Handbook and Operations Manual

Institute for Intelligent Systems

THE UNIVERSITY OF MEMPHIS

IIS Handbook and Operations Manual

:

Table of Contents

IIS Handbook and Operations Manual

Preface by Andrew Olney

Welcome to the first version of the IIS Handbook and Operations Manual. The IIS has never had a book like this before, but as the IIS has grown in scope and complexity, so has the need for a handbook that documents our programs and procedures.

This first version (17.05 in year/month notation) reflects my personal views that have developed over the last 10 years as I've served as Associate Director and eventually Director of the IIS. Accordingly, the priorities and biases are mine and have not been endorsed by the IIS community.

My hope is that future versions will be voted into adoption or follow some similar consensus-building process. By hosting this handbook on GitHub and using AsciiDoc, I've made it as easy as I know how for this book to be improved by a distributed group of people, potentially across many years, such that all revisions are well documented. This approach is well-traveled by other open source books, and I've leaned heavily on the model provided by **Pro Git** (2nd edition). To make changes, please refer to the procedures in the README file in the root of this repository.

Getting Started

This chapter provides a common ground for everyone that is assumed for the following chapters. It gives a high-level view of what the IIS is, what it does, and how it works. Some of these topics are further developed for student and faculty concerns in later chapters.

About the IIS

What is the The Institute for Intelligent Systems?

The Institute for Intelligent Systems (IIS) is an interdisciplinary research center at the University of Memphis. Unlike many research centers, the IIS has historically pursued a research agenda set by participating researchers who **self-organize around research problems**, rather than a highly focused formal agenda. The breadth and flexibility of the IIS is reflected in our current mission statement:

> The Institute for Intelligent Systems is dedicated to advancing the state of knowledge and capabilities of intelligent systems, including psychological, biological, and artificial systems.
>
> By conducting cutting-edge research and publishing our findings in peer-reviewed venues, we contribute to the discipline and, ultimately, to the public. In doing so, we are also training the next generation of scientists.
>
> Our mission depends on an interdisciplinary approach that brings together researchers from many different backgrounds, including computer science, cognitive psychology, education, philosophy, linguistics, engineering, English, and biology.

TIP Our "bottom-up" approach to forming research groups makes the IIS very accessible: if you can find some like-minded people interested in your project, you can form a group.

Although the IIS maintains a strong tradition of bottom-up research, our recent strategic plan adopted three focus areas:

- Language & Discourse
- Learning
- Artificial Intelligence

These focus areas represent the deepest areas of *current* expertise at the IIS, but they should not be regarded as an intention to exclude other areas of research or limit what the IIS can become in the future.

Challenge of Interdisciplinary Research

Interdisciplinary research is hard. To be successful, we have to step outside our disciplinary perspectives and be willing to learn and to be challenged. This means not only learning about other disciplines, which have their own literature, methods, and approach to knowledge, but also learning to work on teams with members of other disciplines. These issues can impact student milestones and professional development, as well as tenure and career advancement for faculty. Within a university that has traditional departments, interdisciplinary research can challenge typical departmental procedures from both an academic (e.g. committee composition) and finance (e.g. course buyout) perspective.

Assume Ignorance

The best policy in interdisciplinary research is to assume a limited understanding of other fields. To paraphrase Bilbo, you don’t know half of other fields half as well as you would like; and you like less than half of other fields half as well as they deserve. Misunderstandings often arise when researchers use themselves (or members of their field) as a model for researchers in other fields. The reason this doesn’t work is that the differences between fields can be quite profound, well beyond the superficial issue of the area or object of study. The differences include:

Knowledge

What does it mean to "know" something and how can we arrive at "knowing?" Not all fields are empirical; not all fields use quantitative research methods.

Language

Each field has its own vocabulary and style of discourse that expresses "knowing." The same word or phrase may have different meanings across fields, e.g. "deep learning" means neural networks to computer science, but "deep learning" means durable, semantically interconnected memory to a psychologist.

Scholarly products

What scholarly products does a "successful" researcher produce, and how often? Some fields give great weight to books; some fields give great weight to conference proceedings.

Student training

How are students trained in the field? What kind of mentoring is expected, and what experiences are students expected to have? Some fields may expect hundreds or thousands of hours of clinical practice or bench work.

Communicate the Obvious

Perhaps the biggest problem facing interdisciplinary teams is that communication on these issues either doesn’t happen, or happens late in the project. Therefore it is a good idea to communicate about what, at first, may seem obvious.

For each researcher

- What questions are interesting

- What they can contribute to the project in terms of answering these questions or the questions of others
- How many hours they can **really** contribute per week, including meetings, and any "dead" times during the year where they won't be contributing much
- What they need to do that year to be successful (e.g. to their department, includes teaching, etc)
- What publications are planned
- What mentored students need along the above [1: Ideally, students working on interdisciplinary projects will be mentored by someone in their field. If this is not the case, students should keep in close communication with their advisor to ensure that their professional development is on track.]

Avoid the Technician Trap

Interdisciplinary researchers can contribute to a project in many ways. Many times, they contribute distinct technical skills related to their field. While this contribution can be important to the project as a whole, it may not make for a rewarding experience for the researcher. Application of technical skills may not count as scholarly output and thus not count towards career advancement. For example, a computer scientist may write software or a psychologist may run an experiment, but these activities in themselves do not constitute scholarly products. Moreover, these activities may be **extremely** time consuming and actually prevent the researcher from engaging in other activities.

When considering a project, researchers should be aware of the technician trap and consider two key questions

- Can I make this successful
- Can this make me successful

If there are conditions attached to these questions, i.e. to be successful, the researcher will need "X", then this is a worthy topic of discussion and negotiation on the front end of the project. For example, a psychologist may need to change the design of the experiment to make it publishable in their field, while still retaining the characteristics needed to advance the project. Some give and take is needed in any interdisciplinary project to ensure all researchers can benefit from the project.

Interdisciplinary Norms for Publishing

Different fields have different norms for publishing. This can affect who is on the paper, who is first author, and where the paper is published. It is important to clarify publishing expectations **at the start of the project** when there is no resource conflict (because the resource, i.e. the opportunity to publish, does not yet exist).

Things to ask:

- What is the criteria for authorship
- Who will take the lead writing up different results

- Who else will be listed as an authors on these papers
- Are students expected/encouraged to publish work on this project
- Can a student be first author (assuming that concept is valid; some fields list authors alphabetically)

Remember, in some fields, non-intellectual contributions (e.g. collecting data) do not warrant authorship, and in others, the PI of a grant may be expected to be on all publications as last author - different fields have different standards.

For an extended discussion of this subject in psychology, see these APA guidelines.

A Short History of the IIS

The IIS was formed in 1985 as a result of informal meetings between Dr. Stan Franklin (Computer Science), Dr. Art Graesser (Psychology), and Dr. Terry Horgan (Philosophy), with Graesser and Franklin initially serving as Co-Directors. Dr. Don Franceschetti (Physics) developed a five-year plan for the IIS which led to the recognition by the State of Tennessee of the IIS as an Institute. The Cognitive Science Seminar began in 1985 and continues to this day.

NOTE The Cognitive Science Seminar has been in continuous operation since 1985. It is now a cross-listed course with Psychology, Philosophy, and Computer Science.

In 1998, as a result of multiple major grants to faculty participating in the IIS, the IIS received an operating budget of $25,000, 5% indirect cost recovery from grants, and two staff positions that were both filled by 2000.

In 2003, the IIS moved to the FedEx Institute of Technology (FIT) and began hiring two faculty with split appointments between the IIS and other departments.

Beginning in 2007, the IIS began hiring 100% faculty lines. These lines are fully within the IIS but have tenure housed in other departments. This development marked a number of administrative changes in the IIS as it continued to grow into a mini-department. An additional staff person was added to support grant submissions (i.e. pre-award support) in recognition of the importance of grant funding to the IIS mission.

In 2010, the IIS was allocated 10 graduate assistantships, additional faculty lines, additional staff, 20% indirect cost recovery, and the 4th floor of the FIT. The IIS Student Organization (IISSO) was formed and tasked with managing student travel awards. The IIS added its first degree program, the Cognitive Science Graduate Certificate.

In 2014, the IIS launched a multi-year strategic planning process. Many of the recommendations of the strategic planning process have already been implemented, including a new undergraduate Minor in Cognitive Science. However parts of the strategic plan will take years to unfold.

For more detailed history of the IIS, up to 2015, please refer to the IIS Strategic Plan Self-Study.

IIS Membership

For many years the IIS did not have a formal membership definition. [2: Stan Franklin had a retrospective method that calculated membership based on participation over a period of time. However this did not allow one to "sign up," which is a common attribute of memberships.]

Since 2008, the IIS has had a membership letter mechanism: faculty can join the IIS as "IIS Affiliates" by signing the letter and returning it to the IIS Director. The letter states what the Affiliate can expect from the IIS and vice versa. In a nutshell, the IIS expects participation and, ultimately, grant activity that funds IIS services. In return the Affiliate has access to the various resources the IIS provides.

There is not a similar membership letter for students. Students may join the IIS Student Organization (IISSO) according to the current policies and procedures of that organization. However, being a member of the IISSO does not automatically grant access to IIS resources. To access IIS resources, students must be "claimed" by an IIS Affiliate faculty member as described later in this handbook.

IIS Student Organization (IISSO)

The IIS Student Organization (IISSO) is a self-organized group of students, most of whom work with IIS Affiliates. In addition to providing a community for students, the IISSO makes recommendations to the IIS regarding the allocation of various resources to students, including student travel awards and thesis/dissertation awards. The IISSO also organizes the poster session at the Speed Date, guest speaker lunches, and other social events.

Current information about the IISSO can be found at https://sites.google.com/site/iissomemphis2/

Students and faculty should refer to current information on the website.

A copy of the IISSO charter, **which may not be current**, follows.

Charter

Mission Statement

The IISSO is a student-led advocacy and coordinating group at the Institute for Intelligent Systems (IIS). The IISSO represents the interests of undergraduate and graduate students that are affiliated with the IIS. The IISSO actively pursues opportunities to improve student welfare through academic and social projects, and also strives to advance student research by providing opportunities for funding, networking, and promotion of research. IISSO meetings are scheduled once or twice a semester and all IIS-affiliated students are welcome to attend. The meeting minutes will be distributed through the IISSO email list within one week of the meeting. Student elections for the IISSO officers are held annually in May.

Its major responsibilities include a) distributing travel funds to students that attend academic conferences and workshops*, b) organizing student lunches with visiting speakers, c) providing opportunities for students to present their research and/or lead workshops/seminars, d) providing opportunities to interact with professors interviewing for IIS-faculty positions, e) keeping students informed of IISSO and IIS activities, f) training the new IISSO officers-elect, and currently on pilot basis, g) distributing thesis and dissertation funds to students toward the completion of their degrees*.

*See eligibility requirements and submission guidelines on the Funding page.

Qualification for Membership

Any undergraduate or graduate student affiliated with the IIS or with a strong interest in cognitive science interdisciplinary research is able to join and participate in IISSO events.

Membership does not have any strict requirements. To join, simply email the President to have your name added to the mailing list (MUST be your U of M email). The IISSO strives to advance student research by providing opportunities for funding, networking, and promotion of research. As such, members have access to the following benefits: participation in the IISSO research fair (usually toward the end of the Fall and Spring semesters), the opportunity to collaborate with researchers in different fields, as well as the chance to meet and network with experts from outside of the U of M in a variety of cognitive science fields (through the Cognitive Science Seminar lunches), attend and/or present workshops to further your academic career/CV.

While anyone is welcome to join and may benefit from any or all of the abovementioned benefits of membership, limited travel funding (and thesis/dissertation funding) can be awarded to members who meet certain criteria. Specifically, one must be an IISSO member collaborating on a project with an IIS-approved faculty member. Upon joining the IISSO, if you are interested in possibly applying for travel funding in the future, you should check with your advisor or any professor you are collaborating with to see if they are IIS-affiliated and if they have added their students to the list of IISSO-travel-funding-approved students. Even if you meet this requirement (and others detailed on the travel funding page), no one is guaranteed funding. For more information on travel funding, please see the IISSO Funding Page. There is also preliminary information on the Thesis/Dissertation funding on the Funding page. You may also contact any of the current officers with questions or concerns about membership and eligibility.

Events

Research Meetings

Individual projects and labs schedule weekly meetings in 405 and 407 FIT, the IIS main meeting rooms. Affiliates can book rooms with Renee Cogar. These meetings and descriptions are publicly posted to our website, and all IIS Affiliates and students are encouraged to drop in or participate in ongoing research meetings. There are approximately 25 regularly scheduled research meetings per week for externally funded projects.

Cognitive Science Seminar

The Cognitive Science Seminar is a hybrid course/public lecture. Students who register for the course (COMP/PHIL/PSYC 7514/8514) attend a course-only portion from 2:20pm to 4pm as well as a guest lecture portion, which is also open to the public, from 4pm to 5:20pm.

The topic of the seminar varies semester by semester, as do the faculty leading it. It is common for students to have lunch with guest speakers and for faculty to take them to dinner. More information about the seminar can be found in the faculty chapter.

Guest Speakers

IIS Affiliates will often host visitors who are willing to give a talk to the wider IIS community. Although the IIS does not provide material support for such ad-hoc speakers, we encourage Affiliates to make use of our meeting rooms and work with IIS staff to promote the talk.

