
psiturk Documentation

Release 3.0.6

McDonnell, J.V., Martin, J.B., Markant, D.B., Coenen, A., Rich, A.S.

Mar 30, 2021

FIRST STEPS

1	Web based experiments	3
2	What is Mechanical Turk?	5
3	Where does psiturk come in?	7
4	Do I need to know how to code to use psiturk?	9
5	Do I need to setup and manage a webserver?	11
6	Do I need to setup and manage a database?	13
7	My IRB is concerned about privacy, will psiturk work for me?	15
8	Is psiturk only for use with Amazon Mechanical Turk?	17
9	Getting started developing	19
9.1	Step 1: Install psiturk	19
9.2	Step 2: Create a default project structure	20
9.3	Step 3: Launch psiturk in the new project directory	20
9.4	Step 4: Start the server	21
9.5	Step 5: Debug/view your experiment	21
9.6	Step 6: Check your data	21
10	Collecting data	23
10.1	Step 1: Enter credentials	23
10.2	Step 2: Create a sandboxed HIT/Ad	24
10.3	Step 3: Check your data	24
10.4	Step 4: Monitor progress	24
10.5	Step 5: Approve workers	25
10.6	Step 6: Switch to “live” mode	25
11	Dashboard Overview	27
11.1	Dynamic Filtered Table	27
11.2	Batch Actions	28
11.3	Campaigns	28
11.4	Tasks	28
11.5	Configuration	29
12	Installation guide	31
12.1	From Source	31
12.2	Running inside a Virtual Environment	31

12.3	System-specific notes	32
13	Recording Data	35
13.1	Types of Data	35
13.2	Saving the data	36
14	Retrieving Data	37
14.1	Methods	37
14.2	How the datastring is structured	38
15	Setting Up an Amazon Mechanical Turk Account	41
15.1	Accounts Creation and Linking	41
15.2	AWS Credentials	41
16	Server-side computations	43
16.1	Example: Automatically computing performance bonus	43
16.2	The basic logic of using <code>custom.py</code>	46
16.3	Tips about debugging your custom route	46
17	Alternative recruitment channels	47
18	Deploy to Heroku	49
19	Example project walkthrough	55
19.1	Background	55
19.2	Initialize the demo code	55
19.3	Set Your AWS Credentials	56
19.4	Configure the option for the demo experiment	56
19.5	Launch the psiTurk shell	56
19.6	Start/stop the experiment server	57
19.7	Debug/test the experiment locally	57
19.8	Experiment Structure	59
19.9	Launch in AMT sandbox	59
19.10	Accessing your data	61
19.11	Automatically computing a bonus	61
19.12	Approve/Reject Workers	61
19.13	Assigning bonuses	61
19.14	Launch “live” experiment	61
19.15	Conclusion	62
20	Using jsPsych+psiturk	63
21	Hosting your own HTTPS Experiment	65
21.1	Purchase a domain name from a domain name registrar	66
21.2	Create an account on Cloudflare if necessary	67
21.3	Add a new “site” to your Cloudflare account with the name of your purchased domain name	67
21.4	Set Cloudflare to be your DNS provider	68
21.5	Create a DNS A record	68
21.6	Enable Flexible SSL	70
21.7	Start your psiTurk server	71
22	Using commercial survey tools	73
22.1	An example with Qualtrics	73
22.2	What about not-Qualtrics?	74
23	Automatically compute bonuses	75

24	psiturk.js API	77
24.1	Creating the psiTurk object	77
24.2	psiturk.taskdata	77
24.3	psiturk.preloadPages (pagelist)	78
24.4	psiturk.getPage (pagename)	78
24.5	psiturk.showPage (pagename)	78
24.6	psiturk.preloadImages (imagelist)	78
24.7	psiturk.recordTrialData (datalist)	79
24.8	psiturk.recordUnstructuredData (field, value)	79
24.9	psiturk.saveData ([callbacks])	79
24.10	psiturk.completeHIT ()	79
24.11	psiturk.doInstructions (pages, callback)	80
24.12	psiturk.finishInstructions ()	81
25	psiTurk commands	83
25.1	amt_balance	83
25.2	config	84
25.3	debug	85
25.4	download_datafiles	85
25.5	help	86
25.6	hit	87
25.7	worker	89
25.8	quit	93
25.9	server	93
25.10	status	94
25.11	mode	94
26	Migrating from psiTurk 2 to 3	97
26.1	Announcing psiTurk 3.0	97
26.2	Migration technical considerations	98
27	Disclaimer	101
28	Getting help	103
29	Code of Conduct	105
29.1	Our Pledge	105
29.2	Our Standards	105
29.3	Enforcement Responsibilities	106
29.4	Scope	106
29.5	Enforcement	106
29.6	Enforcement Guidelines	106
29.7	Attribution	107
30	Project Roadmap	109
30.1	General priorities	109
30.2	Version 3.0	110
31	Contributing to psiTurk	111
31.1	Contribution guidelines	111
31.2	Decision process	113
32	Welcome to psiturk	115
32.1	Video introduction	115
32.2	How to use our docs	115

32.3	Open source, community-built	116
32.4	Join the community	116

To begin with it can be helpful to lay out the problems `psiturk` does and does not solve to help you gauge whether it will be useful to you.

However, before getting started let's make sure we have some shared terminology.

WEB BASED EXPERIMENTS

Web based experiments present stimuli, videos, questionnaires, and animations to people over the Internet using standard web browser technologies. Increasingly researchers use crowdwork websites to facilitate the recruitment to these tasks. However, people use web based experiments for many types of designs even for in-person experiments, iPad/tablet based experiments in the field, fMRI designs, etc... The critical issue is if you want to use the browser as your “task interface.” `psiturk` can be used to help anytime you want to perform an experiment in a browser.

WHAT IS MECHANICAL TURK?

Amazon Mechanical Turk (AMT) is an online platform that lets you post a wide variety of tasks to a large pool of participants. Instead of spending weeks to run experiments in the lab, it lets you collect data of a large number of people within a couple of hours.

Some key terminology for understanding the AMT model:

- **HIT (Human Intelligence Task)** - A unit of work (e.g. a psychology experiment)
- **Requester** - The person or entity that posts HITs (e.g. a researcher or lab)
- **Worker** - The person that completes HITs (i.e. a participant in your study)

Workers get paid a fixed amount for each HIT which is determined by the requester. Requesters can also make bonus payments to specific workers. Amazon collects a 10% fee for each payment (which goes even higher when you post hits with many assignments).

WHERE DOES PSITURK COME IN?

