Parallel & Distibuted Computing: Lecture 3

Alberto Paoluzzi

March 13, 2017

Version Control System (VCS), Julia packages

Git

Version Control System

- Git is a version control system (VCS) for tracking changes in computer files and coordinating work on those files among multiple people.

Version Control System

- Git is a version control system (VCS) for tracking changes in computer files and coordinating work on those files among multiple people.
- It is primarily used for software development, but it can be used to keep track of changes in any (set of) files.

Version Control System

- Git is a version control system (VCS) for tracking changes in computer files and coordinating work on those files among multiple people.
- It is primarily used for software development, but it can be used to keep track of changes in any (set of) files.
- Git was created by Linus Torvalds in 2005 for development of the Linux kernel

Data flows and storage levels

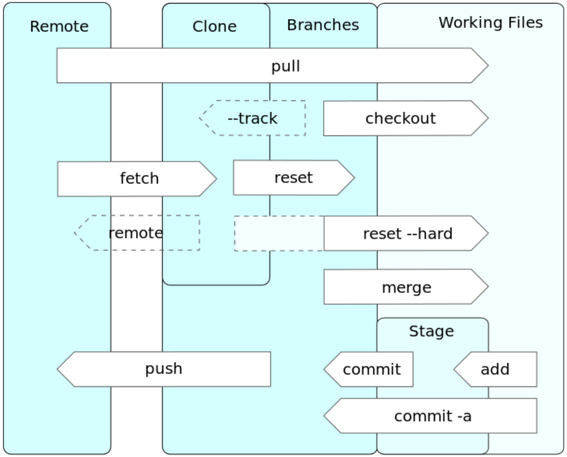
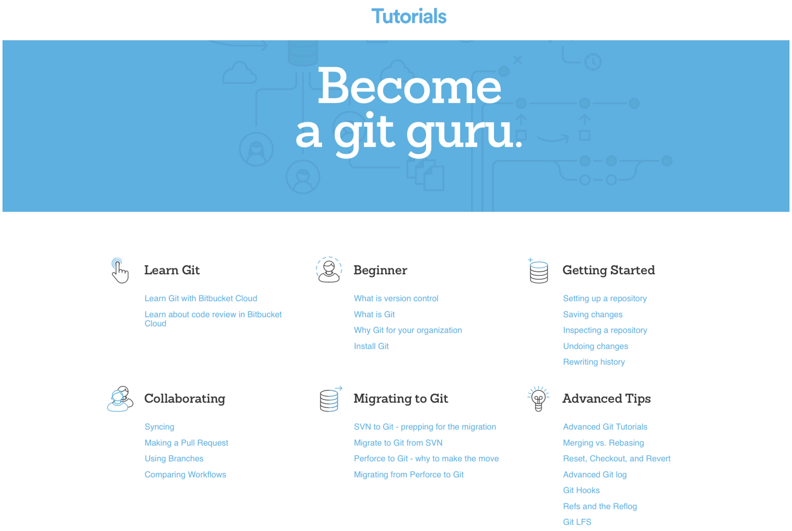


Figure 1: Some data flows and storage levels in the Git revision control system

Tutorials



Tutorials

Become
a git guru.

Learn Git

- Learn Git with Bitbucket Cloud
- Learn about code review in Bitbucket Cloud

Beginner

- What is version control
- What is Git
- Why Git for your organization
- Install Git

Getting Started

- Setting up a repository
- Saving changes
- Inspecting a repository
- Undoing changes
- Rewriting history

Collaborating

- Syncing
- Making a Pull Request
- Using Branches
- Comparing Workflows

Migrating to Git

- SVN to Git - prepping for the migration
- Migrate to Git from SVN
- Perforce to Git - why to make the move
- Migrating from Perforce to Git

Advanced Tips

- Advanced Git Tutorials
- Merging vs. Rebasing
- Reset, Checkout, and Revert
- Advanced Git log
- Git Hooks
- Refs and the Reflog
- Git LFS

Git characteristics

Strong support for non-linear development Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history.

Git characteristics

Strong support for non-linear development Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history.

- In Git, a core assumption is that a change will be merged more often than it is written, as it is passed around to various reviewers.

Git characteristics

Strong support for non-linear development Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history.

- In Git, a core assumption is that a change will be merged more often than it is written, as it is passed around to various reviewers.
- In Git, branches are very lightweight: a branch is only a reference to one commit.