Speed Date

At the end of each semester (typically 4pm to 7pm the Friday before Study Day), the IIS holds a Speed Date event in which 8-10 Affiliates give 5 minute presentations of their current research interests. The presentation portion is followed by a wine and cheese poster session, organized by the IISSO, where students present their latest research. Affiliates who wish to volunteer to speak at the Speed Date are always welcome to do so, otherwise Affiliates are invited based on how much time has passed since their last presentation, with priority given to Affiliates who recently joined.

Communications

Email

The IIS maintains multiple mailing lists:

- IIS-Faculty: faculty affiliates.
- IIS-Students: **interested** students, some of which might not be working with IIS affiliates.
- IIS-Pro: grant-supported staff and postdocs
- IIS-Admin: president, provost, vice president of research, FIT administration, deans and chairs of all faculty affiliates, staff in communications

Web

The IIS website has the following elements:

- List of affiliates and staff
- Degree programs
- Calendar of research meetings
- Links to resources
- List of affiliate projects

The IIS Student Organization (IISSO) has their own website with specific information for students.

Social Media

Currently the IIS participates in:

- LinkedIn : there is a LinkedIn group for current and former students, staff, and faculty.
- Facebook

IIS social media accounts are managed by Leah Windsor.

Press Releases

In the past, faculty and students were largely on their own with respect to press releases, meaning that they had to write them, submit them to communications, and then wait and see if anything came of it.

The current process is that Mary Ann Dawson is on the IIS-Admin list and will proactively create press releases. If faculty or students have a particular story they think is worthy of a press release, they are encouraged to contact Mary Ann Dawson directly.

Branding

Use of IIS logo and branding is encouraged on presentation slides and conference posters. Discretion and care should be taken to avoid using the IIS brand in such a way that it endorses an activity besides an individual's research. Sponsorships and similar promotion/endorsements must be approved by the IIS Director.

Logo art is available on the IIS website under Resources.

Signage

The IIS has several signs used primarily for IIS events like the Speed Date. These signs can be used for other purposes but need approval (like branding) and can be checked out from Renee Cogar.

Brochures

General brochures about the IIS are available. While the IIS can provide these in limited quantities (e.g. 100), it should be recognized that these are produced at cost and should be used strategically (e.g. to attract students). General brochures can be obtained from Renee Cogar.

IIS can also assist faculty affiliates with the creation of brochures specific to their lab or grant project. Faculty wishing to create custom brochures should contact Adam Remsen.

Outreach

The IIS engages in outreach and encourages students and faculty to participate. These activities help promote the IIS brand, attract future students, and help us connect with new collaborators. Outreach activities can take place on-campus or off-campus.

On-campus, IIS faculty and students are frequently asked to participate in lab tours and associated demonstrations. Often touring groups are middle or high school students, so the tours/demonstrations need to be tailored to a general audience. These are excellent opportunities for graduate students to practice their presentation skills.

Off-campus events include demonstrations at schools or community events, speaking engagements, competition judging, and similar activities. Often times off-campus outreach is spearheaded by a particular faculty member, but to the extent possible, the IIS can support these activities by providing signs or similar materials.

Resources

Resources specific to students and faculty are addressed in their respective chapters. The resources that follow are available to everyone. Additional resources are listed on the IIS website Resource page.

Meeting space

Faculty and students can book meeting rooms with Renee Cogar. In general, booking is on a first-come, first-served basis.

Teleconference room

Faculty and students can also book the IIS teleconference system with Renee Cogar. The system is located in 407 FIT.

The system is an enterprise version of Google Hangouts with the following features

- Accomodates a large group with one screen, camera, and microphone
- Allows a large group of attendees (~20)
- Can connect participants via telephone in addition to Google Hangouts
- For recurring meetings with fixed participants, can automatically start meetings and invite participants based on a calendar event (managed by Renee Cogar).

Library

The IIS Library is open to faculty and students at any time. The library holds a large collection of books donated by faculty (approximately 2,000).

Each book is indexed and cataloged, and may be searched in two ways - Book metadata such as author and title - Full text search using Google Books search restricted to our library

Eyetracker room

The IIS has a "single instrument" experiment room, which currently contains a Tobii T60 Eyetracker.

This room may be booked for single instrument experiments through Renee Cogar.

Lab space

Lab space on the 4th floor of the FIT is assigned based on grant funding. The IIS has a strong history of collegiality with regard to space, with faculty between grants giving up space in full confidence that when they need it again, other faculty will be equally accomodating. Additionally, some space has been allocated to IIS faculty lines, who may not have space elsewhere.

Students or faculty without grant funding, who want temporary space, should contact the IIS Director. Temporary allocations may be warranted for pilot projects that lead to grant funding.

Subscriptions and Social media

Linguistic Data Consortium subscription (LDC)

The Linguistic Data Consortium (LDC) is perhaps the world's leading repository of linguistic data. LDC datasets are commonly used in shared tasks and to benchmark algorithms. They can cost thousands of dollars each.

The IIS has an institutional subscription, which gives us total access to each dataset released in a subscription year as well as a reduced price on non-covered datasets. Datasets in our subscription years are available on campus at http://psychiisnas.memphis.edu/LDC/start.html

Certain datasets require special license agreements. If a dataset appears to be missing, it may require a signature release. Contact Renee Cogar for such requests.

Dreamspark/Imagine Subscription

The IIS has a DreamSpark subscription. IIS Students, faculty, and staff may download Microsoft titles from our webstore.

The primary types of software offered are

- Windows operating systems
- SQL server and similar server titles
- Visual studio and similar development tools

Faculty and students can obtain Microsoft Office products directly from the University at http://www.memphis.edu/getoffice

GitHub

The IIS maintains a private GitHub account for faculty projects.

Google Apps for Education

The IIS maintains its own Google Apps for Education account, which provides all major Google services except email.

Equipment

Equipment check out

Various equipment is available for checkout, including

- Projectors

- Google glass

Equipment may be checked out through Renee Cogar.

Computers

Under current federal guidelines, general-purpose computers cannot be charged to grants. A general purpose computer is any computer not specially configured for a single task. For example, if a student checks their email or writes a paper on a computer, it is a gneral-purpose computer, but if a computer’s primary purpose is to drive an eyetracker, that is an "unlike circumstance" computer that can be charged to a grant.

The IIS provides general-purpose computers to support grants. Typically these computers are replaced on a 3-year cycle. Older computers replaced on this program are available to all IIS Affiliates and will be announced as replacement occurs.

Printing

The IIS maintains multiple printers, chiefly in 410 FIT. Like general-purpose computers, these printers are considered part of general office supplies. Each computer is attached to the network and can be accessed from any computer. To do so, walk up to the printer, open its information menu, and write down the IP address of the printer. Use this IP address when adding the printer to your computer. Please keep the printer clean, filled with paper, and report problems to Renee Cogar as a courtesy to your colleagues.

Degree Programs

This IIS hosts two degree programs, the Cognitive Science graduate certificate (a graduate program) and the Cognitive Science Minor (an undergraduate program). Currently both programs consist of courses hosted in other departments, but often taught by IIS Affiliates. Information about both programs is available at http://www.memphis.edu/cognitive-science/

Importantly, that link lists the specific courses offered in the current/upcoming semester that apply to the certificate and minor, which should help students when registering for classes.

Cognitive Science Graduate Certificate

The certificate is available to all enrolled graduate students from any department. Additionally, students can be on the certificate as **non-degree seeking** and by doing so be eligible for GA positions. Similarly, non-degree seeking students on the certificate are eligible for financial aid. So a student between UG and a graduate program may find taking the certificate to be an effective use of their time while they are waiting to be admitted to a graduate program.

Ideally students interested in the certificate will apply for admission right away, but it is also possible to enter the certificate right before graduating and have previous classes retroactively count toward the certificate (as long as no more than 6 hours are "double counted" towards the certificate and another degree).

Students already enrolled in a graduate program (or who have previously been degree seeking) can apply to enter the certificate by filling out a change of status form:

http://www.memphis.edu/gradschool/pdfs/forms/changeofstatus.pdf

with degree program as "Graduate Certificate", Major: Cognitive Science.

and then filling out the form, printing/signing, and mailing to the Graduate School or walking over to the Graduate School.

If the student is currently enrolled as non-degree and has never been degree seeking he or she must complete an online readmit application.

For non-degree students, the IIS graduate coordinator, Andrew Olney, is the advisor of record and must clear classes each semester. For degree-seeking students the regular advisor remains the advisor of record; however feel free to contact me for advice on course selection.

In addition to the listed courses that can count towards the certificate, the graduate coordinator may approve course substitutions. Faculty are encouraged to contact the graduate coordinator if they are teaching a "one off" seminar course that is aligned with the certificate or a new course that is similarly aligned. Students are encouraged to contact the graduate coordinator for advice regarding courses, particularly courses outside their area.

In the semester of graduation, students on the certificate must fill out the following forms

- Apply to graduate
- Graduate certificate candidacy form
- Course substitution forms (as needed)

Forms are available at http://www.memphis.edu/gradschool/resources/forms_index.php

Andrew Olney will sign off on the forms and route them for further approval.

The deadlines are listed here:

http://www.memphis.edu/gradschool/current_students/graduation_information/graduation_deadlines.php

Note that deadlines for graduation paperwork are early in the semester

Please contact Andrew Olney if you have any questions.

Cognitive Science Minor

The minor is available to all undergraduate students **except** Psychology majors. Psychology majors should instead take the Cognitive Science concentration within the Psychology major.

Unlike the graduate certificate, the minor is largely administered by the College of Arts and Sciences (CAS). Students wishing to declare the minor should follow the CAS procedure at http://www.memphis.edu/cas/advising/declare_major.php

A student wishing a course substitution or exception should contact the IIS undergraduate coordinator, Andrew Olney, who will contact graduation analysts in CAS to make the change.

Staff

Most staff are available during academic hours and on an as-needed basis. The major exceptions are grant preparation and software development.

Grant preparation support is negotiated in advance and coordinated according to other grant submissions that are due at approximately the same time. This coordination encourages faculty to notify of intent to submit and work with our grant preparation staff well in advance of the submission deadline.

Software development likewise is time-intensive for any given project. Currently software development support is allocated using an RFP system. Faculty write a 1-2 page narrative describing how their project would benefit from software development support. Support is allocated in 2 month blocks per project on a 6 month cycle. During any 2 month block, our software development staff member is 100% committed to the assigned project.

Senior Administrative Secretary (Renee Cogar)

- Manage all meeting space (405/407)
- Manage all IIS email lists
- Manage library; order books and journals
- Handles travel bookings as requested
- Book events like the Speed Date
- Handles mail, faxing, and phone communications

Financial Services Associate (Jenice Jackson)

- Manage IIS Operating and IDCR accounts
- Purchasing
- GA contracts
- Reimbursement
- Travel authorization
- Inventory

Pre-Award Coordinator (Adam Remsen)

- Track and route funding opportunities to faculty
- Grant proposal preparation
- Budget, budget justification, advance account requests
- Edits documents and checks for sponsor compliance
- Maintains current and pending support database

- Edits faculty CVs
- Prepares proposal packages; coordinates submission with internal and external research offices
- Other grant-related activities as needed

Business Officer (Mattie Haynes)

- Manage all IIS grant accounts
- Monitor IIS Operating and IDCR accounts
- Manage program income
- Provide monthly reports on grant and IDCR accounts
- Rebudget as needed
- Manage summer salary, AY incentive, and check compliance
- HR functions; prepare contracts for research faculty
- Act as liaison with the office of research support services, accounts payable, financial planning,
- and grants accounting on behalf of the IIS (faculty, staff, students, etc.).

Senior Software Developer (Andrew Tackett)

- Develop and maintain grant-sponsored software
- Trains others in use and installation of software, prepared documentation
- Maintains versioning, code repositories, and releases of software
- Supervises and trains students in software development
- GitHub administrator

Director (Alistair Windsor)

- Immigration and sabbatical invitation letters
- Annual evaluations for faculty
- Annual evaluations for staff
- Assign research space as needed
- Assign computing resources as needed
- Assign IIS GAs on annual basis
- Monitor and update website as needed
- Coordinate Cognitive Science Graduate Certificate (admissions, matriculation, etc)
- Recruit and coordinate instructors for the CogSci seminar
- Recruit and coordinate speakers for the Speed Date
- Coordinate public demonstrations and outreach
- Serve as committee chair/member for faculty hires

- Recruit and retain IIS Affiliates, including counteroffers
- Work with IISSO to maintain student travel funding program
- Produce annual report and program reviews as needed
- DreamSpark, Google Apps for Education, and LinkedIn group administrator
- Monitor the IIS and intervene as needed

Summary

You should have a basic understanding of what the IIS is, what it does, and how it works.

Students

This chapter is dedicated to students. It assumes you've already read the Getting Started chapter. If you have not read that, go back and read it now!

The IIS has different support for students than it does for faculty. The major support structure is the IISSO. After discussing the IISSO, we will cover some additional topics with a view to the interdisciplinary issues that sometimes come up and your professional development. These sections are intended to supplement, rather than replace, the mentoring you receive from your major professor.

IISSO

IISSO currently makes funding recommendations for student conference travel and thesis/dissertation funding.

The policies guiding these recommendations is constantly in flux, so students are advised to check the following link for the current policies https://sites.google.com/site/iissomemphis2/travel-info-1

IMPORTANT	Students who don't follow the policies either receive no funding or greatly reduced funding.

Just as important, the IISSO organizes various social events that contribute to professional development, guest speaker lunches and the Speed Date poster session.