The `psiturk` toolbox is designed to help you run fully-customized and dynamic web-experiments over the Internet. Specifically, it allows you to:

1. Run a web server for your experiment
2. Test and debug your experiment
3. Deploy your experiment to a secure, high availability cloud server
4. Interact with sites like Amazon Mechanical Turk to recruit, post HITs, filter, and pay participants (AMT workers)
5. Manage databases and export data

`psiturk` also includes a powerful scripting interface for automating the types of tasks you typically perform when running a web-based experiment. These scripts can also be used to avoid around the extra fee Amazon charges for certain “large” HITs.

DO I NEED TO HOW TO CODE TO USE PSITURK?

Well, yes. `psiturk` experiments are run in web browsers. To develop a web browser experiment, you need to have basic web programming skills with HTML, CSS, and JavaScript. You sometimes might also use Python to customize your experiment.

To get you started, `psiturk` provides a fully functioning example experiment in the [Example project walkthrough](#) section that you can use as a template for your own study. `psiturk` also includes a library of basic Javascript functions (see [psiturk.js API](#)) that you can insert into your code to handle page transitions, load images, and record data.

However, for actually programming the interface of your task people often use other javascript tools like the outstanding [jsPsych](#).

The good news is that many people have developed experiments using `psiturk` and you can use these examples to bootstrap your own efforts.

If you are looking for ways to run experiments online without coding consider [Pavlovio](#).

DO I NEED TO SETUP AND MANAGE A WEBSERVER?

No, because `psiturk` does this for you.

If you run an experiment on the web you need a way to send the content to the user. This usually requires some type of http/https web server. While you can do that from your personal computer (with some setup/configuration), currently the easiest way to do this is to deploy (i.e., run on a remote cloud server) your completed experiment to the web via the free-tier on Heroku. When you are developing locally/testing your code `psiturk` provides you a small, simplified web server running on your computer that mimics the environment that is run when your experiment is officially “deployed.”

TL;DR: `psiturk` provides a lightweight mini-webserver that you can use to develop your task locally on your desktop. When you are ready to begin data collection you deploy to the cloud. We have a smooth workflow for doing this for free on Heroku, although other options are possible.

DO I NEED TO SETUP AND MANAGE A DATABASE?

No, because `psiturk` does this for you (sound familiar?).

Browsers do not allow you to write directly to the user's file system. As a result data has to be stored someplace. This often requires some type of **database**.

There are [non-database solutions for some systems](#) (e.g., Mechanical Turk) that store the data in a special field on the crowdworker site. However, if you then want to use your experiment in the lab or with a different crowdworking platform you have to figure something new out. Your IRB might have issues with saving subject data on Amazon's servers linked to the subject's Mechanical Turk account identity.

`psiturk` helps you interface your experiment with a robust and secure database. `psiturk` automatically handles anonymization of your subject's identity. When you deploy a `psiturk` experiment it can create a free, robust Postgres database on Heroku that will be entirely deleted when you complete your experiment (after you download your anonymized files of course!).

If you do have your own database though `psiturk` is happy to use it.

TL;DR: Yes you need a database, but there is no database software required for you to personally install or maintain. `psiturk` will create free, managed databases for you if you don't have one or bring you own!

MY IRB IS CONCERNED ABOUT PRIVACY, WILL PSITURK WORK FOR ME?

When you use `psiturk` communication is directly between you and your participant. There is no centralized data collection or monitoring. `psiturk` data saving routines anonymize your databases so that no potentially identifiable information is associated with a datafile except with a special secret key that only you (as the `psiturk` developer) know. If you set up your own database it is up to you to secure it, but if you use the Heroku version it is password protected and even can be deleted after data collection (and your data is downloaded to a file for safe keeping and analysis of course!).

IS PSITURK ONLY FOR USE WITH AMAZON MECHANICAL TURK?

No! You can use `psiturk` anytime you need to deliver an experiment over the Internet. Some researchers for instance have used `psiturk` to design iPad experiments that are run in person at schools or museums. In these cases none of the Mechanical Turk specific features are needed. `psiturk` original was designed (and named) based on its connection to Mechanical Turk but as time goes on it has become more general. We still like the name and logo though.

That said, if you *do* recruit people from a crowdworking site it can be a hassle to figure out what their bonus payment was and to remember to properly compensate people. `psiturk` provides a simple, scriptable API to interfacing with common payment tasks, especially with Amazon Mechanical Turk. Non-`psiturk` solutions involve using the clumsy tools and web interfaces provided by the crowd labor sites. Ick!

GETTING STARTED DEVELOPING

Generally there is a distinction between using *psiturk* locally to help you **develop** your experiment code versus **deploying** it in a way that enables data collection. This guide shows how easy it is to get up and running the developing your experiment code using *psiturk*.

Generally one develops an experiment entirely on a local personal computer. *psiturk* creates a local copy of the web environment that will later run on the cloud when deployed.

For this reason, this section of the guide steps you through the process of installing *psiturk* on your personal machine and stepping through common debugging steps.

Overview

- *Step 1: Install psiturk*
- *Step 2: Create a default project structure*
- *Step 3: Launch psiturk in the new project directory*
- *Step 4: Start the server*
- *Step 5: Debug/view your experiment*
- *Step 6: Check your data*

9.1 Step 1: Install psiturk

psiturk can be installed easily on any system that has a Python (≥ 3.6) installation and has the python package manager *pip*. At the terminal ():

```
pip install psiturk
```

See also:

Installation guide

After you install it can be helpful to double check that you have the correct version installed:

```
psiturk version  
psiTurk version 3.0.6
```

If you are following this guide you want this to be version 3.0.6 or higher.

9.2 Step 2: Create a default project structure

psiturk includes a simple example project which you can use to get started.

```
psiturk-setup-example

Creating new folder `psiturk-example` in the current working directory
...
Creating default configuration file (config.txt)
```

psiturk should be locally run in the top level folder of your project. Enter this shell command to enter the newly created folder:

```
cd psiturk-example
```

The default project include several default files you will later modify. To list them:

```
ls -la
ls -la
total 40
ls -la
total 47
drwxr-xr-x   7 user  staff   224 Mar 29 21:40 .
drwx-----@ 147 user  staff  4704 Mar 29 12:53 ..
-rw-r--r--   1 user  staff   8649 Mar 29 21:40 config.txt
-rw-r--r--   1 user  staff   3461 Mar 29 21:40 custom.py
drwxr-xr-x   8 user  staff   256 Mar 29 21:40 static
drwxr-xr-x  18 user  staff   576 Mar 29 21:40 templates
```

See also:

[anatomy-of-project](#)

9.3 Step 3: Launch psiturk in the new project directory

psiturk should be run in the top level folder of your project. You should be greeted with a welcome screen and command prompt.