Git characteristics

Strong support for non-linear development Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history.

- In Git, a core assumption is that a change will be merged more often than it is written, as it is passed around to various reviewers.
- In Git, branches are very lightweight: a branch is only a reference to one commit.

Distributed development Git gives each developer a local copy of the full development history, and changes are copied from one such repository to another.

Git characteristics

Strong support for non-linear development Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history.

- In Git, a core assumption is that a change will be merged more often than it is written, as it is passed around to various reviewers.
- In Git, branches are very lightweight: a branch is only a reference to one commit.

Distributed development Git gives each developer a local copy of the full development history, and changes are copied from one such repository to another.

- These changes are imported as added development branches, and can be merged in the same way as a locally developed branch.

Git characteristics

Compatibility with existant systems and protocols Repositories can be published via Hypertext Transfer Protocol (HTTP), or a Git protocol over Secure Shell (ssh).

Git characteristics

Compatibility with existant systems and protocols Repositories can be published via Hypertext Transfer Protocol (HTTP), or a Git protocol over Secure Shell (ssh).

- Subversion and svk repositories can be used directly with git-svn.

Git characteristics

Compatibility with existant systems and protocols Repositories can be published via Hypertext Transfer Protocol (HTTP), or a Git protocol over Secure Shell (ssh).

- Subversion and svk repositories can be used directly with git-svn.

Efficient handling of large projects Torvalds has described Git as being very fast and scalable

Git characteristics

Cryptographic authentication of history The Git history is stored in such a way that the ID of a particular version (a commit in Git terms) depends upon the complete development history leading up to that commit.

Git characteristics

Cryptographic authentication of history The Git history is stored in such a way that the ID of a particular version (a commit in Git terms) depends upon the complete development history leading up to that commit.

- Once it is published, it is not possible to change the old versions without it being noticed.

Git characteristics

Cryptographic authentication of history The Git history is stored in such a way that the ID of a particular version (a commit in Git terms) depends upon the complete development history leading up to that commit.

- Once it is published, it is not possible to change the old versions without it being noticed.

Pluggable merge strategies As part of its toolkit design, Git has a well-defined model of an incomplete merge.

Git characteristics

Cryptographic authentication of history The Git history is stored in such a way that the ID of a particular version (a commit in Git terms) depends upon the complete development history leading up to that commit.

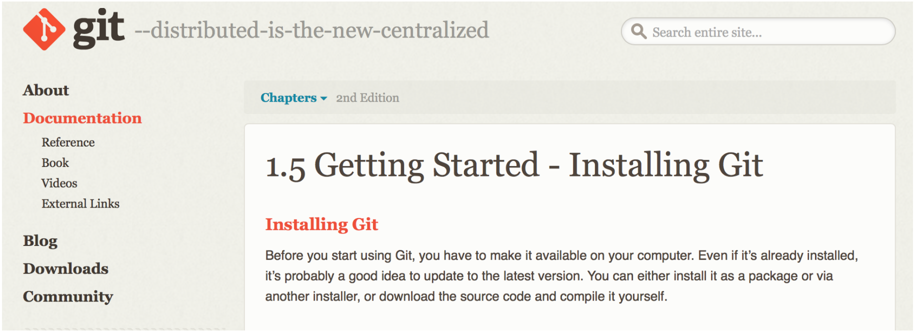
- Once it is published, it is not possible to change the old versions without it being noticed.

Pluggable merge strategies As part of its toolkit design, Git has a well-defined model of an incomplete merge.

- It has multiple algorithms for completing it, until telling that manual editing is needed.