Guest speaker lunches are an extremely important way for students to get practice talking to unfamiliar colleagues about their research. This is a scenario that will occur again and again in professional life, particularly on job interviews, and guest speaker lunches provide a friendly, low-risk environment in which to practice.

The Speed Date poster session is similarly a great opportunity to practice speaking and presentation skills. Unlike an academic conference, where the audience may be specialists in a particular area, the Speed Date audience is highly diverse. This creates an opportunity to practice speaking with interested colleagues who are non-specialists, again in a friendly low-risk environment. Students are encouraged to use posters from recent or upcoming conference presentations rather than creating posters specifically for the Speed Date, though creating posters specifically for the Speed Date is not discouraged.

Officer Responsibilities

Students are also encouraged to take various service roles in the IISSO, detailed below.

President

- To act as the spokesperson for the IISSO
    - To be aware of the responsibilities of all IISSO members
- To delegate the responsibilities and charges of the IISSO officers and representatives as necessary

- To confirm responsibilities and charges of each member are fulfilled
- To stay informed of IISSO member activities
- To initiate and delegate the development and planning of annual IISSO activities
    - To assist in developing subcommittees in order for annual activities to be fulfilled
- To call meetings and forums deemed important to the IISSO and the IIS
- To attend IISSO Director meetings, IISSO meetings, and forums
- To determine what information from IISSO meeting minutes is sent to the larger IIS group
- To motivate IISSO members to fulfill their responsibilities
- To consult with the Vice-President before requesting new strategies or activities be implemented for the IISSO
- To gather consensus among IISSO members before implementing new strategies or activities for the IISSO or IIS group

Vice-President

- To assume the responsibilities of the IISSO President should there be an extended absence
- To act as the spokesperson for the IISSO in the absence of the President or as requested by the President
- To be aware of the responsibilities of all IISSO members
- To support and assist the IISSO President in fulfilling his/her responsibilities
- To attend IISSO Director meetings, IISSO meetings, and forums
- To act as a “sounding board” to the President in order to encourage well-rounded ideas before promoting them to the IISSO as a whole
- To gather consensus among IISSO members before implementing new strategies or activities for the IISSO or student body
- To motivate IISSO members to fulfill their responsibilities
- Serves as research coordinator until further notice.

Research Coordinator

- To act as Chair of the IISSO research conferences
    - The Research Coordinator is responsible for handling required tasks for the IISSO research fair each semester, including:
        - Establish submission guidelines and time frame
        - Secure/reserve venue
        - Recruit faculty and graduate students to host workshops/seminars
        - Supervise registration
        - Recruit staff/volunteers
        - Create room layouts and assignments (table, chairs, podium, etc.)

        - Secure supplies and A/V needs

Treasurer

- To meet with IIS Directors before the Fall semester to determine the budget for the upcoming year
- To communicate funding procedures and deadlines to IIS student AND faculty affiliates in the first week of the Fall, Spring, and Summer semesters
- To provide and maintain updated travel applications
    - The Treasurer will collect and review the applications for completeness and solicit applicants for additional information as needed
- To score the applications to determine how much funding will be offered
    - The scoring procedure should adhere to a policy determined by the Treasurer, IIS Directors, and IIS Administration at the beginning of each school year
- To notify the applicants of their awards
    - Award notices should be in the form of a written letter signed by the Treasurer that indicates the recipient and the amount of the award. Students who are away on internship are notified by email.
- To record and compile a text document at the end of the each round of funding of who received funding, for how much, and for what event the funding was applied
    - This text document needs to be submitted to the IIS Director and the IIS Finance Secretary (Mattie Haynes)
    - This text document, minus the award amount, needs to be submitted to the IISSO Secretary/Archivist for inclusion in the semester newsletter.

Secretary

- Secretarial Duties
    - To attend every meeting of the IISSO and record minutes
    - To keep the cumulative meeting minutes together in either hard or electronic copies (and available on the organization website) for members or interested individuals to access
    - To keep a record of any organization documents

Ambassador

- To organize and initiate outreach activities during the academic year including:
    - tours of the IIS labs for prospective students
    - informational workshops highlighting the research activities of the IIS
- To coordinate student lunches with visiting lecturers/faculty
- To organize and meet monthly with a committee comprised of student representatives from all IIS-affiliated departments
    - A status report from the committee will be presented at the monthly IISSO meetings

- To encourage students affiliated with the IIS to become active members of the IISSO
    - Requires assisting Secretary/Archivist, Research Coordinator, and Social Coordinator in coordinating activities with IIS-affiliated students/departments

Social Coordinator

- To plan all IISSO student lunches with visiting professors or speakers. This includes speakers for the Cognitive Science Seminar and prospective job candidates when applicable.
    - Coordinate with the Professor running the current semester's cognitive Science seminar in order to arrange the lunches.
    - Email IISSO members when lunch opportunities arise and compile a list of who attended, and send list to the treasurer for attendance records.
    - Submit all lunch receipts to Vickie Middleton.

Webmaster

- To maintain and update the IISSO website, including:
    - Oversee the development and distribution of the IISSO website, containing organization information and news.
    - Keep track of organization history (events, research, meeting dates, etc.) on the website
    - Gather consensus among IISSO members pertaining to website updates, additions, and general requested changes.
    - Keep the website information up to date (membership directory, current officers, upcoming event, etc.)

Recruitment Officer

- To attend IISSO director and general meetings
- To carry out annual IISSO activities as delegated by the President, Vice President, and/or subcommittees
- To seek out important happenings in affiliated departments and report this information at IISSO meetings
- To stay updated as to the events and activities within each department and report this information to the IISSO
- To volunteer for subcommittees in order to assist in carrying out IISSO events and activities
- To encourage students, both graduate and undergraduate, to become involved in the IISSO

Expectations

One of the trickiest things about doing interdisciplinary work is navigating the different ideas and workflows from other fields. Often other fields will do things "a certain way," and **expect** others to do things the same way.

As a student, it is particularly important to be aware of expectations, the fact that expectations from faculty other than your major professor may be different, and that they matter.

Hiring examples

Non-computing faculty hires computing student

A non-computing faculty may have a funded project that requires some programming. That faculty member may have a strong idea of what needs to be done, and be able to communicate the design, or they may not. Similarly, that faculty member may have experience with software development and understand the timecourse of development, or they may not.

A computing student who is hired as a graduate assistant to such a faculty member needs to be very careful. Why? Because the expectations of the faculty member and of the student may be far apart. The faculty member may assume that it will take 3 months to develop the software. The student may have **no idea** how long it takes to develop, but hopes the faculty member is understanding of any unforeseen challenges and the learning curve.

Computing faculty hires psychology student

A computing faculty may have a project that requires some kind of human experiment for validation. So the faculty may hire a psychology student to run the experiment. The faculty member may know something about running experiments, or they may not. Likewise, they may have experience designing experiments, or they may not. Again, a psychology student hired as a graduate assistant needs to be very careful because of expectation mismatch.

The faculty member may not understand that it can take 1-2 months to get a study approved by IRB and that the study has to be completely specified in the IRB submission (all materials, etc). The faculty member may have unrealistic expectations about how long it will take to run the study, e.g. an eyetracker study with 100 subjects. And the psychology student may not really know how long some things will take.

Dealing with these examples

How can we make these situations better? Ideally both sides need to learn a little about the other. However, that can’t happen without communication.

TIP When doing interdisciplinary work, communicate twice as much (both as often and in length) as you would with somone in your field.

Otherwise, two really bad things can happen

- The faculty member thinks things are done when they aren’t
- The faculty member thinks you aren’t working when you are

Thesis/dissertation expectations

Different fields have different standards for theses and dissertations. In some fields, these will be brand new work; in other fields, these will be existing publications sandwiched with an introduction and conclusion.

When you ask someone from another field to be on your committee, you are implicitly agreeing to, either partially or in full, meet the standards of their field. No professor will be on your committee and let you say/do things that run contrary to their field. So, think very carefully about your committee membership, **talk** in advance to potential committee members to make sure they are agreeable to your plan.

Additionally, different fields have different conventions about

- How defense meetings are conducted (formal/informal, when questions are taken)
- Format of the document (e.g proposal in perfect format vs. draft)
- How far in advance the document is sent to the committee (e.g. 2 weeks or more)

Again, it is important to discuss these things with your committee and your major professor. You should also be very careful about if your department will allow outside members on your committee. Most departments require a committee chair in your department, some allow only one member from another field, some **require** a member from another field.

Baseline expectations

In addition to the above, there are baseline expectations for all graduate students. These are things that all faculty members expect, regardless of discipline.

Communication
Emails returned in under a day

Deadlines
Should never be missed

Work
Documented such that if you were never seen again, **someone could pick up where you left off ten years from now** and understand what you did

Attendance
Posted the hours your will be at your desk ; Don’t skip meetings; let people know **in advance** when you are sick

Initiative
Work independently; suggest ideas to problems; participate in meetings (that means speaking)

Website/CV
Up to date with your status and accomplishments

Writing
Grammatical; organized; proper lit review

Reading
Familiarity with literature in your area

Setting goals

In the end, your degree and your professional career is up to you. You need to figure out your goals and how to best achieve them. Your major professor can help, and resources from the IIS can help. But they will only help if you actually make use of them.

One way to approach goal setting is to work backwards.

What job are you preparing for?

It is important to recognize that you are preparing, now, for a future job. You want to be the **successful applicant** for that job, meaning you are selected over all other applicants. How will you achive that in a competitive world?

TIP Start preparing for your future job today

Jobs are specialized. You will want to prepare slightly differently for different kinds of jobs. Here are some examples:

University Tenure-Track
Publishing is key; teaching is mostly irrelevant

College Tenure-Track
Teaching experience is important; publishing less important than university

Industry Research
Publishing less important than university; technical skills important; internships useful

Industry
Technical skills important; internship experience becomes important

Most faculty will try to prepare you for a university tenure-track. That's the hardest path, and doesn't necessarily shut you down from other paths. How hard is the university path? According to one study, about 20% of PhDs get tenure track jobs. You have to work very hard.

Let's break it down.

Who has the job you want?

All academics have their CVs online. If you want to be a tenure-track professor at Stanford, go to that department website and look at the people they hired in the last five years. It's important to look at recent hires because hiring criteria change over time. Look at the appointment date on these CVs and then filter out the publications/accomplishments that were obtained the year before hire. Why the year before? Because academics go on the job market in the fall to land a job **the following August** Using this process, your goal is defined: to get this person's job, you need to have a CV that was at least as impressive as theirs.

Break it down by year

You should now have an idea of what your yearly publication rate should be and what mix (conference/journal/etc). Perhaps it is two first author publications and two co-authored publications. How do you plan for this? It is important to talk to your major professor and come up with a plan. Saying "I will publish two first author papers this year" is not a plan (it’s just a re-statement of the goal). Again, you can work backwards, but you probably will need guidance to do this well.

Break it down by paper

Conferences have deadlines. If you know what conference you want to publish in, you can use that date and work backwards. Journals don’t have deadlines, but they take a long time. A year or more might pass between submission and seeing your paper in print. In either case, you need to plan. Assuming you already have an idea:

- Talk to your advisor to make sure it’s a good idea
- Do a literature review so you really understand how your idea fits in existing research. If this is 30 previous papers, you might read one paper a day for a month.
- Do whatever work it takes to realize and evaluate your idea. Try to get your advisor to help you.
- Write your idea and results up. If you write a page a day, it will take 8-12 days for a conference paper, 15-30 days for a journal article.
- Send your draft paper to your coauthors for feedback. Expect 1-2 weeks for feedback and time to merge their comments.

As you can see, it will take several months to go from idea to submitted publication. There are a couple of strategies you can use to increase your output. First, always have something under review. Since conferences are seasonal, this implies you will be submitting to journals during other parts of the year. Second, have multiple projects going on at the same time. If you work with other people, this will naturally happen, because its unlikely different research projects will be in synchrony. If you do this, you will have different papers in different stages all the time. This style of work is common in industry. In TV, for example, a show will often have one episode in pre-production, one in production, and another in post-production at any given time.

Everything else

Usually your future employer will not care about your academic transcript. Some might, and it is important to consider that. However, if you spend all your time making straight A’s in your courses but don’t publish well, you are not going to be a successful candidate for a tenure-track job.

On the flip side, there is something else that is important for a tenure-track that students don’t often think about — letters of recommendation. Of course your major professor can write you a letter, but you will also need other letters (typically two others). You should have a long-term plan for cultivating these letters so that they are as good as they can possibly be. The letter should come from someone who knows your work well, has seen you present, has published with you or worked closely with you on a project, and **most importantly** is enthusiastic about your work.

Graduate Assistantships

Students cannot request assistantships directly. Graduate assistantships (GAs) are supported by grants, your home department, and in some cases, the IIS.

Rules

See the Graduate School website for current rules; but generally

- GAs are "on contract" to work 20 hours a week
- GAs receive both stipend and tuition
- GAs must maintain a 3.0 grade point average

IMPORTANT Read your GA contract carefully!

Departmental GAs and grant GAs have the following important difference

- Deparmental GAs must take a full load ({:ga-tuition-hours:} hours) until they start their thesis/dissertation
- Grant GAs can take less than a full load as long as they have student status

Continued employment

Before the end of each semester, you need to

- Ask the faculty member you report to if you will be funded next semester
- Talk to {:financial-services-associate:} about your contract

Extra tuition expenses

Sometimes students want to take an online course or more than {:ga-tuition-hours:} hours. If this happens, you should get permission, otherwise you may be asked to cover those expenses. Why? For departmental GAs, the IIS receives exactly {:ga-tuition-hours:} hours of tuition reimbursement from the university. If you go over {:ga-tuition-hours:} hours, the IIS has to pay from its indirect cost budget, which is the same budget that funds student travel and other items. For grant GAs, this is usually less of a problem, but grants like the IIS have a budget item for tuition that is calculated at {:ga-tuition-hours:} hours. So if you go over, you can actually put the grant into deficit, which the PI of the grant becomes responsible for.