There is also an extensive help system in the command line. Type `help` to see a list of available commands. Type `help <cmd>` to get more information about a particular command (e.g., `help server`).

```
cd psiturk-example
psiturk

welcome...
psiTurk version 2.1.1
Type "help" for more information.
[psiTurk server:off mode:sdbx #HITS:0]$
```

See also:

[command-line-overview](#)

9.4 Step 4: Start the server

The psiturk server is the web server which responds to external requests. To start or stop the server type `server on`, `server off`, or `server restart`.

```
[psiTurk server:off mode:sdbx #HITS:0]$ server on

Experiment server launching...
Now serving on http://localhost:22362
[psiTurk server:on mode:sdbx #HITS:0]$
```

9.5 Step 5: Debug/view your experiment

To debug or test the experiment, simply type `debug`. This will launch the default web browser on your system and point it at your experiment in a method which is helpful for testing.

Hint: If you are running on a remote server and want to disable launching the browser type `debug -p` (print only) which will print the debugging URL but not launch the browser.

Altering the experiment code is beyond the scope of the quick start, but see [this guide](#) for details on how to modify and extend the stroop example.

A short summary is that you make changes to the files in the *static/* and *templates/* folder to reflect your experiment design. You do not need to restart the server as you change these files locally. The changes will be reflected the next time you load the experiment url.

```
[psiTurk server:on mode:sdbx #HITS:0]$ debug

Launching browser pointed at your randomized debug link, feel free to request another.
  http://localhost:22362/ad?assignmentId=debugX12JJ8&hitId=debugA7NP2T&
↪workerId=debugS9K039
[psiTurk server:on mode:sdbx #HITS:0]$
```

Notice that the debug link assigns random values to the *assignmentId*, *hitId*, and *workerId*. These are values typically provided by the Mechanical Turk system but which are set randomly during debugging so you can isolate this data in your analysis later.

9.6 Step 6: Check your data

By default psiTurk saves your data to a SQLite database `participants.db` in your base project folder. You can check that everything is being recorded properly by opening that file in a SQLite tool like Base.

See also:

[databases-overview](#)

At this point you develop your experiment until you are confident it is ready to run. Then you move on to our guide to [Collecting data](#).

COLLECTING DATA

Once you have your task completely developed (see *Getting started developing*) then you may be ready to collect data. There are several ways you might recruit subjects including studies conducted in the lab but in this section we focus on the common use case of recruiting on Amazon Mechanical Turk.

Overview

- *Step 1: Enter credentials*
- *Step 2: Create a sandboxed HIT/Ad*
- *Step 3: Check your data*
- *Step 4: Monitor progress*
- *Step 5: Approve workers*
- *Step 6: Switch to “live” mode*

10.1 Step 1: Enter credentials

In order to get access to the Amazon Mechanical Turk features of `psiturk`, you need obtain and enter credentials for accessing Amazon Web Services. These can be added to `~/.psiturkconfig`. You can leave the `aws_region` at the default value.

```
cat ~/.psiturkconfig

[AWS Access]
AWS_ACCESS_KEY_ID = YourAccessKeyId
AWS_SECRET_ACCESS_KEY = YourSecretAccessKey
aws_region = us-east-1
```

See also:

Setting Up an Amazon Mechanical Turk Account

10.2 Step 2: Create a sandboxed HIT/Ad

In order to make the experiment available to workers on Amazon Mechanical Turk you need to:

1. Run your psiturk server on a machine that is publicly-accessible.
2. Post a HIT on AMT, which will point MTurkers to your psiturk server address.

Use the `ad_url` settings to point to the location of your publicly-accessible experiment.

See the [Deploy to Heroku](#) guide for an example of running your experiment on the webserver of a platform-as-a-service cloud provider.

The example below uses the Amazon Mechanical Turk “sandbox,” which is a place for testing your task without actually offering it live to real paid workers.

Run the following to post a HIT, and answer all prompts:

```
::
```

```
[psiTurk server:on mode:sdbx #HITs:0]$ hit create
```

Your HIT should now be visible on <http://workersandbox.mturk.com> if you search for your requester account name or the HIT title word “Stroop” (set in `config.txt`).

Warning: Important! Test to make sure that your Ad URL can be accessed from a place external to the network from which you created the HIT. If it cannot be accessed, then MTurkers won’t be able to access your HIT!

10.3 Step 3: Check your data

By default psiTurk saves your data to a SQLite database `participants.db` in your base project folder. You can check that everything is being recorded properly by opening that file in a SQLite tool like Base.

See also:

[databases-overview](#)

10.4 Step 4: Monitor progress

One simple way to monitor how many people have actually accepted your HIT is with the `hit list --active` or `hit list --reviewable` commands.

This shows the HITid for each task, how many have completed, and how many are pending.

See also:

See these FAQs:

- [interpret-hit-status](#)
- [why-no-hits-available](#)

10.5 Step 5: Approve workers

psiTurk provides many tools for approving workers, assigning bonuses, etc. Try `help hit` and `help worker`.

One simple approach is to approve all the workers associated with a particular HIT (once all the assignments are complete). To do this, use the `worker approve --hit <HITID>` command.

```
[psiTurk server:on mode:sdbx #HITs:0]$ worker approve --hit ↵  
↪28K4SME3ZZ2MZI004SETTTXTTAG44LT  
Approving....
```

10.6 Step 6: Switch to “live” mode

In order to create public hits on the “live” AMT site, you need to switch modes in the command shell using the `mode` command. To get back to the sandbox, just type `mode` again.

To avoid mistakes, psiTurk defaults to sandbox mode (this behavior can be changed in `config.txt`)

From here, everything is exactly the same as described for sandbox hits above.

```
[psiTurk server:on mode:sdbx #HITs:1]$ mode  
Entered live mode  
[psiTurk server:on mode:live #HITs:0]$
```


DASHBOARD OVERVIEW

A dashboard is available at route `/dashboard`. The dashboard can be enabled by setting `settings-enable-dashboard` to `True`, and by setting a `settings-login-username` and `settings-login-pw`.

The dashboard has many features, including a dynamic filtered table, batch actions, managing campaigns.

Contents

- *Dynamic Filtered Table*
- *Batch Actions*
- *Campaigns*
- *Tasks*
- *Configuration*

11.1 Dynamic Filtered Table

View current status of participants, queried from the psiTurk database.

