

1 commit 1 branch 0 releases 1 contributor

branch: master → my\_new\_repo / +

Initial commit

brfitzpatrick authored 33 minutes ago latest commit 4930beee35

LICENSE Initial commit 33 minutes ago

README.md Initial commit 33 minutes ago

README.md

my\_new\_repo

Code

Issues 0

Pull requests 0

Viki

Pulse

Graphs

Settings

HTTPS clone URL

`https://github.com/brfitzpatrick/my_new_repo`

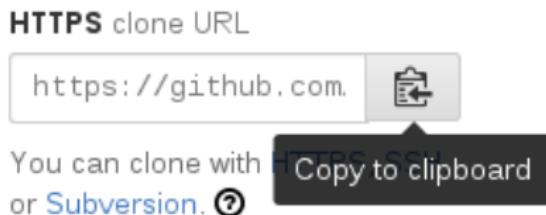
You can clone with Subversion.

Copy to clipboard

Download ZIP

# Clone your Repository: GitHub Servers → your Hard Drive

Use the HTTPS URL you copied earlier:



## All Users

```
> git clone https://github.com/.../my_new_repo  
 .git  
> cd ./my_new_repo/  
> ls  
> git status
```

# Git

## The Four States of Files in a Directory Git is Tracking

Files in a directory which you are version controlling with Git can exist in one of four states:

- **committed** data stored in the database for the associated repository
- **modified** local copy of a file (in your working directory) is different to the most recent version of that file stored in the database for the associated repository i.e. you have changed something but not committed the changes (yet)
- **staged** the local copy of a file which you have modified is marked to be added to the database in the next commit snapshot you send to the database
- **untracked** Git is not recording revisions to this file

- ① you add files to or modify existing files in your local copy of the repository
- ② you stage these modified files to be committed to the database for the repository
- ③ you perform a commit which takes the files as they were when staged and stores a snapshot of them in the database for the repository
- ④ you push the changes recorded in this commit to your remote copy of this repository on the GitHub servers

# Git

## Adding New Files

- ① Open your favourite program for authoring R code (e.g. RStudio)
- ② Create an new R script (it can be very short)
- ③ Save your R Script in the local version of your repository

# Git

## Adding New Files to the List of Tracked Files

```
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what
  will be committed)
```

We have to tell Git that we want it to track the changes we make to this new file

# Git

## Adding New Files to the List of Tracked Files

We have to tell Git that we want it to track the changes we make to this new file

```
> git add filename.R
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

new file:   filename.R
```

We can still change our mind about adding the file to the repository at this stage quite easily.

Once you're ready to 'commit' this change to the permanent record of changes in this repository we make a 'commit'

```
> git commit  
[master 311acfb] Initial Commit of some R code.  
1 files changed, 0 insertions(+), 0 deletions  
(-) create mode 100644 filename.R
```

You will be prompted to write a short description of the changes you are committing - these become invaluable as your project grows and if you are collaborating.

Once you've written a commit message, save it and close the text editor you used to write it.

You should see confirmation of the commit immediately below the git commit line.

# Git

Now that you've committed some changes to your local clone of the repository this local version will be ahead of the version on the GitHub Server

```
> git status
On branch master
Your branch is ahead of 'origin/master' by 1
commit.
(use "git push" to publish your local commits)
```

Seeing as we are all currently connected to the internet, it's a good time to 'push' these changes to GitHub server ('origin/master' in the default setup) to bring the remote copy of this repository up to date.

# Git

## Pushing changes to local repository to remote repository

```
> git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 353 bytes | 0 bytes/s
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:brfitzpatrick/my_new_repo.git
  4930bee..99d61f1  master -> master
```

If you return to the page for your repository on the GitHub website and refresh the page you should see your most recent commit listed on the repository page.

# Git

## Pushing changes to local repository to remote repository

Furthermore, now that we have ‘pushed’ our most recent changes our local copy of the repository to the remote copy of the repository on the GitHub servers these two versions of the repository will be identical

```
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

# Undoing Things Safely

Checking out Previous Commits to a New Branch

To revert to a previous version of a file that you have a 'snapshot' of from a previous 'commit'

Click on the file name on the GitHub website and click the 'History' button:

Provided we have committed all changes to the file then it is safe to use the checkout command which will overwrite the file in the current local working directory with the file from the previous 'commit'.

You can think of commits a bit like save points, if we have two we can move between them but if we load an old save without first making a current save we will lose the unsaved changes

List existing branches of current repository:

```
git branch
```

then, choose a commit you'd like to go back to, copy the corresponding SHA from the history tab on the website and checkout this previous commit to a new branch called 'testing', here abbreviated to (*SHA* = 58...a44):