Some perspective

You can look at a GA as just your job, or you can look at it as an investment in your career. As a job, why would you work more than 20 hours if you weren't paid to? However, as an investment in your career, you may choose to work more than 20 hours to be competitive on the job market post-degree.

Speaking and Writing

Speaking and writing are fundamental skills for your professional career. This can be particularly challenging for non-native speakers, but speaking and writing are often challenging for native speakers who are not used to the **genre** of academic presenting and writing. In other words, you may know how to write, but you don't know how to write right!

For students who need a lot of help with the mechanics of writing, the University offers several specialized resources:

- The Center for Writing and Communication
- The Center for International Language Services

IMPORTANT All students need to practice academic speaking and writing

The cornerstone of expertise is **deliberate practice**. To become an expert academic speaker and writer, you need to carefully think about what you are doing, how you are doing it, what the alternative ways are, and then practice the best way. It helps a lot if you get feedback from others.

How do you get feedback and the opportunity to practice? Most programs have informal meetings where students present, and many labs encourage practice talks. Make use of these opportunities! It takes **years** of practice to get really good at academic speaking, so keep practicing and study others as they present. This is particularly important for job talks, which are slightly different than any other talk you will give.

Summary

At this point, you should know **most** everything you need to be a successful IIS student.

Faculty

This chapter is dedicated to faculty. It assumes you've already read the Getting Started chapter. If you have not read that, go back and read it now! You might also want to read the Students chapter so you know what your students are supposed to know.

The IIS is primarily geared toward supporting faculty research. We do that by providing resources and services. These resources and services are designed to help you do pilot work for grants, submit grants, lead grants, and wind down grants.

Why do we focus so much on grants? While research does not need to be backed by a grant to be worthwhile, the IIS is almost entirely funded by grant activity through indirect cost recovery and salary buyout from its faculty lines. Without ongoing grant activity, the IIS would have to eliminate many of the resources we currently provide. Eventually, a lack of grant activity would probably lead to our staff being reassigned to other units. In other words, grants are an existential concern for the IIS, and we give priority to activities that support faculty through the grant life cycle.

Establishing yourself

At the end of the day, the IIS is a community of researchers above everything else. When you are new, or trying to move your research in a new direction, how do you establish yourself? The IIS provides several community building activities for faculty.

Cognitive Science Seminar

If you decide to lead it, the seminar is an excellent venue for giving other faculty a **deep** look at your research area. Even if you are just a participant, the seminar is the best way to help with your optics.

TIP If you don't attend the seminar regularly, people probably won't know who you are.

Leading the seminar can be very helpful for new faculty or faculty interested in pursuing a new research direction. The seminar lets you engage other IIS Affiliates for an entire semester - it is comparable to them sitting in a class on your research area for a whole semester. Because other Affiliates know your area better, your leading the seminar creates a natural context for collaborative projects and proposals.

The seminar traditionally has invited speakers for at least some of the public talks, and the IIS provides $5000 for these speaker's travel, hotel, and meals. The speakers can be faculty with whom you're already collaborating or would like to collaborate, so having invited speakers can also advance your research agenda. Speakers typically have a lunch with students, which presumably includes your students, and are taken to dinner by faculty, again presumably you and faculty who either already overlap in terms of research or who you'd like to recruit into that area.

The seminar is cross listed with three departments (PSYC/PHIL/COMP) and is a required course for the CogSci graduate certificate. Faculty leading the seminar often recruit students from other departments into their labs by getting to know them in the course. The format of the course is typically discussion based, with a final paper/project. These details can be changed as needed, but it is important to remember that because the class is cross listed, assignments need to be suitable for students from different backgrounds. One way to handle this is to give slightly different options for assignments depending on how the student registered for the class (i.e. under PSYC vs. under COMP).

Finally, the seminar is very popular with Affiliates and is usually booked a year in advance. So if you're thinking about leading a seminar, talk to the IIS Director, Alistair Windsor, right away.

Speed Date

The Speed Date has some of the same advantages for establishing yourself as the CogSci Seminar, but is a much smaller commitment. It can be useful for getting yourself introduced to the community if you haven't had a chance to lead a CogSci Seminar yet. Faculty sometimes get a little too focused on their 5-minute presentation; while that is important the reception afterwards is also important. The reception will give you a chance to talk to other faculty while your presentation is fresh on their minds. Also, if you have students with posters, their presentations to other faculty can be a great way to get your message out.

Getting on projects

The IIS has a tradition of being very welcoming to faculty. If there is an established project that you think you might be interested in, talk to the faculty leading the project and see if it would make sense to sit in. By sitting in on a project you will get to know other faculty's research and have opportunities for collaboration that might lead to publications.

Faculty resources

The IIS provides various resources, but with the exception of staff time and a portion of our graduate assistantships, availability of these resources need to be confirmed each year as funds become available. Unfortunately it is difficult for the IIS to declare availability by a specific date, for example, September 1st, because our indirect cost recovery funds are not released to us predictably.

Graduate assistantships

The IIS assigns GAs to our faculty lines (which often don't receive departmental support: Banerjee, Bidelman, Olney, Pavlik, Rus), then to IIS grantees whose external grants have ended and need "bridging" support between grants, and then to the IIS generally. The IIS has a strong tradition of faculty not abusing this arrangement; it is relatively common for our faculty lines to release their GA slots back to the pool when they have grants to support their students.

Each year in late spring, assuming that any slots are unfilled, the IIS issues a request for proposals to all Affiliates for 1-year GAs that start the following fall. The 1-year GAs are non-renewable, cover up to 9 hours of tuition (any additional hours must be approved on a case by case basis), and are paid $1200/month for pre-masters students and $1500/month for post-masters students.

Faculty proposals should be 1-2 pages in length and include (1) what the student would work on (2) how the student is currently funded (3) how the student's efforts will mesh with your own efforts to gain future external support.

Faculty who are not planning to submit are encouraged to volunteer to review the proposals.

Software development support

The IIS currently solicits proposals for software developer time on a six month cycle. Each cycle consists of two-month assignments to three different projects. After two months support will shift to the next project, so establishing sustainability/continuity on each project is key. That means that other personnel should work closely our senior software developer, Andrew Tackett, so that they can function independently after the two-month assignment ends. Faculty receiving assignments will be asked to provide feedback on Andrew Tackett's performance for annual evaluation purposes.

What can a software developer do? The items below are taken directly from Andrew Tackett's job description:

- Develops large and complex system projects and prototype implementations by participating in and overseeing the design, coding, testing, debugging and documentation activities of project staff.
- Establishes project plans and schedules and monitors progress, providing status reports as required.
- Conducts detailed systems analysis to define system scope, objectives and implementation approach. Entails attending research meetings and collaboratively working with research staff, faculty, and students.
- Develops system definition, architecture and detailed needs analysis including hardware and software recommendations based on system analysis.
- Maintains versioning, code repositories, and releases of grant-developed software to prevent past research from being lost or becoming obsolete due to hardware changes.
- Contributes text to project reports and technical papers.
- Establishes and implements user training programs, tutorials and other materials necessary to present and support research results.
- Maintains testing suites and runs regression tests on software to ensure that software modifications do not negatively impact past research.
- Responsible for technical presentations and system demonstrations (preparation, setup, and delivery) at conferences and funder events.
- Responsible for new developments and technologies by reading journals and other pertinent publications.
- Performs other duties as directed or required.

Travel support

Travel funds are provided for the following activities:

- To support conference travel where the faculty member is presenting a peer-reviewed publication (or equivalent by discipline)

- To support travel to sponsor meetings for the purpose of obtaining new funding (typically DoD)

Sponsor meetings that are part of existing grant obligations should be funded by those grants, and grant-related travel should be funded by grants.

The funding amounts are $2000/yr for Affiliates and $3000/yr for IIS faculty lines.

Travel authorization paperwork and reimbursement will be handled by the IIS.

Laptops for PIs

A laptop is provided to PIs with a new grant award, but no more than once every two years per PI.

Staff

See here

Grant life cycle

Obtaining and managing grants can be fairly complex. The IIS has a variety of resources and staff support to help. Probably the best way to think about grants is in terms of a life cycle.

1. Idea
2. Finding opportunities
3. Forming a team
4. Proposal
5. Award
6. Opening a Banner Fund
7. Spending appropriately
8. Covering overages
9. Fiscal reporting
10. Closing the account

Idea

It all starts with an idea. Ideas can come from many places, but at the IIS we have a particular orientation towards interdisciplinary research.

> Interdisciplinary research is a mode of research by teams or individuals that integrates information, data, techniques, tools, perspectives, concepts, and/or theories from two or more disciplines or bodies of specialized knowledge to advance fundamental understanding or to solve problems whose solutions are beyond the scope of a single discipline or area of research practice.
>
> — Committee on Facilitating Interdisciplinary Research, Facilitating interdisciplinary research. 2004. p. 2.

How do you have an interdisciplinary idea? Most likely by working with people from other disciplines, learning about those disciplines, and making the connection between your discipline and those disciplines. This should happen naturally as you participate in the IIS community. A complementary approach is to look for authentic problems in city, county, or state where you can make both a research contribution and a societal contribution.

Finding opportunities

There are many different types of grants:

- Federal grants
  - Department of Defense
  - Institute for Education Sciences
  - National Institutes of Health
  - National Science Foundation
  - **Any federal agency over a certain size is likely to have some kind of grant program**
- Industry grants
  - Microsoft
  - Amazon
  - **Industry usually funds grants only in areas relevant to them**
- Foundations
  - Toyota USA Foundation
  - John Templeton Foundation
  - Spencer Foundation
  - **Foundations often have specific causes that they fund**

Most sponsors have multiple opportunities at any given time, which means that there are literally hundreds or thousands of opportunities to navigate.

There are several strategies you can use to find potential sponsors:

Meet with {:pre-award-coordinator:}
Meet and discuss your area and interests so {:pre-award-coordinator:} can search existing

opportunities and forward future opportunities to you.

Talk to colleagues

Usually the funders of a research area are well established. Colleagues at the IIS or your department likely can make good recommendations.

Read acknowledgements

Most grants require acknowledgement in publications. When reading papers in your field, make a note of who funded the research. For federal grants there is often a grant ID mentioned that can be used to find a complete public record of the grant on the web.

Explore

Especially for the larger federal sponsors, it is worthwhile to spend some time exploring their opportunities at a high level so you know how they are structured. For example, NSF is hierarchically structured with funding programs under divisions under directorates.

Try a recommendation service

The university has been experimenting with various services that recommend grants to faculty based on their profiles, like GrantForward.

Forming a team

You may decide to form a team before the proposal stage or during the proposal stage. The advantage of forming a team beforehand (or alternatively working with faculty beforehand) is that you will have a better idea of each team members strengths and research interests. If you form a team during the proposal phase, you will have an extra layer of complexity in figuring out how to organize the team. This can be particularly complicated if you are working across institutions on a large project.

Proposal

It is very important that you disclose your intent to submit (even if it is tentative) to {:pre-award-coordinator:} as early as possible and more than a month before the submission deadline.

It is also important to understand that {:pre-award-coordinator:} does not have complete control over your proposal submission. {:pre-award-coordinator:}'s job is to:

- Read all the requirements of the solicitation
- Track, and to the extent possible, create all required proposal components
- Get approvals (particularly budget) from our Office of Sponsored Programs (OSP)
- Route a Proposal Summary Form through Cayuse to get signatures from chairs and deans

{:pre-award-coordinator:} does not submit the final proposal. When the proposal is finalized, {:pre-award-coordinator:} will notify OSP, and OSP will add any additional institutional information and submit the proposal. Therefore it is very important to have the proposal ready before the sponsor's deadline so that {:pre-award-coordinator:} can pass the proposal to OSP in time for them to do their job.

indirect costs, travel, effort, hires, allowable costs, cost share

Summary

At this point, you should know **most** everything you need to be a successful IIS Affiliate.

Git on the Server

At this point, you should be able to do most of the day-to-day tasks for which you'll be using Git. However, in order to do any collaboration in Git, you'll need to have a remote Git repository. Although you can technically push changes to and pull changes from individuals' repositories, doing so is discouraged because you can fairly easily confuse what they're working on if you're not careful. Furthermore, you want your collaborators to be able to access the repository even if your computer is offline – having a more reliable common repository is often useful. Therefore, the preferred method for collaborating with someone is to set up an intermediate repository that you both have access to, and push to and pull from that.

Running a Git server is fairly straightforward. First, you choose which protocols you want your server to communicate with. The first section of this chapter will cover the available protocols and the pros and cons of each. The next sections will explain some typical setups using those protocols and how to get your server running with them. Last, we'll go over a few hosted options, if you don't mind hosting your code on someone else's server and don't want to go through the hassle of setting up and maintaining your own server.

If you have no interest in running your own server, you can skip to the last section of the chapter to see some options for setting up a hosted account and then move on to the next chapter, where we discuss the various ins and outs of working in a distributed source control environment.