- Filter by:
 - mode
 - experimental condition
 - current code version
 - whether the participant has a status of ‘complete’
- Group by:
 - condition

11.2 Batch Actions

Functionality to manually trigger the following actions:

- Workers:
 - Approve all HITs
 - Bonus all submissions via the “auto” method (based on the value set in the “bonus” column in the database).
- HITS:
 - Expire all
 - Approve all workers for all hits
 - Delete all

11.3 Campaigns

Campaigns are scheduled as *tasks*. Campaigns have the following features:

- Set a target “goal” for number of workers for a given task code version.
- Stagger the posting of HITs by a specified interval.
- Post HITs in batches of 9 assignments. This keeps the MTurk commission at 20%, instead of 40%.
- Monitors the number of available HITs, and continues posting rounds of HITs until the campaign goal has been met.
- Manually cancel a campaign.

The “Campaigns” tab also displays past campaigns.

11.4 Tasks

psiTurk “tasks” are stored in their own table in the database specified by the settings-database-url. To enable a specific psiturk server to run tasks, set settings-do-scheduler to `true`.

The “tasks” tab allows for scheduling an “Approve All workers” task which will be run at a set interval. Will “approve” all submissions currently marked as “Submitted” in the psiTurk database.

The tab will also display any currently running campaigns. To edit a campaign, visit the “Campaigns” tab.

Warning: The *managing* of tasks and the *running* of tasks is handled separately!

This is because psiTurk uses `APScheduler` for tasks, which does not currently handle interprocess synchronization (see [this APScheduler FAQ](#)).

This means that any dashboard can view, create, delete, and update tasks, while a single separate psiTurk server instance can be set up with only one thread for task-running.

Note!: if settings-do-scheduler is set to `True`, and settings-threads is greater than 1, psiTurk will refuse to start! This is a safeguard, because, again, `APScheduler` cannot handle interprocess task-running synchronization.

11.5 Configuration

- Set dashboard mode.
- View AMT balance.

INSTALLATION GUIDE

psiTurk is supported for python \geq v3.6 on any [Unix-Like](#) operating system (i.e., *not Windows*).

When psiTurk is successfully installed, you will have a new command line tool available called `psiturk`. The `psiturk` command provides a number of functions to you including launching the server and interacting with the Mechanical Turk and Amazon Web Services (AWS) systems.

Requirements:

- python (\geq v3.6)
- pip (to install, see [here](#).)

To install the latest released version of psiTurk:

```
pip install psiturk
```

To upgrade psiturk:

```
pip install -U psiturk
```

12.1 From Source

You can install the bleeding edge version of psiTurk from source just as you would install any other Python package:

```
pip install git+https://github.com/NYUCCL/psiturk.git
```

To update from source:

```
pip install -U git+https://github.com/NYUCCL/psiturk.git
```

12.2 Running inside a Virtual Environment

It can be desirable to keep each of your experiments' dependencies (python and python package versions) isolated from each other. For example, if you want to install the development version of psiTurk (as described *above*) in one experiment, but not all the others installed on your system, [Virtual Environments](#) provide a solution.

You can install via pip:

```
sudo pip install virtualenv virtualenvwrapper
```

This will install the virtualenv tool as well as the supplementary virtualenvwrapper tools that make working with virtualenvs easier. You create a virtual environment as follows:

```
$ mkvirtualenv my-experiment

Running virtualenv with interpreter /usr/bin/python2
New python executable in my-experiment/bin/python2
Also creating executable in my-experiment/bin/python
Installing setuptools, pip...done.
```

(if mkvirtualenv is not recognized, follow the instructions [here](#))

Then, at any point in the future, to activate the virtual environment, use the workon command:

```
$ workon my-experiment
(my-experiment) $ which python python pip easy_install

~/virtualenvs/my-experiment/bin/python
~/virtualenvs/my-experiment/bin/pip
~/virtualenvs/my-experiment/bin/easy_install
```

As you can see, when the environment is active, running python or pip will run copies specific to your project. Any packages installed with pip or easy_install will be installed inside your my-experiment virtualenv rather than system-wide.

Install psiTurk as above into the `_virtual_` environment—i.e., with the virtual environment activated.:

```
(my-experiment) $ pip install psiturk
```

You can use the `deactivate` command to leave the virtualenv.

12.3 System-specific notes

12.3.1 Mac OS X

Apple users will need to install a C compiler via XCode; to do so, install XCode from the App store. Once you have downloaded it, install the command line tools from the preferences menu as instructed [here](#). For earlier versions of Mac OS X (e.g., Snow Leopard) you may need to install XCode using the installation disc that came with your computer. The command line tools are an option during the installation process for these systems.

12.3.2 Linux

psiTurk is relatively painless to install on most Linux systems since the installation requirements come installed by default in most distributions.

If you encounter install problems when installing using pip as above, a likely cause is that you are missing the package from your distribution that contains a needed header file. In this case, one way to troubleshoot the problem is to do a web search for the name of your distribution and the name of the missing header file (which often appears in the error text produced by a failed pip install). That search will likely turn up the name of the package for your distribution that supplies the needed header file.

As an example, before installing psiTurk on a minimal Debian server (such as the one provided by many server hosting companies) you will need to install some additional packages, as illustrated by the following example command:

```
apt install python-pip python-dev libncurses-dev
```

If you would like to use mysql as your backend database (which is optional, and can be done at any time), further packages are needed. On a Debian system, they are:

```
apt install python-pymysql python-sqlalchemy libmysqlclient-dev
```

If you have additional specific issues, or if you can report the steps needed to install psiTurk on a particular Linux distribution, please help us update the documentation!

12.3.3 Windows

psiTurk is currently not supported on Windows. This is due to a technical limitation in the ability to run server processes on Windows. However, there are a number of options to get around this (see below for details on each option):

- [Windows Subsystem for Linux \(WSL\)](#) on Windows 10. **Recommended.**
- Virtualization through [VirtualBox](#) or similar software.

Windows Subsystem for Linux (WSL)

Windows now has the option to run a Linux translation layer inside Windows (WSL 1) or even a full Linux kernel (WSL 2). Either will allow you to run psiturk within the Linux subsystem. See <https://docs.microsoft.com/en-us/windows/wsl/install-win10> for instructions on how to activate WSL on your system.

After you activate WSL and install a Linux distribution of choice, install psiturk within a WSL-connected command prompt as above for [Linux](#).

Virtualization

Warning: WSL may not be compatible with concurrent usage of other hypervisors.

You can install a program like [VirtualBox](#) on your pc. Programs like these are called hypervisors and emulate a computer within your computer. Your physical machine is called a host and the virtual machine is called a guest. This technique allows you to install a Linux guest regardless of what OS the host is running.

Virtualization requires some computing power from the host so this option is not recommended if your psiturk experiment requires a lot of computing power as well or if it's expected to have a lot of participants active at once. However, it is a good option to develop and test your psiturk experiments on Windows systems prior to Windows 10. If you are running Windows 10 or higher see below for the WSL option, which is much easier on your system than virtualization.