Git Install

Getting Started - Installing Git



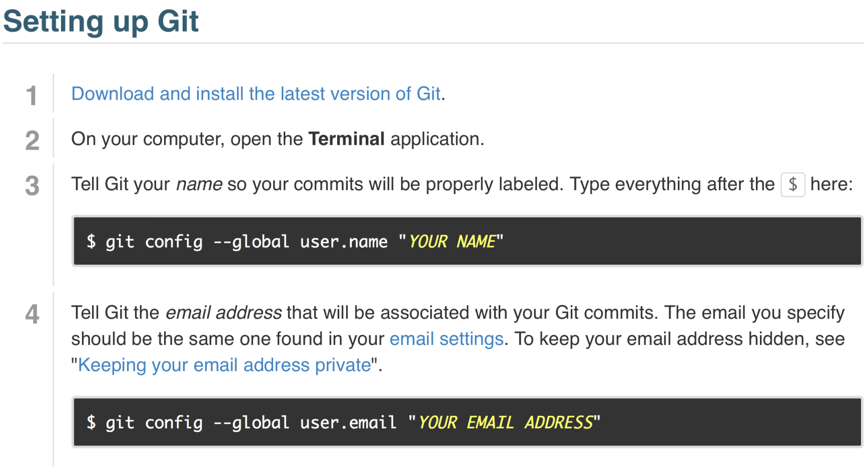
1.5 Getting Started - Installing Git

Installing Git

Before you start using Git, you have to make it available on your computer. Even if it's already installed, it's probably a good idea to update to the latest version. You can either install it as a package or via another installer, or download the source code and compile it yourself.

Figure 2: Git Install

Git configuring



Setting up Git

1 Download and install the latest version of Git.

2 On your computer, open the **Terminal** application.

3 Tell Git your *name* so your commits will be properly labeled. Type everything after the $ here:

```
$ git config --global user.name "YOUR NAME"
```

4 Tell Git the *email address* that will be associated with your Git commits. The email you specify should be the same one found in your email settings. To keep your email address hidden, see "Keeping your email address private".

```
$ git config --global user.email "YOUR EMAIL ADDRESS"
```

GitHub

Web-based Git = VCS + Internet hosting service

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

GitHub provides:

- access control

Williams, Alex (9 July 2012). GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz. Tech Crunch.

Web-based Git = VCS + Internet hosting service

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

GitHub provides:

- access control
- and several collaboration features:

Williams, Alex (9 July 2012). GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz. Tech Crunch.

Web-based Git = VCS + Internet hosting service

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

GitHub provides:

- access control
- and several collaboration features:
  - bug tracking,

Williams, Alex (9 July 2012). GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz. Tech Crunch.

Web-based Git = VCS + Internet hosting service

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

GitHub provides:

- access control
- and several collaboration features:
    - bug tracking,
    - feature requests,

Williams, Alex (9 July 2012). GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz. Tech Crunch.

Web-based Git = VCS + Internet hosting service

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

GitHub provides:

- access control
- and several collaboration features:
  - bug tracking,
  - feature requests,
  - task management,

Williams, Alex (9 July 2012). GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz. Tech Crunch.

Web-based Git = VCS + Internet hosting service

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

GitHub provides:

- access control
- and several collaboration features:
  - bug tracking,
  - feature requests,
  - task management,
  - and wikis for every project.

Williams, Alex (9 July 2012). GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz. Tech Crunch.

Hello World project on GitHub

You'll learn how to:

- Create and use a repository

Hello World project on GitHub

You'll learn how to:

- Create and use a repository
- Start and manage a new branch

Hello World project on GitHub

You'll learn how to:

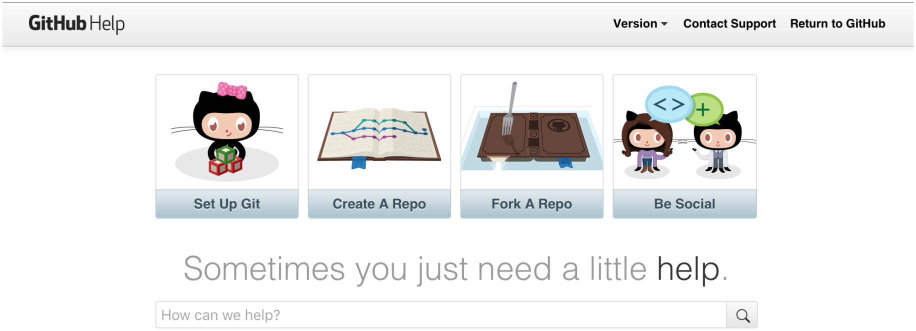
- Create and use a repository
- Start and manage a new branch
- Make changes to a file and push them to GitHub as commits

Hello World project on GitHub

You'll learn how to:

- Create and use a repository
- Start and manage a new branch
- Make changes to a file and push them to GitHub as commits
- Open and merge a pull request

GitHub help system



Sometimes you just need a little help.