A remote repository is generally a *bare repository* – a Git repository that has no working directory. Because the repository is only used as a collaboration point, there is no reason to have a snapshot checked out on disk; it's just the Git data. In the simplest terms, a bare repository is the contents of your project's `.git` directory and nothing else.

The Protocols

Git can use four major protocols to transfer data: Local, HTTP, Secure Shell (SSH) and Git. Here we'll discuss what they are and in what basic circumstances you would want (or not want) to use them.

Local Protocol

The most basic is the *Local protocol*, in which the remote repository is in another directory on disk. This is often used if everyone on your team has access to a shared filesystem such as an NFS mount, or in the less likely case that everyone logs in to the same computer. The latter wouldn't be ideal, because all your code repository instances would reside on the same computer, making a catastrophic loss much more likely.

If you have a shared mounted filesystem, then you can clone, push to, and pull from a local file-based repository. To clone a repository like this or to add one as a remote to an existing project, use the path to the repository as the URL. For example, to clone a local repository, you can run something like this:

```
$ git clone /srv/git/project.git
```

Or you can do this:

```
$ git clone file:///srv/git/project.git
```

Git operates slightly differently if you explicitly specify `file://` at the beginning of the URL. If you just specify the path, Git tries to use hardlinks or directly copy the files it needs. If you specify `file://`, Git fires up the processes that it normally uses to transfer data over a network which is generally a lot less efficient method of transferring the data. The main reason to specify the `file://` prefix is if you want a clean copy of the repository with extraneous references or objects left out – generally after an import from another version-control system or something similar (see [_git_internals] for maintenance tasks). We'll use the normal path here because doing so is almost always faster.

To add a local repository to an existing Git project, you can run something like this:

```
$ git remote add local_proj /srv/git/project.git
```

Then, you can push to and pull from that remote as though you were doing so over a network.

The Pros

The pros of file-based repositories are that they're simple and they use existing file permissions and network access. If you already have a shared filesystem to which your whole team has access, setting up a repository is very easy. You stick the bare repository copy somewhere everyone has shared access to and set the read/write permissions as you would for any other shared directory. We'll discuss how to export a bare repository copy for this purpose in Git on the Server.

This is also a nice option for quickly grabbing work from someone else's working repository. If you and a co-worker are working on the same project and they want you to check something out, running a command like `git pull /home/john/project` is often easier than them pushing to a remote server and you pulling down.

The Cons

The cons of this method are that shared access is generally more difficult to set up and reach from multiple locations than basic network access. If you want to push from your laptop when you're at home, you have to mount the remote disk, which can be difficult and slow compared to network-based access.

It's important to mention that this isn't necessarily the fastest option if you're using a shared mount of some kind. A local repository is fast only if you have fast access to the data. A repository on NFS is often slower than the repository over SSH on the same server, allowing Git to run off local disks on each system.

Finally, this protocol does not protect the repository against accidental damage. Every user has full shell access to the "remote" directory, and there is nothing preventing them from changing or removing internal Git files and corrupting the repository.

The HTTP Protocols

Git can communicate over HTTP in two different modes. Prior to Git 1.6.6 there was only one way it could do this which was very simple and generally read-only. In version 1.6.6 a new, smarter protocol was introduced that involved Git being able to intelligently negotiate data transfer in a manner similar to how it does over SSH. In the last few years, this new HTTP protocol has become very popular since it's simpler for the user and smarter about how it communicates. The newer version is often referred to as the "Smart" HTTP protocol and the older way as "Dumb" HTTP. We'll cover the newer "smart" HTTP protocol first.

Smart HTTP

The "smart" HTTP protocol operates very similarly to the SSH or Git protocols but runs over standard HTTP/S ports and can use various HTTP authentication mechanisms, meaning it's often easier on the user than something like SSH, since you can use things like username/password authentication rather than having to set up SSH keys.

It has probably become the most popular way to use Git now, since it can be set up to both serve anonymously like the `git://` protocol, and can also be pushed over with authentication and encryption like the SSH protocol. Instead of having to set up different URLs for these things, you can now use a single URL for both. If you try to push and the repository requires authentication (which it normally should), the server can prompt for a username and password. The same goes for read access.

In fact, for services like GitHub, the URL you use to view the repository online (for example, "https://github.com/schacon/simplegit[]") is the same URL you can use to clone and, if you have access, push over.

Dumb HTTP

If the server does not respond with a Git HTTP smart service, the Git client will try to fall back to the simpler "dumb" HTTP protocol. The Dumb protocol expects the bare Git repository to be served like normal files from the web server. The beauty of the Dumb HTTP protocol is the simplicity of setting it up. Basically, all you have to do is put a bare Git repository under your HTTP document root and set up a specific `post-update` hook, and you're done (See [_git_hooks]). At that point, anyone who can access the web server under which you put the repository can also clone your repository. To allow read access to your repository over HTTP, do something like this:

```
$ cd /var/www/htdocs/
$ git clone --bare /path/to/git_project gitproject.git
$ cd gitproject.git
$ mv hooks/post-update.sample hooks/post-update
$ chmod a+x hooks/post-update
```

That's all. The `post-update` hook that comes with Git by default runs the appropriate command (`git update-server-info`) to make HTTP fetching and cloning work properly. This command is run when you push to this repository (over SSH perhaps); then, other people can clone via something like

```
$ git clone https://example.com/gitproject.git
```

In this particular case, we're using the `/var/www/htdocs` path that is common for Apache setups, but you can use any static web server – just put the bare repository in its path. The Git data is served as basic static files (see [_git_internals] for details about exactly how it's served).

Generally you would either choose to run a read/write Smart HTTP server or simply have the files accessible as read-only in the Dumb manner. It's rare to run a mix of the two services.

The Pros

We'll concentrate on the pros of the Smart version of the HTTP protocol.

The simplicity of having a single URL for all types of access and having the server prompt only when authentication is needed makes things very easy for the end user. Being able to authenticate with a username and password is also a big advantage over SSH, since users don't have to generate SSH keys locally and upload their public key to the server before being able to interact with it. For less sophisticated users, or users on systems where SSH is less common, this is a major advantage in usability. It is also a very fast and efficient protocol, similar to the SSH one.

You can also serve your repositories read-only over HTTPS, which means you can encrypt the content transfer; or you can go so far as to make the clients use specific signed SSL certificates.

Another nice thing is that HTTP/S are such commonly used protocols that corporate firewalls are often set up to allow traffic through these ports.

The Cons

Git over HTTP/S can be a little more tricky to set up compared to SSH on some servers. Other than that, there is very little advantage that other protocols have over the "Smart" HTTP protocol for serving Git.

If you're using HTTP for authenticated pushing, providing your credentials is sometimes more complicated than using keys over SSH. There are however several credential caching tools you can use, including Keychain access on OSX and Credential Manager on Windows, to make this pretty painless. Read [_credential_caching] to see how to set up secure HTTP password caching on your system.

The SSH Protocol

A common transport protocol for Git when self-hosting is over SSH. This is because SSH access to servers is already set up in most places – and if it isn't, it's easy to do. SSH is also an authenticated network protocol; and because it's ubiquitous, it's generally easy to set up and use.

To clone a Git repository over SSH, you can specify ssh:// URL like this:

```
$ git clone ssh://user@server/project.git
```

Or you can use the shorter scp-like syntax for the SSH protocol:

```
$ git clone user@server:project.git
```

You can also not specify a user, and Git assumes the user you're currently logged in as.

The Pros

The pros of using SSH are many. First, SSH is relatively easy to set up – SSH daemons are commonplace, many network admins have experience with them, and many OS distributions are set up with them or have tools to manage them. Next, access over SSH is secure – all data transfer is encrypted and authenticated. Last, like the HTTP/S, Git and Local protocols, SSH is efficient, making the data as compact as possible before transferring it.

The Cons

The negative aspect of SSH is that you can't serve anonymous access of your repository over it. People must have access to your machine over SSH to access it, even in a read-only capacity, which doesn't make SSH access conducive to open source projects. If you're using it only within your corporate network, SSH may be the only protocol you need to deal with. If you want to allow anonymous read-only access to your projects and also want to use SSH, you'll have to set up SSH for you to push over but something else for others to fetch over.

The Git Protocol

Next is the Git protocol. This is a special daemon that comes packaged with Git; it listens on a dedicated port (9418) that provides a service similar to the SSH protocol, but with absolutely no authentication. In order for a repository to be served over the Git protocol, you must create the `git-daemon-export-ok` file – the daemon won't serve a repository without that file in it – but other than that there is no security. Either the Git repository is available for everyone to clone or it isn't. This means that there is generally no pushing over this protocol. You can enable push access; but given the lack of authentication, if you turn on push access, anyone on the internet who finds your project's URL could push to your project. Suffice it to say that this is rare.

The Pros

The Git protocol is often the fastest network transfer protocol available. If you're serving a lot of traffic for a public project or serving a very large project that doesn't require user authentication for read access, it's likely that you'll want to set up a Git daemon to serve your project. It uses the same data-transfer mechanism as the SSH protocol but without the encryption and authentication overhead.

The Cons

The downside of the Git protocol is the lack of authentication. It's generally undesirable for the Git protocol to be the only access to your project. Generally, you'll pair it with SSH or HTTPS access for the few developers who have push (write) access and have everyone else use `git://` for read-only access. It's also probably the most difficult protocol to set up. It must run its own daemon, which requires xinetd configuration or the like, which isn't always a walk in the park. It also requires firewall access to port 9418, which isn't a standard port that corporate firewalls always allow. Behind big corporate firewalls, this obscure port is commonly blocked.

Getting Git on a Server

Now we'll cover setting up a Git service running these protocols on your own server.

NOTE Here we'll be demonstrating the commands and steps needed to do basic, simplified installations on a Linux based server, though it's also possible to run these services on Mac or Windows servers. Actually setting up a production server within your infrastructure will certainly entail differences in security measures or operating system tools, but hopefully this will give you the general idea of what's involved.

In order to initially set up any Git server, you have to export an existing repository into a new bare repository – a repository that doesn't contain a working directory. This is generally straightforward to do. In order to clone your repository to create a new bare repository, you run the clone command with the `--bare` option. By convention, bare repository directories end in `.git`, like so:

```
$ git clone --bare my_project my_project.git
Cloning into bare repository 'my_project.git'...
done.
```

You should now have a copy of the Git directory data in your `my_project.git` directory.

This is roughly equivalent to something like

```
$ cp -Rf my_project/.git my_project.git
```

There are a couple of minor differences in the configuration file; but for your purpose, this is close to the same thing. It takes the Git repository by itself, without a working directory, and creates a directory specifically for it alone.

Putting the Bare Repository on a Server

Now that you have a bare copy of your repository, all you need to do is put it on a server and set up your protocols. Let's say you've set up a server called `git.example.com` that you have SSH access to, and you want to store all your Git repositories under the `/srv/git` directory. Assuming that `/srv/git` exists on that server, you can set up your new repository by copying your bare repository over:

```
$ scp -r my_project.git user@git.example.com:/srv/git
```

At this point, other users who have SSH access to the same server which has read-access to the `/srv/git` directory can clone your repository by running

```
$ git clone user@git.example.com:/srv/git/my_project.git
```

If a user SSHs into a server and has write access to the `/srv/git/my_project.git` directory, they will also automatically have push access.

Git will automatically add group write permissions to a repository properly if you run the `git init` command with the `--shared` option.

```
$ ssh user@git.example.com
$ cd /srv/git/my_project.git
$ git init --bare --shared
```

You see how easy it is to take a Git repository, create a bare version, and place it on a server to which you and your collaborators have SSH access. Now you're ready to collaborate on the same project.

It's important to note that this is literally all you need to do to run a useful Git server to which several people have access – just add SSH-able accounts on a server, and stick a bare repository somewhere that all those users have read and write access to. You're ready to go – nothing else needed.

In the next few sections, you'll see how to expand to more sophisticated setups. This discussion will include not having to create user accounts for each user, adding public read access to repositories, setting up web UIs and more. However, keep in mind that to collaborate with a couple of people on a private project, all you *need* is an SSH server and a bare repository.

Small Setups

If you're a small outfit or are just trying out Git in your organization and have only a few developers, things can be simple for you. One of the most complicated aspects of setting up a Git server is user management. If you want some repositories to be read-only to certain users and read/write to others, access and permissions can be a bit more difficult to arrange.

SSH Access

If you have a server to which all your developers already have SSH access, it's generally easiest to set up your first repository there, because you have to do almost no work (as we covered in the last section). If you want more complex access control type permissions on your repositories, you can handle them with the normal filesystem permissions of the operating system your server runs.

If you want to place your repositories on a server that doesn't have accounts for everyone on your team whom you want to have write access, then you must set up SSH access for them. We assume that if you have a server with which to do this, you already have an SSH server installed, and that's how you're accessing the server.

There are a few ways you can give access to everyone on your team. The first is to set up accounts for everybody, which is straightforward but can be cumbersome. You may not want to run `adduser` and set temporary passwords for every user.

A second method is to create a single *git* user on the machine, ask every user who is to have write access to send you an SSH public key, and add that key to the `~/.ssh/authorized_keys` file of your new *git* user. At that point, everyone will be able to access that machine via the *git* user. This doesn't affect the commit data in any way – the SSH user you connect as doesn't affect the commits you've recorded.

Another way to do it is to have your SSH server authenticate from an LDAP server or some other centralized authentication source that you may already have set up. As long as each user can get shell access on the machine, any SSH authentication mechanism you can think of should work.