After you install the virtual machine software you need an installation image for a Linux based OS. You can choose any Linux distribution you like but [Ubuntu](#) is a good choice if you don't know which one to pick. You can usually obtain an *.iso file for the Linux distribution you like. These are virtual cd-roms. You can load them into your virtual machine and begin installing the guest OS. Once that is complete you boot your virtual machine into Linux and follow the installation steps for Linux.

RECORDING DATA

13.1 Types of Data

To record data in your task, you make calls to the [psiturk.js Javascript API](#). There are three kinds of data that psiTurk will help you produce:

1. *Trial-by-trial log file*
2. *Unstructured (field, value) pairs*
3. *Browser events*

13.1.1 Recording trial data

The first dataset that will be produced by your experiment will be a simple log file, which you add to a single line at a time. In order to add a line of data to the log, use `psiturk.recordTrialData`:

```
psiturk.recordTrialData(['this', 'is', 1, 'line'])
```

The list of values that you supply to `recordTrialData` will then be appended to the log. It is up to you how to structure those lists; you will have to parse them as part of your analysis. Each time you call `psiturk.recordTrialData`, it will also record the time it was called (with a UTC timezone).

13.1.2 Recording unstructured data

In addition to trial by trial data, there is often a need to record information about a participant in the form of (field, value) pairs, for which you can use `psiturk.recordUnstructuredData`:

```
psiturk.recordUnstructuredData('age', 24)  
psiturk.recordUnstructuredData('response', 'yes')
```

Like the trial-by-trial data, it is up to you to decide whether or not to use this function. For some kinds of experiments (like simple surveys), this might be the only function you need.

13.1.3 Browser event data

The third dataset is generated automatically without any input from the experiment, and is used to track special kinds of events that occur as a worker is interacting with the page. Currently, this includes:

1. “resize” events: when the worker changes the size of their browser window (the first value recorded is the initial size of the window)
2. “focus” events: when the worker switches to and from a different browser window or application. If the worker leaves the experiment window, a “focus off” event is recorded; when they return a “focus on” event is recorded.

Note: Information about how to retrieve recorded data sets can be found [here](#).

13.2 Saving the data

It’s important to remember that `psiturk.recordTrialData` and `psiturk.recordUnstructuredData` only modify the `psiturk` object on the client side. If you want to save the data that has been accumulated to the server, you must call `psiturk.saveData()`.

It’s up to you how often `psiturk.saveData()` syncs the task data to the server (e.g., after every block, or once at the end of the experiment). Using `saveData` frequently will limit the loss of data if the participant runs into an error, but keep in mind that it involves a new request to the server each time it is called.

RETRIEVING DATA

This section covers methods for retrieving datasets, as well as the structure of saved data.

14.1 Methods

There are several ways to retrieve experiment data from the database.

14.1.1 Retrieving using `download_datafiles`

The simplest way to retrieve data is using the `download_datafiles` command. This creates three csv files containing the three kinds of data: `trial data`, `question data`, and `event data`.

14.1.2 Retrieving programmatically

While the `download_datafiles` shell command is the simplest way to retrieve experiment data, a more powerful and flexible solution is to retrieve the data programmatically. Many languages offer libraries for interfacing with mysql and sqlite databases - below is an example using python and the sqlalchemy package to retrieve data from a mysql database. We add `+pymysql` to the `db_url` to let sqlalchemy make use of pymysql package. (You can leave the `database_url` in `config.txt` as `mysql://` though – psiturk adds `+pymysql` internally). By including code such as this at the beginning of your analysis script, you can be sure the the data you’re analyzing is always complete and up-to-date.

```
from sqlalchemy import create_engine, MetaData, Table
import json
import pandas as pd

db_url = "mysql+pymysql://username:password@host.org/database_name"
table_name = 'my_experiment_table'
data_column_name = 'datastring'
# boilerplate sqlalchemy setup
engine = create_engine(db_url)
metadata = MetaData()
metadata.bind = engine
table = Table(table_name, metadata, autoload=True)
# make a query and loop through
s = table.select()
rows = s.execute()

data = []
#status codes of subjects who completed experiment
statuses = [3,4,5,7]
```

(continues on next page)

(continued from previous page)

```

# if you have workers you wish to exclude, add them here
exclude = []
for row in rows:
    # only use subjects who completed experiment and aren't excluded
    if row['status'] in statuses and row['uniqueid'] not in exclude:
        data.append(row[data_column_name])

# Now we have all participant datastrings in a list.
# Let's make it a bit easier to work with:

# parse each participant's datastring as json object
# and take the 'data' sub-object
data = [json.loads(part)['data'] for part in data]

# insert uniqueid field into trialdata in case it wasn't added
# in experiment:
for part in data:
    for record in part:
        record['trialdata']['uniqueid'] = record['uniqueid']

# flatten nested list so we just have a list of the trialdata recorded
# each time psiturk.recordTrialData(trialdata) was called.
data = [record['trialdata'] for part in data for record in part]

# Put all subjects' trial data into a dataframe object from the
# 'pandas' python library: one option among many for analysis
data_frame = pd.DataFrame(data)

```

14.2 How the datastring is structured

The main data from an experiment participant is held in a string of text in the *datastring* field of the data table. Understanding how this string is structured is important to be able to parse the string into a useful format for your analyses.

The *datastring* is structured as a *json object*. In the description that follows, sub-objects are indicated by names wrapped in angle brackets (<>).

14.2.1 Top Level

The top level of the datastring contains summary information about the worker, as well as the datastring sub-objects:

```

{"condition": condition,
 "counterbalance": counterbalance,
 "assignmentId": assignmentId,
 "workerId": workerId,
 "hitId": hitId,
 "currenttrial": trial_number_when_data_was_saved,
 "useragent": useragent,
 "data": <data>,
 "questiondata": <questiondata>,
 "eventdata": <eventdata>,
 "mode": <mode>}

```

14.2.2 data

The data sub-object contains a list of the data recorded each time `psiturk.recordTrialData()` is called in the experiment:

```
[
  {
    "uniqueid": uniqueid,
    "current_trial": current_trial_based_on_num_of_calls_to_recordTrialData,
    "dateTime": current_time_in_system_time,
    "trialdata": <datalist>
  }
  //,
  // ...
]
```

Here, `<datalist>` is whatever is passed to `psiturk.recordTrialData()` in the experiment. This could be in any format, such as a string or list, but we recommend saving data in a json format so that all data is stored in a clear, easy-to-parse “field-value” format. `<dateTime>` is recorded in UTC time.