Julia packages

Julia package listing

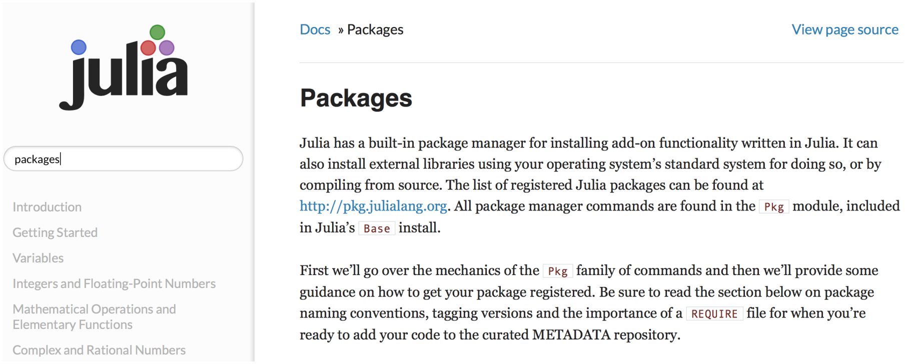
https://pkg.julialang.org

Listing all 1297 registered packages for the Julia programming language.

Last updated 2017-03-12 — Package ecosystem pulse

Packages tested on Julia versions:
v0.4.7 (previous release) — **v0.5.1 (current release)** — v0.6-pre (unstable)

Packages


julia
packages
Introduction
Getting Started
Variables
Integers and Floating-Point Numbers
Mathematical Operations and Elementary Functions
Complex and Rational Numbers
Docs » Packages
View page source

Packages

Julia has a built-in package manager for installing add-on functionality written in Julia. It can also install external libraries using your operating system's standard system for doing so, or by compiling from source. The list of registered Julia packages can be found at http://pkg.julialang.org. All package manager commands are found in the `Pkg` module, included in Julia's `Base` install.

First we'll go over the mechanics of the `Pkg` family of commands and then we'll provide some guidance on how to get your package registered. Be sure to read the section below on package naming conventions, tagging versions and the importance of a `REQUIRE` file for when you're ready to add your code to the curated METADATA repository.

Some `Pkg` commands

Package Manager Functions

Your package requirements are in the file `~/.julia/v0.4/REQUIRE`

`Pkg.status()` prints out a summary of the state of packages you have installed.

Some `Pkg` commands

Package Manager Functions

Your package requirements are in the file `~/.julia/v0.4/REQUIRE`

`Pkg.status()` prints out a summary of the state of packages you have installed.

`Pkg.add(pkg, vers...)` Add a requirement entry for pkg to Pkg.dir("REQUIRE") and call Pkg.resolve().

Some `Pkg` commands

Package Manager Functions

Your package requirements are in the file `~/.julia/v0.4/REQUIRE`

`Pkg.status()` prints out a summary of the state of packages you have installed.

`Pkg.add(pkg, vers...)` Add a requirement entry for pkg to Pkg.dir("REQUIRE") and call Pkg.resolve().

`clone(url[, pkg])` Clone a package directly from the git URL url. The package does not need to be a registered in Pkg.dir("METADATA").

Some `Pkg` commands

Package Manager Functions

Your package requirements are in the file `~/.julia/v0.4/REQUIRE`

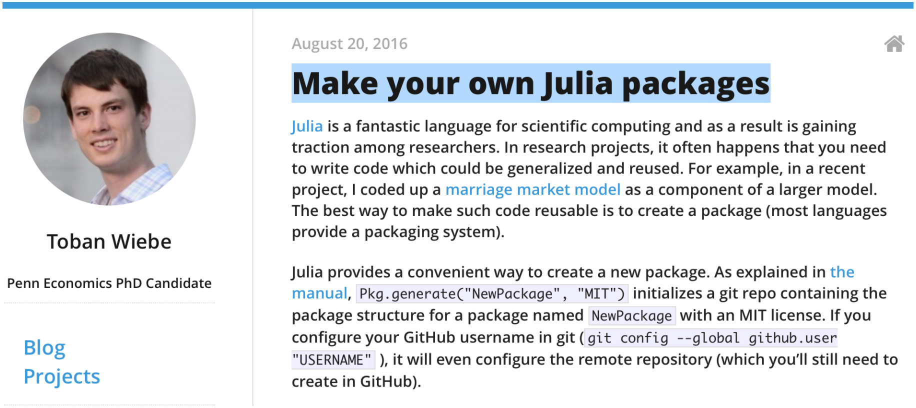
`Pkg.status()` prints out a summary of the state of packages you have installed.