Generating Your SSH Public Key

That being said, many Git servers authenticate using SSH public keys. In order to provide a public key, each user in your system must generate one if they don't already have one. This process is similar across all operating systems. First, you should check to make sure you don't already have a key. By default, a user's SSH keys are stored in that user's `~/.ssh` directory. You can easily check to see if you have a key already by going to that directory and listing the contents:

```
$ cd ~/.ssh
$ ls
authorized_keys2  id_dsa       known_hosts
config            id_dsa.pub
```

You're looking for a pair of files named something like `id_dsa` or `id_rsa` and a matching file with a `.pub` extension. The `.pub` file is your public key, and the other file is your private key. If you don't have these files (or you don't even have a `.ssh` directory), you can create them by running a program called `ssh-keygen`, which is provided with the SSH package on Linux/Mac systems and comes with Git for Windows:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/schacon/.ssh/id_rsa):
Created directory '/home/schacon/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/schacon/.ssh/id_rsa.
Your public key has been saved in /home/schacon/.ssh/id_rsa.pub.
The key fingerprint is:
d0:82:24:8e:d7:f1:bb:9b:33:53:96:93:49:da:9b:e3 schacon@mylaptop.local
```

First it confirms where you want to save the key (`.ssh/id_rsa`), and then it asks twice for a passphrase, which you can leave empty if you don't want to type a password when you use the key.

Now, each user that does this has to send their public key to you or whoever is administrating the Git server (assuming you're using an SSH server setup that requires public keys). All they have to do is copy the contents of the `.pub` file and email it. The public keys look something like this:

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAklOUpkDHrfHY17SbrmTIpNLTGK9Tjom/BWDSU
GPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEVf4h9lFX5QVkbPppSwg0cda3
Pbv7kOdJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhXpHAZsMciLq8V6RjsNAQwdsdMFvSlVK/7XA
t3FaoJoAsncM1Q9x5+3V0Ww68/eIFmb1zuUFljQJKprrX88XypNDvjYNby6vw/Pb0rwert/En
mZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so1d01QraTlMqVSsbx
NrRFi9wrf+M7Q== schacon@mylaptop.local
```

For a more in-depth tutorial on creating an SSH key on multiple operating systems, see the GitHub guide on SSH keys at https://help.github.com/articles/generating-ssh-keys.

Setting Up the Server

Let's walk through setting up SSH access on the server side. In this example, you'll use the `authorized_keys` method for authenticating your users. We also assume you're running a standard Linux distribution like Ubuntu. First, you create a `git` user and a `.ssh` directory for that user.

```
$ sudo adduser git
$ su git
$ cd
$ mkdir .ssh && chmod 700 .ssh
$ touch .ssh/authorized_keys && chmod 600 .ssh/authorized_keys
```

Next, you need to add some developer SSH public keys to the `authorized_keys` file for the `git` user. Let's assume you have some trusted public keys and have saved them to temporary files. Again, the public keys look something like this:

```
$ cat /tmp/id_rsa.john.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCB007n/ww+ouN4gSLKssMxXnBOvf9LGt4L
ojG6rs6hPB09j9R/T17/x4lhJA0F3FR1rP6kYBRsWj2aThGw6HXLm9/5zytK6Ztg3RPKK+4k
Yjh6541NYsnEAZuXz0jTTyAUfrtU3Z5E003C4oxOj6H0rfIF1kKI9MAQLMdpGW1GYEIgS9Ez
Sdfd8AcCIicTDWbqLAcU4UpkaX8KyGlLwsNuuGztobF8m72ALC/nLF6JLtPofwFBlgc+myiv
O7TCUSBdLQlgMVOFq1I2uPWQOkOWQAHukEOmfjy2jctxSDBQ220ymjaNsHT4kgtZg2AYYgPq
dAv8JggJICUvax2T9va5 gsg-keypair
```

You just append them to the `git` user's `authorized_keys` file in its `.ssh` directory:

```
$ cat /tmp/id_rsa.john.pub >> ~/.ssh/authorized_keys
$ cat /tmp/id_rsa.josie.pub >> ~/.ssh/authorized_keys
$ cat /tmp/id_rsa.jessica.pub >> ~/.ssh/authorized_keys
```

Now, you can set up an empty repository for them by running `git init` with the `--bare` option, which initializes the repository without a working directory:

```
$ cd /srv/git
$ mkdir project.git
$ cd project.git
$ git init --bare
Initialized empty Git repository in /srv/git/project.git/
```

Then, John, Josie, or Jessica can push the first version of their project into that repository by adding it as a remote and pushing up a branch. Note that someone must shell onto the machine and create a bare repository every time you want to add a project. Let’s use `gitserver` as the hostname of the server on which you’ve set up your `git` user and repository. If you’re running it internally, and you set up DNS for `gitserver` to point to that server, then you can use the commands pretty much as is (assuming that `myproject` is an existing project with files in it):

```
# on John's computer
$ cd myproject
$ git init
$ git add .
$ git commit -m 'initial commit'
$ git remote add origin git@gitserver:/srv/git/project.git
$ git push origin master
```

At this point, the others can clone it down and push changes back up just as easily:

```
$ git clone git@gitserver:/srv/git/project.git
$ cd project
$ vim README
$ git commit -am 'fix for the README file'
$ git push origin master
```

With this method, you can quickly get a read/write Git server up and running for a handful of developers.

You should note that currently all these users can also log into the server and get a shell as the `git` user. If you want to restrict that, you will have to change the shell to something else in the `passwd` file.

You can easily restrict the `git` user to only doing Git activities with a limited shell tool called `git-shell` that comes with Git. If you set this as your `git` user’s login shell, then the `git` user can’t have normal shell access to your server. To use this, specify `git-shell` instead of bash or csh for your user’s login shell. To do so, you must first add `git-shell` to `/etc/shells` if it’s not already there:

```
$ cat /etc/shells   # see if `git-shell` is already in there.  If not...
$ which git-shell   # make sure git-shell is installed on your system.
$ sudo vim /etc/shells  # and add the path to git-shell from last command
```

Now you can edit the shell for a user using `chsh <username> -s <shell>`:

```
$ sudo chsh git -s $(which git-shell)
```

Now, the `git` user can only use the SSH connection to push and pull Git repositories and can't shell onto the machine. If you try, you'll see a login rejection like this:

```
$ ssh git@gitserver
fatal: Interactive git shell is not enabled.
hint: ~/git-shell-commands should exist and have read and execute access.
Connection to gitserver closed.
```

Now Git network commands will still work just fine but the users won't be able to get a shell. As the output states, you can also set up a directory in the `git` user's home directory that customizes the `git-shell` command a bit. For instance, you can restrict the Git commands that the server will accept or you can customize the message that users see if they try to SSH in like that. Run `git help shell` for more information on customizing the shell.

Git Daemon

Next we'll set up a daemon serving repositories over the "Git" protocol. This is common choice for fast, unauthenticated access to your Git data. Remember that since it's not an authenticated service, anything you serve over this protocol is public within its network.

If you're running this on a server outside your firewall, it should only be used for projects that are publicly visible to the world. If the server you're running it on is inside your firewall, you might use it for projects that a large number of people or computers (continuous integration or build servers) have read-only access to, when you don't want to have to add an SSH key for each.

In any case, the Git protocol is relatively easy to set up. Basically, you need to run this command in a daemonized manner:

```
$ git daemon --reuseaddr --base-path=/srv/git/ /srv/git/
```

`--reuseaddr` allows the server to restart without waiting for old connections to time out, the `--base-path` option allows people to clone projects without specifying the entire path, and the path at the end tells the Git daemon where to look for repositories to export. If you're running a firewall, you'll also need to punch a hole in it at port 9418 on the box you're setting this up on.

You can daemonize this process a number of ways, depending on the operating system you're running. On an Ubuntu machine, you can use an Upstart script. So, in the following file

```
/etc/init/local-git-daemon.conf
```

you put this script:

```
start on startup
stop on shutdown
exec /usr/bin/git daemon \
    --user=git --group=git \
    --reuseaddr \
    --base-path=/srv/git/ \
    /srv/git/
respawn
```

For security reasons, it is strongly encouraged to have this daemon run as a user with read-only permissions to the repositories – you can easily do this by creating a new user *git-ro* and running the daemon as them. For the sake of simplicity we'll simply run it as the same *git* user that `git-shell` is running as.

When you restart your machine, your Git daemon will start automatically and respawn if it goes down. To get it running without having to reboot, you can run this:

```
$ initctl start local-git-daemon
```

On other systems, you may want to use `xinetd`, a script in your `sysvinit` system, or something else – as long as you get that command daemonized and watched somehow.

Next, you have to tell Git which repositories to allow unauthenticated Git server-based access to. You can do this in each repository by creating a file named `git-daemon-export-ok`.

```
$ cd /path/to/project.git
$ touch git-daemon-export-ok
```

The presence of that file tells Git that it's OK to serve this project without authentication.

Smart HTTP

We now have authenticated access through SSH and unauthenticated access through `git://`, but there is also a protocol that can do both at the same time. Setting up Smart HTTP is basically just enabling a CGI script that is provided with Git called `git-http-backend` on the server. This CGI will read the path and headers sent by a `git fetch` or `git push` to an HTTP URL and determine if the client can communicate over HTTP (which is true for any client since version 1.6.6). If the CGI sees that the client is smart, it will communicate smartly with it, otherwise it will fall back to the dumb behavior (so it is backward compatible for reads with older clients).

Let's walk through a very basic setup. We'll set this up with Apache as the CGI server. If you don't have Apache setup, you can do so on a Linux box with something like this:

```
$ sudo apt-get install apache2 apache2-utils
$ a2enmod cgi alias env
```

This also enables the `mod_cgi`, `mod_alias`, and `mod_env` modules, which are all needed for this to work properly.

You'll also need to set the Unix user group of the `/srv/git` directories to `www-data` so your web server can read- and write-access the repositories, because the Apache instance running the CGI script will (by default) be running as that user:

```
$ chgrp -R www-data /srv/git
```

Next we need to add some things to the Apache configuration to run the `git-http-backend` as the handler for anything coming into the `/git` path of your web server.

```
SetEnv GIT_PROJECT_ROOT /srv/git
SetEnv GIT_HTTP_EXPORT_ALL
ScriptAlias /git/ /usr/lib/git-core/git-http-backend/
```

If you leave out `GIT_HTTP_EXPORT_ALL` environment variable, then Git will only serve to unauthenticated clients the repositories with the `git-daemon-export-ok` file in them, just like the Git daemon did.

Finally you'll want to tell Apache to allow requests to `git-http-backend` and make writes be authenticated somehow, possibly with an Auth block like this:

```
<Files "git-http-backend">
    AuthType Basic
    AuthName "Git Access"
    AuthUserFile /srv/git/.htpasswd
    Require expr !(%{QUERY_STRING} -strmatch '*service=git-receive-pack*' ||
%{REQUEST_URI} =~ m#/git-receive-pack$#)
    Require valid-user
</Files>
```

That will require you to create a `.htpasswd` file containing the passwords of all the valid users. Here is an example of adding a “schacon” user to the file:

```
$ htpasswd -c /srv/git/.htpasswd schacon
```

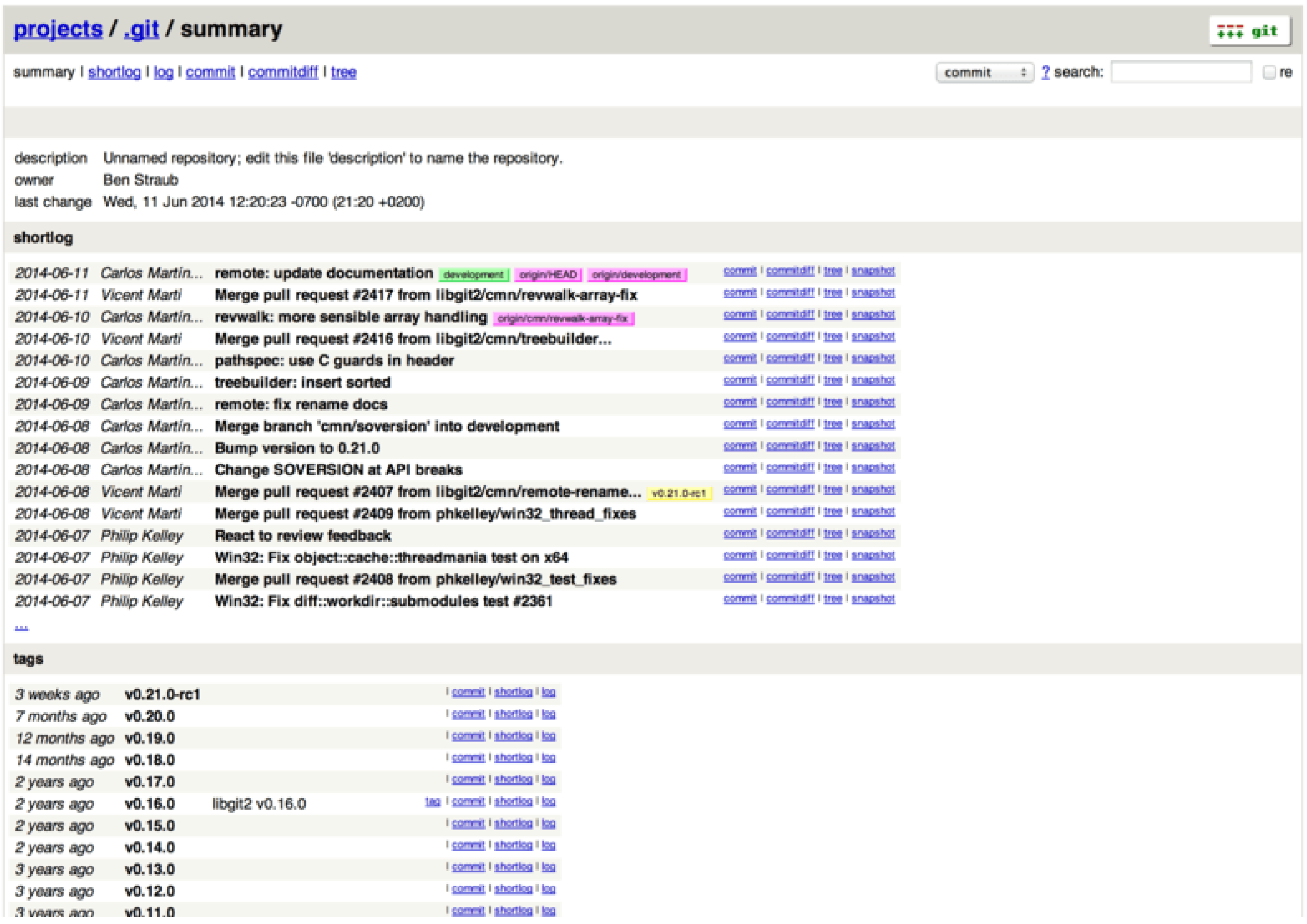
There are tons of ways to have Apache authenticate users, you'll have to choose and implement one of them. This is just the simplest example we could come up with. You'll also almost certainly want to set this up over SSL so all this data is encrypted.