14.2.3 questiondata

The questiondata sub-object contains all items recorded using `psiturk.recordUnstructuredData()`.

```
{
  "field1": value1,
  "field2": value2,
  ...
}
```

14.2.4 eventdata

The eventdata sub-object contains a list of events (such as window resizing) that occurred during the experiments:

```
[{"eventtype": eventtype,
  "value": value,
  "timestamp": current_time_in_system_time,
  "interval": interval},
  ...
]
```


SETTING UP AN AMAZON MECHANICAL TURK ACCOUNT

psiTurk can interface with Amazon Mechanical Turk (although it doesn't have to!). To do so, you need to create an account on Amazon's website in order to use it. There are a number of steps involved here which have to do with signing up with Amazon and creating several accounts. Luckily they are a one-time process for a given AWS account.

15.1 Accounts Creation and Linking

Carefully follow [AWS's guide](#) for setting up the necessary accounts for using Amazon Mechanical Turk. Before doing so, note the following:

- **Step 5** discusses setting up the Developer Sandbox. Carefully follow all steps in this section, including the steps in the note for linking your aws account *specifically to the sandbox*.
- **Step 6** in the guide is "Set up an AWS SDK". You may skip this step – psiTurk uses the [Python/Boto](#) (Boto3) SDK under the hood.
- **Step 7** in the guide suggests the option of enabling AWS Billing for your account. However, at least one psiTurk user has reported difficulties doing so, needing to contact AWS customer support before being able to post hits.

15.2 AWS Credentials

psiTurk uses the [Python/Boto](#) (Boto3) SDK to communicate with the AWS API. In order to do so, boto must have access to the user's AWS credentials, generated in section [Setting Up an Amazon Mechanical Turk Account](#).

There are two approaches for setting the keys: (1) in a file called `.psiturkconfig` located in the user's home directory, and (2) in any of the ways that Boto expects.

15.2.1 .psiturkconfig approach

If set here, the keys should be lowercased, and under an 'AWS Access' section key, as follows:

```
[AWS Access]
aws_access_key_id = foo
aws_secret_access_key = bar
```

15.2.2 Boto approach

If AWS credentials are not found via the *.psiturkconfig* approach, then Boto will search for them via its typical methods. That is, psiTurk users can store AWS credentials in *any way that Boto expects*. Specifically, the credentials variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESSSS_KEY` can be set via one of the following methods, listed in order of Boto-preference:

1. Environment variables (can optionally be set in `.env`)
2. Shared credential file (`~/.aws/credentials`)
3. AWS config file (`~/.aws/config`)
4. Boto2 config file (`/etc/boto.cfg` and `~/.boto`)

Note: psiTurk sets the `AWS_DEFAULT_REGION` to ‘us-east-1’, and this cannot be overridden.

For example, if a user’s `AWS_ACCESS_KEY_ID` were ‘foo’, their `AWS_SECRET_ACCESS_KEY` were ‘bar’, they might set the following in their `~/.aws/credentials` file:

```
AWS_ACCESS_KEY_ID=foo
AWS_SECRET_ACCESSSS_KEY=bar
```

Note that Boto3 respects certain environment variables that alter which files are searched for credentials and configuration settings. See [here](#) for more information. So I still don’t get it, do I program my experiment in psiturk or what?

Well, the short answer is no. You write the **interface** parts of your experiment using HTML/CSS/Javascript typically (although Flash or Unity are in theory possible as well).

That said, in some cases it can be easier to design the logic, flow, and randomization of your experiment in Python and then just use Javascript as a light weight “dumb” display interface. For instance, perhaps your experiment is based on complex video game written in [Open AI Gym](<https://gym.openai.com>). To allow humans to interface with this you don’t want to have to re-write the entire game logic in Javascript. *psiturk* makes this easy by allowing you to pass data back and forth between a python sever and a “dumb” javascript client running in the browser. In this case all the brains would be running inside *psiturk* and then the Javascript just makes the pictures and buttons in the browser appear (sometimes known as a Model-View-Controller development model).

SERVER-SIDE COMPUTATIONS

Sometimes you might like to add additional urls or “routes” to your project. For instance you could make a password protected dashboard to visualize your data as it comes in, add additional functionality to your psiturk experiment, or add more complex server-side computations (e.g., fitting a computational model to the subject in real time and using that to adapt the stimuli people view).

This can be achieved by using [Flask Blueprints](#). psiTurk will look for a file called `custom.py` in the project directory, and import any blueprint from that module named `custom_code`. See below for examples.

16.1 Example: Automatically computing performance bonus

It is hard to use the main task to directly modify the database. However, you may use `custom.py` file with a function called `compute_bonus` to put the correct amount of bonus in the database. You could do this in Javascript perhaps but the problem is that participants can modify the javascript in their browser and increase their bonus. Instead it is better if bonuses are computed on the server side. The `custom.py` script may look like the following:

```
from flask import Blueprint, request, render_template, jsonify, abort, current_app
# dealing with error
from psiturk.experiment_errors import ExperimentError

# Database setup
from psiturk.db import db_session, init_db
from psiturk.models import Participant
# dealing with json like reading from database
from json import dumps, loads

# explore the Blueprint
custom_code = Blueprint('custom_code', __name__, template_folder='templates', static_
↳ folder='static')

#-----
# example computing bonus
#-----

@custom_code.route('/compute_bonus', methods=['GET'])
def compute_bonus():
    # check that user provided the correct keys
    # errors will not be that gracefull here if being
    # accessed by the Javascript client
    if not 'uniqueId' in request.args:
        # i don't like returning HTML to JSON requests... maybe should change this
        raise ExperimentError('improper_inputs')
```

(continues on next page)

(continued from previous page)

```

uniqueId = request.args['uniqueId']

try:
    # lookup user in database
    user = Participant.query.\
        filter(Participant.uniqueid == uniqueId).\
        one()
    user_data = loads(user.datastring) # load datastring from JSON
    bonus = 0

    for record in user_data['data']: # for line in data file
        trial = record['trialdata']
        if trial['phase'] == 'TEST':
            if trial['hit'] == True:
                bonus += 0.02
    user.bonus = bonus
    db_session.add(user)
    db_session.commit()
    resp = {"bonusComputed": "success"}
    return jsonify(**resp)
except:
    abort(404) # again, bad to display HTML, but...