`Pkg.add(pkg, vers...)` Add a requirement entry for pkg to Pkg.dir("REQUIRE") and call Pkg.resolve().

`clone(url[, pkg])` Clone a package directly from the git URL url. The package does not need to be a registered in Pkg.dir("METADATA").

`Pkg.rm()` to remove the requirement for it from the `REQUIRE` file

Make your own Julia packages



Toban Wiebe

Penn Economics PhD Candidate

Blog
Projects

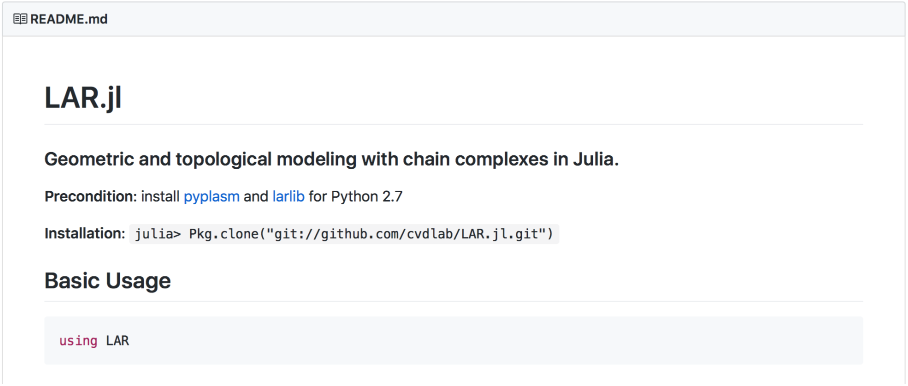
August 20, 2016

Make your own Julia packages

Julia is a fantastic language for scientific computing and as a result is gaining traction among researchers. In research projects, it often happens that you need to write code which could be generalized and reused. For example, in a recent project, I coded up a marriage market model as a component of a larger model. The best way to make such code reusable is to create a package (most languages provide a packaging system).

Julia provides a convenient way to create a new package. As explained in the manual, `Pkg.generate("NewPackage", "MIT")` initializes a git repo containing the package structure for a package named `NewPackage` with an MIT license. If you configure your GitHub username in git (`git config --global github.user "USERNAME"`), it will even configure the remote repository (which you'll still need to create in GitHub).

Package example: `LAR.jl`



README.md

LAR.jl

Geometric and topological modeling with chain complexes in Julia.

Precondition: install pyplasm and larlib for Python 2.7

Installation: `julia> Pkg.clone("git://github.com/cvdlab/LAR.jl.git")`

Basic Usage

```
using LAR
```

`include()` vs `require()` vs ...

- `using/import/require` are for acquiring “shared resources” – i.e. modules that different bits of code might all independently want to use

`include()` vs `require()` vs ...

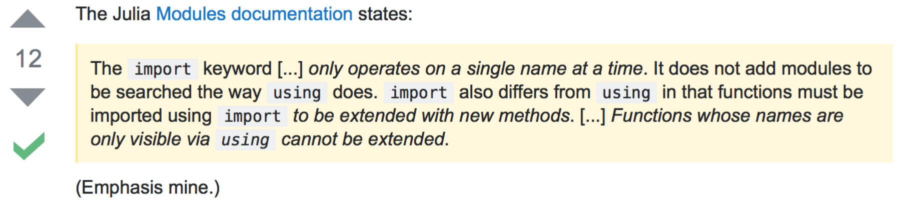
- `using/import/require` are for acquiring “shared resources” – i.e. modules that different bits of code might all independently want to use
- `include` is just about splitting a single source file into multiple pieces.

include() vs require() vs ...

- `using/import/require` are for acquiring "shared resources" – i.e. modules that different bits of code might all independently want to use
- `include` is just about splitting a single source file into multiple pieces.
- `reload` Like "require", except forces loading of files regardless of whether they have been loaded before. Typically used when interactively developing libraries.

A common question

What is the difference between `using` and `import` in Julia when I'm building my own module?



12

The Julia Modules documentation states:

> The `import` keyword [...] *only operates on a single name at a time*. It does not add modules to be searched the way `using` does. `import` also differs from `using` in that functions must be imported using `import` *to be extended with new methods*. [...] *Functions whose names are only visible via* `using` *cannot be extended*.

(Emphasis mine.)

References