We don't want to go too far down the rabbit hole of Apache configuration specifics, since you could well be using a different server or have different authentication needs. The idea is that Git comes with a CGI called `git-http-backend` that when invoked will do all the negotiation to send and receive data over HTTP. It does not implement any authentication itself, but that can easily be controlled at the layer of the web server that invokes it. You can do this with nearly any CGI-capable web server, so go with the one that you know best.

NOTE For more information on configuring authentication in Apache, check out the Apache docs here: http://httpd.apache.org/docs/current/howto/auth.html

GitWeb

Now that you have basic read/write and read-only access to your project, you may want to set up a simple web-based visualizer. Git comes with a CGI script called GitWeb that is sometimes used for this.



projects / .git / summary

summary | shortlog | log | commit | commitdiff | tree

description	Unnamed repository; edit this file 'description' to name the repository.
owner	Ben Straub
last change	Wed, 11 Jun 2014 12:20:23 -0700 (21:20 +0200)

shortlog

2014-06-11	Carlos Martín...	remote: update documentation development origin/HEAD origin/development	commit | commitdiff | tree | snapshot
2014-06-11	Vicent Marti	Merge pull request #2417 from libgit2/cmn/revwalk-array-fix	commit | commitdiff | tree | snapshot
2014-06-10	Carlos Martín...	revwalk: more sensible array handling origin/cmn/revwalk-array-fix	commit | commitdiff | tree | snapshot
2014-06-10	Vicent Marti	Merge pull request #2416 from libgit2/cmn/treebuilder...	commit | commitdiff | tree | snapshot
2014-06-10	Carlos Martín...	pathspec: use C guards in header	commit | commitdiff | tree | snapshot
2014-06-09	Carlos Martín...	treebuilder: insert sorted	commit | commitdiff | tree | snapshot
2014-06-09	Carlos Martín...	remote: fix rename docs	commit | commitdiff | tree | snapshot
2014-06-08	Carlos Martín...	Merge branch 'cmn/soversion' into development	commit | commitdiff | tree | snapshot
2014-06-08	Carlos Martín...	Bump version to 0.21.0	commit | commitdiff | tree | snapshot
2014-06-08	Carlos Martín...	Change SOVERSION at API breaks	commit | commitdiff | tree | snapshot
2014-06-08	Vicent Marti	Merge pull request #2407 from libgit2/cmn/remote-rename... v0.21.0-rc1	commit | commitdiff | tree | snapshot
2014-06-08	Vicent Marti	Merge pull request #2409 from phkelley/win32_thread_fixes	commit | commitdiff | tree | snapshot
2014-06-07	Philip Kelley	React to review feedback	commit | commitdiff | tree | snapshot
2014-06-07	Philip Kelley	Win32: Fix object::cache::threadmania test on x64	commit | commitdiff | tree | snapshot
2014-06-07	Philip Kelley	Merge pull request #2408 from phkelley/win32_test_fixes	commit | commitdiff | tree | snapshot
2014-06-07	Philip Kelley	Win32: Fix diff::workdir::submodules test #2361	commit | commitdiff | tree | snapshot

...

tags

3 weeks ago	v0.21.0-rc1		| commit | shortlog | log
7 months ago	v0.20.0		| commit | shortlog | log
12 months ago	v0.19.0		| commit | shortlog | log
14 months ago	v0.18.0		| commit | shortlog | log
2 years ago	v0.17.0		| commit | shortlog | log
2 years ago	v0.16.0	libgit2 v0.16.0	tag | commit | shortlog | log
2 years ago	v0.15.0		| commit | shortlog | log
2 years ago	v0.14.0		| commit | shortlog | log
3 years ago	v0.13.0		| commit | shortlog | log
3 years ago	v0.12.0		| commit | shortlog | log
3 years ago	v0.11.0		| commit | shortlog | log

Figure 1. The GitWeb web-based user interface.

If you want to check out what GitWeb would look like for your project, Git comes with a command to fire up a temporary instance if you have a lightweight server on your system like `lighttpd` or `webrick`. On Linux machines, `lighttpd` is often installed, so you may be able to get it to run by typing `git instaweb` in your project directory. If you're running a Mac, Leopard comes preinstalled with Ruby, so `webrick` may be your best bet. To start `instaweb` with a non-lighttpd handler, you can run it with the `--httpd` option.

```
$ git instaweb --httpd=webrick
[2009-02-21 10:02:21] INFO  WEBrick 1.3.1
[2009-02-21 10:02:21] INFO  ruby 1.8.6 (2008-03-03) [universal-darwin9.0]
```

That starts up an HTTPD server on port 1234 and then automatically starts a web browser that opens on that page. It's pretty easy on your part. When you're done and want to shut down the server, you can run the same command with the `--stop` option:

```
$ git instaweb --httpd=webrick --stop
```

If you want to run the web interface on a server all the time for your team or for an open source project you're hosting, you'll need to set up the CGI script to be served by your normal web server. Some Linux distributions have a `gitweb` package that you may be able to install via `apt` or `yum`, so you may want to try that first. We'll walk through installing GitWeb manually very quickly. First, you need to get the Git source code, which GitWeb comes with, and generate the custom CGI script:

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
$ cd git/
$ make GITWEB_PROJECTROOT="/srv/git" prefix=/usr gitweb
    SUBDIR gitweb
    SUBDIR ../
make[2]: `GIT-VERSION-FILE' is up to date.
    GEN gitweb.cgi
    GEN static/gitweb.js
$ sudo cp -Rf gitweb /var/www/
```

Notice that you have to tell the command where to find your Git repositories with the `GITWEB_PROJECTROOT` variable. Now, you need to make Apache use CGI for that script, for which you can add a VirtualHost:

```
<VirtualHost *:80>
    ServerName gitserver
    DocumentRoot /var/www/gitweb
    <Directory /var/www/gitweb>
        Options ExecCGI +FollowSymLinks +SymLinksIfOwnerMatch
        AllowOverride All
        order allow,deny
        Allow from all
        AddHandler cgi-script cgi
        DirectoryIndex gitweb.cgi
    </Directory>
</VirtualHost>
```

Again, GitWeb can be served with any CGI or Perl capable web server; if you prefer to use something else, it shouldn't be difficult to set up. At this point, you should be able to visit `http://gitserver/` to view your repositories online.

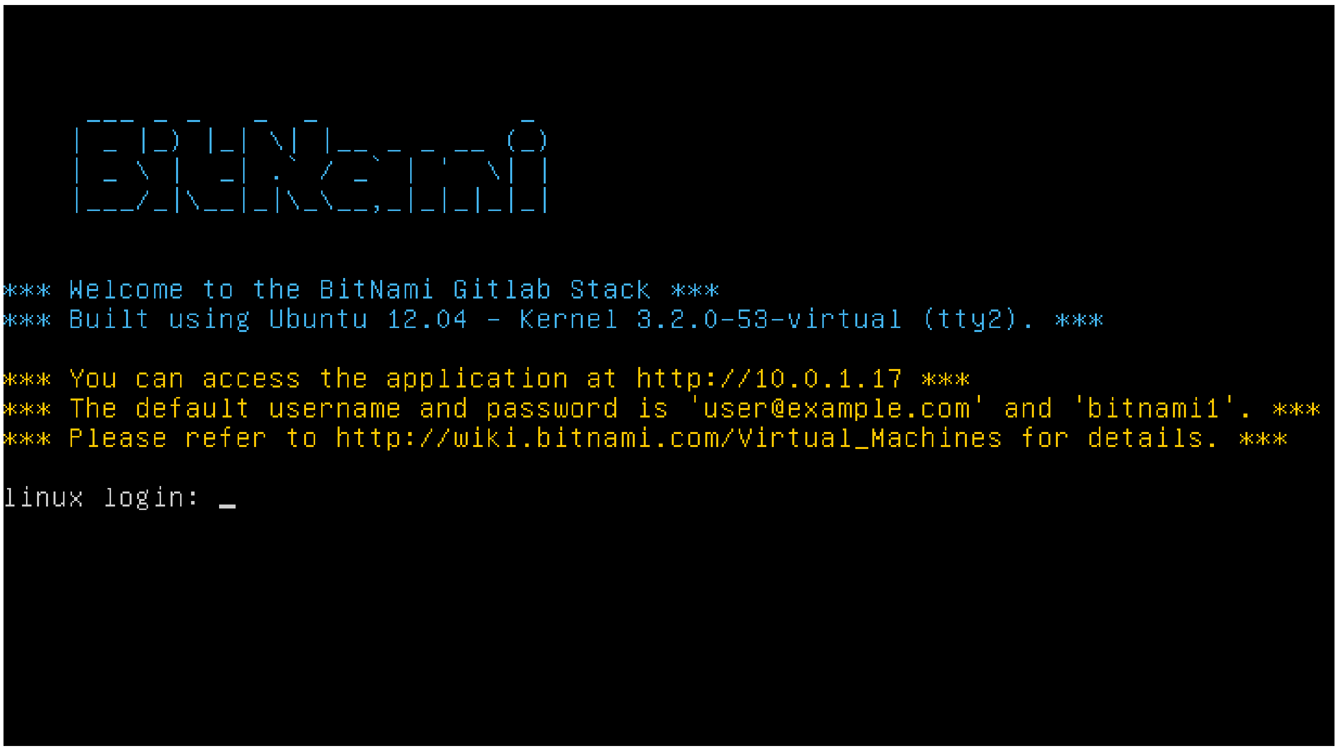
GitLab

GitWeb is pretty simplistic though. If you're looking for a more modern, fully featured Git server, there are some several open source solutions out there that you can install instead. As GitLab is one of the more popular ones, we'll cover installing and using it as an example. This is a bit more complex than the GitWeb option and likely requires more maintenance, but it is a much more fully featured option.

Installation

GitLab is a database-backed web application, so its installation is a bit more involved than some other Git servers. Fortunately, this process is very well-documented and supported.

There are a few methods you can pursue to install GitLab. To get something up and running quickly, you can download a virtual machine image or a one-click installer from <a href="https://bitnami.com/stack/gitlab" class="bare">https://bitnami.com/stack/gitlab</a>, and tweak the configuration to match your particular environment.<a name="__indexterm-47232888177020" type="indexterm"></a> One nice touch Bitnami has included is the login screen (accessed by typing alt-&rarr;); it tells you the IP address and default username and password for the installed GitLab.