```

Accordingly, in the main task file `task.js`, you would call this function with the `computeBonus` function. Add a piece of the code at the end of your experiment:

```

psiTurk.computeBonus("compute_bonus", function () {
    psiTurk.completeHIT(); // when finished saving compute bonus, the quit
});

```

Now let's walk through some key points of this process.

```

from flask import Blueprint, request, render_template, jsonify, abort, current_app

```

The key player in customizing is the `flask` package. It helps you run a webserver (HTTP server).

```

custom_code = Blueprint('custom_code', __name__, template_folder='templates', static_
↪folder='static')

```

Here we create a Blueprint object. Blueprint is an organizing tool. Here what's important for us is to specify the location template folder and static folder which may be used, for example, when you wanna display a HTML file.

```

@custom_code.route('/compute_bonus', methods=['GET'])

```

The first argument in `route` is the URL that when is called will run the function right below it. For example, if you are running your task locally on port 5000, then type in `http://localhost:5000/compute_bonus`, which will call the function `compute_bonus` defined right below. The `methods` argument is defining the information flow communicating with this function – it will “get” information from outside.

BTW, in case you are wondering, the `@` in front of this line is called “decorator”. It uses the current line (in our case, the `route` function) to “decorate” the function right below it. A helpful tutorial that further explains this concept is [here](#).

```

def compute_bonus():
    if not 'uniqueId' in request.args:
        # i don't like returning HTML to JSON requests... maybe should change this

```

(continues on next page)

(continued from previous page)

```
raise ExperimentError('improper_inputs')
uniqueId = request.args['uniqueId']
```

Here we use request to receive the information sent from javascript. In our case it's taken care by the computeBonus function. Looking into computeBonus to see where that “uniqueID” comes from:

```
self.computeBonus = function(url, callback) {
$.ajax(url, {
    type: "GET",
    data: {uniqueId: self.taskdata.id},
    success: callback
  });
};
```

As mentioned before, the url is the route name; the data is a dictionary with one key named “uniqueID”, which is being looked for in the python compute_bonus function.

Now let's coming back to the compute_bonus function:

```
try:
    # lookup user in database
    user = Participant.query.\
        filter(Participant.uniqueid == uniqueId).\
        one()
    user_data = loads(user.datastring) # load datastring from JSON
```

Now the database kicks in. We've created a user object which we will be able to read all data about this user that has been saved in the database, as well as write something.

```
bonus = 0
for record in user_data['data']: # for line in data file
    trial = record['trialdata']
    if trial['phase'] == 'TEST':
        if trial['hit'] == True:
            bonus += 0.02
```

Now we calculate bonus by checking how many trials are correct.

```
user.bonus = bonus
db_session.add(user)
db_session.commit()
```

We assign value for the “bonus” column of this user and commit to the database. This will enable psiturk to give bonus.

```
resp = {"bonusComputed": "success"}
return jsonify(**resp)
```

Finally, we give this call-back message to the original query source, which is our psiTurk.computeBonus function. Trip is done, hurray!!

16.2 The basic logic of using `custom.py`

16.2.1 When is `custom.py`? called?

It is loaded as a module when the psiturk server starts (called by `psiturk/experiment.py`). That is to say, you'd need to restart psiTurk whenever you've made some change of this script!

16.2.2 What is a route and why we need it?

A route is a URL served on the server. We need it because it is impossible for javascript to run python script (or any local files) directly. But you don't have to call from javascript – equally, just access the address like `http://localhost:5000/my_route` in your browser!

(Note if `my_route` is expecting to receive arguments, like the participant ID, then the url becomes like `http://localhost:5000/my_route?id=12345`.)

16.2.3 Call the route from javascript directly without the psiturk function?

In the example above, we used the built-in function of `computeBonus` to call the custom route. Of course you can customize your own call for your favorite route, especially specifying the data sent to it. The key helper is `ajax` which is a jquery API. Add a call in your `task.js` that looks like this:

```
$.ajax("my_route",{
    type: "GET",
    data: {id: myid, data:mydata},
    success: function (response) {
        console.log(response)
    }
});
```

Note the `type` argument should be consistent with what your route function wants (usually either “GET” or “POST”). The `data` argument is usually a dictionary.

16.3 Tips about debugging your custom route

Debugging `custom.py` is tricky since the error message won't just appear in your browser console. You will most likely see an “5000 internal error” which just means there is bug when calling your route. You may, however, try the following:

- Find your error message at `server.log`, which is automatically generated in your current psiturk folder and will record the error messages. This is usually the most informative tool.
- Print messages within your python function, which will appear in the psiturk shell.
- If you are not sure the route is being called, return some error message that will show in your browser (go to your browser with `http://localhost:5000/my_route`)

ALTERNATIVE RECRUITMENT CHANNELS

How to use lab mode, how to recruit from prolific.

DEPLOY TO HEROKU

[Heroku](#) is a cloud service that lets you run applications in the cloud. You can run *psiTurk* on *Heroku* by preparing a git repository and then pushing it to *Heroku* which will deploy and autorun the code for you.

The benefits of *Heroku* include the following:

- It's somewhat easier to manage than [Amazon Web Services EC2](#) for the tech-wary (no need for security groups, no need to ssh in).
- You can set up a free PostgreSQL server (which is highly recommended to use over the default SQLite database that *psiTurk* uses). A database server is required on heroku as files, including *participants.db*, are ephemeral. Data would be lost every time the app spins down.
- You get free SSL for hosting your own ad.
- It's scalable.
- You get a *Heroku* buffering server in front of your *psiTurk* unicorn instance, which helps with performance a little bit.

One downside with *Heroku* is that it can get expensive if you need any kind of horsepower beyond 512MB memory and one node.

What follows is a step-by-step tutorial for setting up a *psiTurk* example experiment on *Heroku* (both the experiment itself and ad) with a *PostgreSQL* database for collecting data.

All commands listed in this tutorial are meant to be typed into your terminal application.

1. Go to the [Heroku website](#) and create a new account if you don't already have one.
2. Make sure that *psiTurk*, *git*, and the [Heroku Command Line Interface](#) are installed on your computer.
3.
 - **If you don't already have a psiturk experiment:**

Create a *psiTurk* example at a desired location

```
psiturk-setup-example
```

Navigate into your newly created *psiTurk* example folder:

```
cd psiturk-example
```

- **If you are starting from an already-existing psiturk project:**

Navigate to your project root directory.

4. **If your experiment is not already in a git repository:** Initialize a Git repository in the root dir of your *psiturk* project the *psiTurk* (your current working directory):

```
git init
```

5. Log in to *Heroku*, entering your heroku credentials when prompted for them:

```
heroku login
```

6. Create a new app on *Heroku*:

```
heroku create
```

Note: Running this command will add a `git remote` to your `.git/config` file, which will make it so that any *heroku* commands run from your project folder will be run against your newly-created heroku app.

7. Run the following psiturk shell command:

```
psiturk-heroku-config
```

Running this command copies all files from psiturk's `heroku_files` folder into your experiment's root directory. These are needed for your experiment to run on Heroku.