```
> git checkout -b testing 58...a44
Switched to a new branch 'testing'
> git branch
  master
* testing
```

# Git

## Undoing Things Safely

Now if you open your .R file you'll see that you have a previous version of the file without the new changes.

Next up we'll make some changes to the version of this file on the branch 'testing' so that it will diverge from the version on the branch 'master'.

Once again check you're on the branch called 'testing'

```
> git branch
```

then make some changes on testing branch by editing the version of the file currently in your working directory then save those changes, add them, commit them and push them 'upsteam' to server:

```
> git push --set-upstream origin testing
```

- Now if you check the website your repository should now have two branches.
- We also know that each branch contains a slightly different file now.
- Next we'll 'merge' your changes from 'testing' back into the 'master' branch.

To merge into a branch we must be on that branch:

```
> git checkout master
```

Now we will attempt to merge 'testing1' into current branch  
(master)

```
> git merge testing1
Auto-merging filename.R
CONFLICT (content): Merge conflict in
filename.R
Automatic merge failed;
fix conflicts and then commit the result.
```

# Resolving a Merge Conflict with a Merge Tool

```
git mergetool
```

This message is displayed because 'merge.tool' is not configured.

See 'git mergetool --tool-help' or 'git help config' for more details.

'git mergetool' will now attempt to use one of the following tools:

```
meld opendiff kdiff3 tkdiff xxdiff
```

...

```
emerge vimdiff
```

Merging:

```
filename.R
```

# Resolving a Merge Conflict with a Merge Tool

```
Normal merge conflict for 'filename.R':  
{local}: modified file  
{remote}: modified file  
Hit return to start merge resolution tool  
(meld):
```

# Resolving a Merge Conflict with a Merge Tool

Demonstrate **Meld**

# Collaborating

Forking a repository is creating a branch from a repository you don't own.

# Create a Repository, Add Some Files

- Log into your Github account
- Click the Repositories Tab
- Click the 'New' button (it's green)
- name your repository
- make it public (we can do a private one later)
- initialize repository with a README
- add a license GPL or MIT are easy FOSS licenses
- clone the repository to your hard drive
- add a file to your local clone of the repository
- commit it to the repository
- push your changes to the remote repository

## Fork a Friends Repository, Add Some Files

- search for a friend's github repository
- on their repository page click fork
- clone your fork of their repository to your hard drive
- edit one of their files
- push your changes to their repo

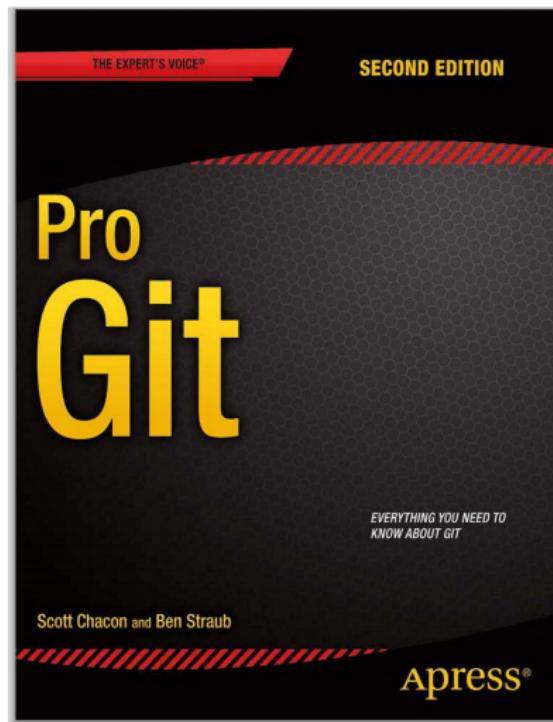
# Accepting and rejecting Pull Requests

## Merging your friends' branches

- Create a new branch to merge your friend's changes into
- switch to that branch
- merge their changes with **Meld**

# Recommended Further Reading

<http://git-scm.com/book/en/v2>



# Recommended Further Reading:

<https://help.github.com/>

**GitHub Help** Version ▾ Contact Support Return to GitHub



Set Up Git      Create A Repo      Fork A Repo      Be Social

Sometimes you just need a little **help**.

How can we help?

### Common Issues

- › Why are my contributions not showing up on my profile?
- › Why is Git always asking for my password?
- › Dealing with non-fast-forward errors
- › Error: Repository not found
- › Do you have custom plans?
- › HTTPS cloning errors
- › What is my disk quota?
- › What are the limits for viewing content and diffs in my repository?
- › Remove sensitive data
- › How do I access my organization account?

# Image Credits

- Git Logo by Jason Long  
<http://git-scm.com/downloads/logos>
- GitHub Logo, [www.github.com](http://www.github.com)
- R Logo, [www.r-project.org](http://www.r-project.org)