```
*** Welcome to the BitNami Gitlab Stack ***
*** Built using Ubuntu 12.04 - Kernel 3.2.0-53-virtual (tty2). ***

*** You can access the application at http://10.0.1.17 ***
*** The default username and password is 'user@example.com' and 'bitnami1'. ***
*** Please refer to http://wiki.bitnami.com/Virtual_Machines for details. ***

linux login: _
```

Figure 2. The Bitnami GitLab virtual machine login screen.

For anything else, follow the guidance in the GitLab Community Edition readme, which can be found at https://gitlab.com/gitlab-org/gitlab-ce/tree/master. There you'll find assistance for installing GitLab using Chef recipes, a virtual machine on Digital Ocean, and RPM and DEB packages (which, as of this writing, are in beta). There's also "unofficial" guidance on getting GitLab running with non-standard operating systems and databases, a fully-manual installation script, and many other topics.

Administration

GitLab's administration interface is accessed over the web. Simply point your browser to the hostname or IP address where GitLab is installed, and log in as an admin user. The default username is `admin@local.host`, and the default password is `5iveL!fe` (which you will be prompted to change as soon as you enter it). Once logged in, click the “Admin area” icon in the menu at the top right.

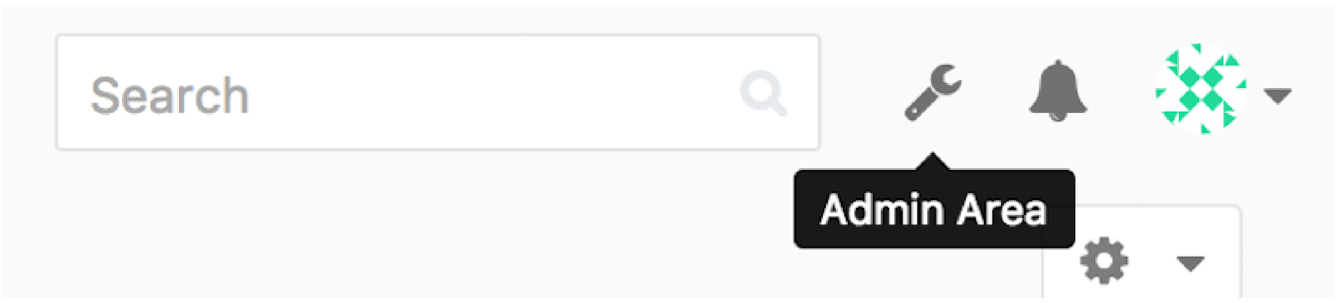
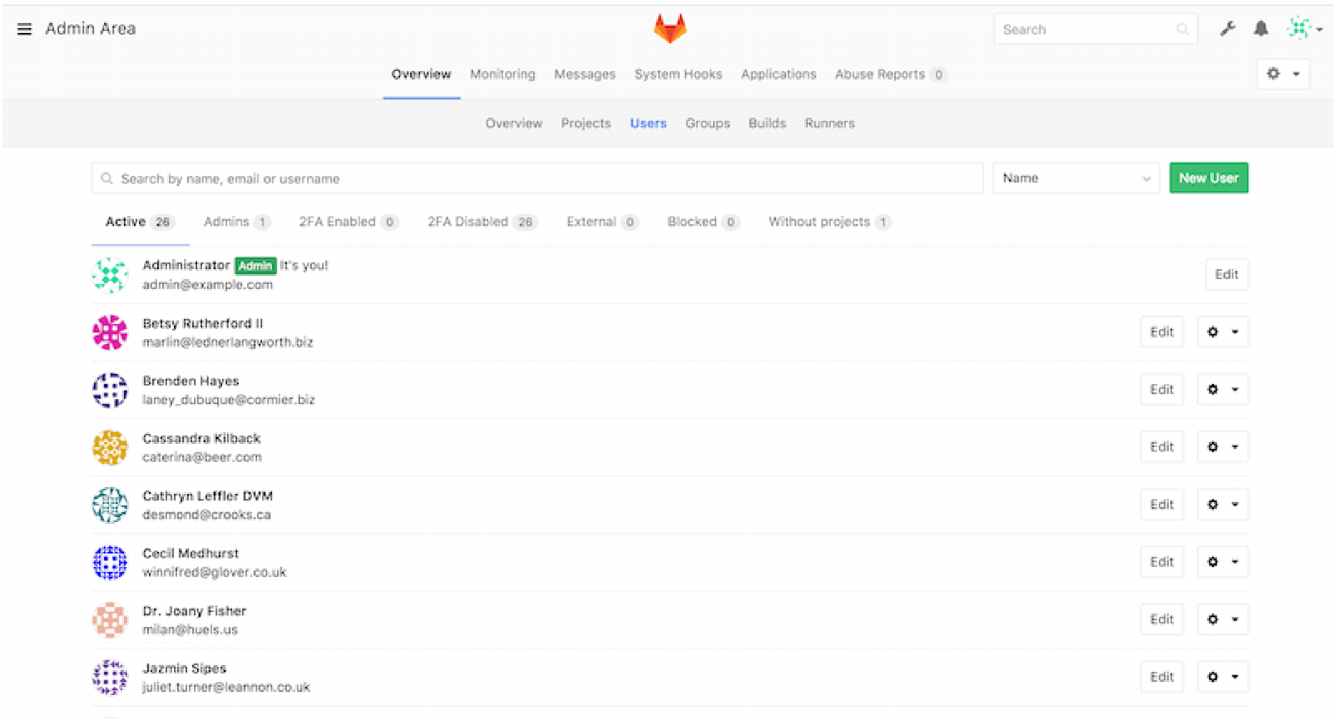


Figure 3. The “Admin area” item in the GitLab menu.

Users

Users in GitLab are accounts that correspond to people. User accounts don’t have a lot of complexity; mainly it’s a collection of personal information attached to login data. Each user account comes with a **namespace**, which is a logical grouping of projects that belong to that user. If the user `jane` had a project named `project`, that project’s url would be http://server/jane/project.



Administrator Admin It's you! admin@example.com	Edit
Betsy Rutherford II marlin@lednerlangworth.biz	Edit
Brenden Hayes laney_dubuque@cormier.biz	Edit
Cassandra Kilback caterina@beer.com	Edit
Cathryn Leffler DVM desmond@crooks.ca	Edit
Cecil Medhurst winnifred@glover.co.uk	Edit
Dr. Joany Fisher milan@huels.us	Edit
Jazmin Sipes juliet.turner@leannon.co.uk	Edit

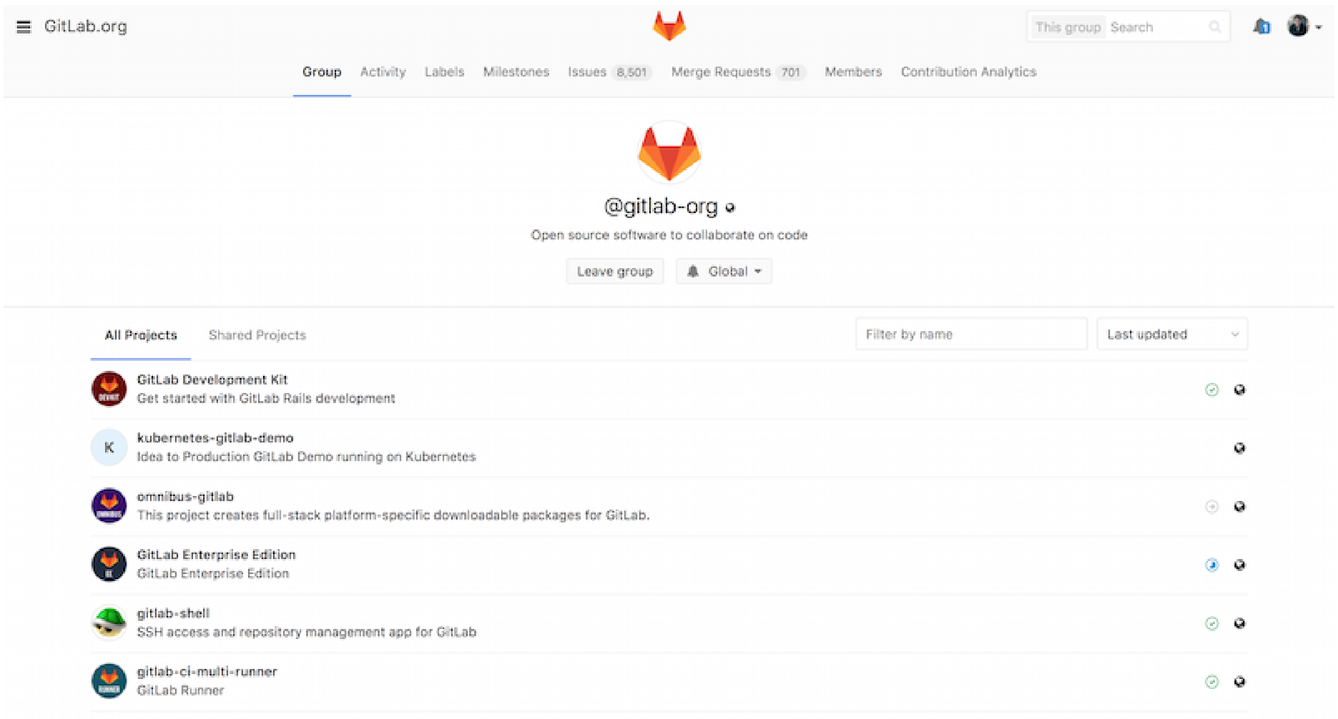
Figure 4. The GitLab user administration screen.

Removing a user can be done in two ways. “Blocking” a user prevents them from logging into the GitLab instance, but all of the data under that user’s namespace will be preserved, and commits signed with that user’s email address will still link back to their profile.

“Destroying” a user, on the other hand, completely removes them from the database and filesystem. All projects and data in their namespace is removed, and any groups they own will also be removed. This is obviously a much more permanent and destructive action, and its uses are rare.

Groups

A GitLab group is an assemblage of projects, along with data about how users can access those projects. Each group has a project namespace (the same way that users do), so if the group `training` has a project `materials`, its url would be http://server/training/materials.



@gitlab-org

Open source software to collaborate on code

GitLab Development Kit
Get started with GitLab Rails development

kubernetes-gitlab-demo
Idea to Production GitLab Demo running on Kubernetes

omnibus-gitlab
This project creates full-stack platform-specific downloadable packages for GitLab.

GitLab Enterprise Edition
GitLab Enterprise Edition

gitlab-shell
SSH access and repository management app for GitLab

gitlab-ci-multi-runner
GitLab Runner

Figure 5. The GitLab group administration screen.

Each group is associated with a number of users, each of which has a level of permissions for the group's projects and the group itself. These range from "Guest" (issues and chat only) to "Owner" (full control of the group, its members, and its projects). The types of permissions are too numerous to list here, but GitLab has a helpful link on the administration screen.

Projects

A GitLab project roughly corresponds to a single Git repository. Every project belongs to a single namespace, either a user or a group. If the project belongs to a user, the owner of the project has direct control over who has access to the project; if the project belongs to a group, the group's user-level permissions will also take effect.

Every project also has a visibility level, which controls who has read access to that project's pages and repository. If a project is *Private*, the project's owner must explicitly grant access to specific users. An *Internal* project is visible to any logged-in user, and a *Public* project is visible to anyone. Note that this controls both `git fetch` access as well as access to the web UI for that project.

Hooks

GitLab includes support for hooks, both at a project or system level. For either of these, the GitLab server will perform an HTTP POST with some descriptive JSON whenever relevant events occur. This is a great way to connect your Git repositories and GitLab instance to the rest of your development automation, such as CI servers, chat rooms, or deployment tools.

Basic Usage

The first thing you'll want to do with GitLab is create a new project. This is accomplished by clicking the "+" icon on the toolbar. You'll be asked for the project's name, which namespace it should belong to, and what its visibility level should be. Most of what you specify here isn't permanent, and can be re-adjusted later through the settings interface. Click "Create Project", and you're done.

Once the project exists, you'll probably want to connect it with a local Git repository. Each project is accessible over HTTPS or SSH, either of which can be used to configure a Git remote. The URLs are visible at the top of the project's home page. For an existing local repository, this command will create a remote named `gitlab` to the hosted location:

```
$ git remote add gitlab https://server/namespace/project.git
```

If you don't have a local copy of the repository, you can simply do this:

```
$ git clone https://server/namespace/project.git
```

The web UI provides access to several useful views of the repository itself. Each project's home page shows recent activity, and links along the top will lead you to views of the project's files and commit log.

Working Together

The simplest way of working together on a GitLab project is by giving another user direct push access to the Git repository. You can add a user to a project by going to the "Members" section of that project's settings, and associating the new user with an access level (the different access levels are discussed a bit in Groups). By giving a user an access level of "Developer" or above, that user can push commits and branches directly to the repository with impunity.

Another, more decoupled way of collaboration is by using merge requests. This feature enables any user that can see a project to contribute to it in a controlled way. Users with direct access can simply create a branch, push commits to it, and open a merge request from their branch back into `master` or any other branch. Users who don't have push permissions for a repository can "fork" it (create their own copy), push commits to *that* copy, and open a merge request from their fork back to the main project. This model allows the owner to be in full control of what goes into the repository and when, while allowing contributions from untrusted users.

Merge requests and issues are the main units of long-lived discussion in GitLab. Each merge request allows a line-by-line discussion of the proposed change (which supports a lightweight kind of code review), as well as a general overall discussion thread. Both can be assigned to users, or organized into milestones.

This section is focused mainly on the Git-related features of GitLab, but as a mature project, it provides many other features to help your team work together, such as project wikis and system maintenance tools. One benefit to GitLab is that, once the server is set up and running, you'll rarely need to tweak a configuration file or access the server via SSH; most administration and general usage can be accomplished through the in-browser interface.

Third Party Hosted Options

If you don’t want to go through all of the work involved in setting up your own Git server, you have several options for hosting your Git projects on an external dedicated hosting site. Doing so offers a number of advantages: a hosting site is generally quick to set up and easy to start projects on, and no server maintenance or monitoring is involved. Even if you set up and run your own server internally, you may still want to use a public hosting site for your open source code – it’s generally easier for the open source community to find and help you with.

These days, you have a huge number of hosting options to choose from, each with different advantages and disadvantages. To see an up-to-date list, check out the GitHosting page on the main Git wiki at https://git.wiki.kernel.org/index.php/GitHosting

We’ll cover using GitHub in detail in GitHub, as it is the largest Git host out there and you may need to interact with projects hosted on it in any case, but there are dozens more to choose from should you not want to set up your own Git server.

Summary

You have several options to get a remote Git repository up and running so that you can collaborate with others or share your work.

Running your own server gives you a lot of control and allows you to run the server within your own firewall, but such a server generally requires a fair amount of your time to set up and maintain. If you place your data on a hosted server, it’s easy to set up and maintain; however, you have to be able to keep your code on someone else’s servers, and some organizations don’t allow that.

It should be fairly straightforward to determine which solution or combination of solutions is appropriate for you and your organization.