This command also runs `heroku config:set ON_CLOUD=1` in your shell on your behalf. This sets an environment variable called `ON_CLOUD` to the value 1 in your heroku app's environment. Setting `ON_CLOUD=1` in your environment tells psiturk to use [some sensible defaults](#) for several config settings. Specifically, it sets defaults for `host`, `threads`, `errorlog`, and `accesslog`.

Warning: Heads up! The `sample config.txt` file generated by psiturk 3 shows defaults in your `config.txt` commented out (prefixed with a `;`). Cloud defaults will override any defaults that are commented-out in your `config.txt`.

But if the cloud defaults are set in your `config.txt` then the cloud defaults will be overridden. To remedy this, you will need to either:

1. change them in your `config.txt` or re-comment them out, or
2. set environment variables on heroku for the corresponding cloud defaults that take precedence over your `config.txt` values.

For the latter, any of the config settings can be overridden in the heroku environment by setting `PSITURK_{uppercase_config_name}` via `heroku config:set`. For example, to override a `config.txt` `threads` on heroku, one could run the following:

```
heroku config:set PSITURK_THREADS=1
```

8. Set a database that your heroku app will use.

- **To get a free heroku-hosted postgresql database:**

Create a Postgres database on the newly created *Heroku* app:

```
heroku addons:create heroku-postgresql
```

This will provision a psiturk-compatible postgresql database, and set an environment variable on your app called `DATABASE_URL` that points to your database.

To see the `DATABASE_URL` given to you by heroku for this newly-provisioned postgresql database, you can run the following:


```
heroku config
```

Important: This URL includes your username and password. Anyone who has access to the `database_url` can connect to your database and has access to the data stored in it!

- **If you already have a publicly-accessible database hosted elsewhere:**

Then you can do one of the following:

1. list its url as your `database_url` in your `config.txt` and be sure that `DATABASE_URL` is not set in your heroku environment (check `heroku config`), or
2. set its url in your heroku environment (`heroku config:set DATABASE_URL=your-url`)

Important: psiTurk prefers environment variables over all other config file settings. Most environment settings need to prepend `PSITURK_` to the corresponding config setting name, with the exception of two environment variables:

1. `DATABASE_URL`
2. `PORT`

These two, if present in the environment, are respected even if not prepended by `PSITURK_`.

This means that if `DATABASE_URL` is set in your heroku environment, it will override any setting you have in `config.txt`.

9. **Optional:** if you want to use the [psiturk dashboard](#) from your heroku instance to run AWS some commands, or if you want your heroku instance to run any [tasks](#) created by the dashboard:

- Set your AWS credentials as environment variables within your heroku app, replacing `<XYZ>` with your access and secret keys for [Amazon Web Services](#):

```
heroku config:set aws_access_key_id=<XYZ>
heroku config:set aws_secret_access_key=<XYZ>
```

10. Stage all the files in your psiTurk example to your Git repository:

```
git add .
```

11. Commit all the staged files to your Git repository:

```
git commit -m "Initial commit"
```

12. Push the code to your *Heroku* git remote, which will trigger a build process on Heroku, which, in turn, runs the command specified in *Procfile*, which autolaunches your *psiTurk* server on the Heroku platform:

```
git push heroku master
```

Note: Any time you want to push changes to your heroku-hosted psiturk experiment, you will need to repeat the above flow of `git add`, `git commit`, `git push`.

13. You can run through your heroku-hosted experiment by visiting your heroku app's url.

To get your app’s url, run `heroku domains` from the root of your local psiturk app, and visit your app’s reported domain url in a browser. From that url, you can conveniently obtain a debugging url by clicking “Begin by viewing the *ad*.”

14. To download data from your heroku app using a locally-run psiturk, set your local psiTurk app to use the same database that your experiment uses when it runs on heroku.

To do so, get the `DATABASE_URL` of your heroku psiturk instance by running `heroku config`, and set the database url in any of the following local places:

1. your `config.txt` file, or
2. your own local environment.

Warning: If you opt to set your database url in your `config.txt` file, then be cautious about sharing your experiment code – the url contains your database username and password!

Once your local psiturk app uses the same database as your heroku app, then you can run the following to download your experiment data, regardless of whether you have run through your experiment hosted locally *or* on Heroku:

```
psiturk download_datafiles
```

This should generate three datafiles for you in your local directory:

- `trialdata.csv`,
- `questiondata.csv`, and
- `eventdata.csv`.

Congratulations, you’ve now gathered data from an experiment running on *Heroku*!

Note: psiTurk will look for a file called `.env` in the root of your psiturk app and read in any `KEY=VALUE` settings in there as environment variables for your psiturk app. Therefore, one could put the following content in a file called `.env` to set the `database_url`:

```
DATABASE_URL=url-for-your-publicly-accessible-database
```

15. To post a hit to MTurk that uses your heroku app, set your local psiTurk `config.txt`’s `ad_url` settings to point to your heroku app. The easiest way to do this is to set `ad_url_domain` in your `config.txt`’s [HIT Configuration] section to equal your heroku domain name.

For example, if running `heroku domains` reported that your heroku domain was `example-app.herokuapp.com`, then you would simply set `ad_url_domain = example-app.herokuapp.com` in your `config.txt`’s [HIT Configuration] setting. With that, HITs posted to mturk should correctly point to your heroku app.

See also:

See the `hit_configuration_ad_url` for more information.

From your *local* psiTurk session, you can now [create and modify HITs](#). When these are accessed by Amazon Mechanical Turk workers, the workers will be directed to the *psiTurk* session running on your *Heroku* app. This means that it is never necessary to launch *psiTurk* and run *server on* from *anywhere* to run an experiment on Heroku. The server is automatically running, accessible via your Heroku domain url. (Of course, if you want to debug locally, you can still run a local server.)

Note: If you stay on the “Free” Heroku tier, your app will go to “sleep” after a period of inactivity. If your app has gone to sleep, it will take a few seconds before it responds if you visit its url. It should respond quickly once it “awakens”. Consider upgrading to a “Hobby” heroku dyno to prevent your app from going to sleep.

Note: If you want to run commands against your *postgresql* db, you can run *heroku pg:psql* to connect, from where you can issue postgres commands. You can also connect directly to your heroku postgres db by installing and running *postgresql* on your local machine, and passing the *DATABASE_URL* that your heroku app uses as a command-line option.

EXAMPLE PROJECT WALKTHROUGH

Perhaps the best way to learn about psiTurk is to go through the steps of configuring and running an experiment. This tutorial will take you through the steps required to run the example experiment – a Stroop task – that ships bundled with a psiTurk installation. This project can be a great starting place for developing your own experiment.

Warning: This guide assumes you already have the psiTurk command line tool installed on your computer. If you haven't you should begin there and come back when it is installed. Instruction [here](#).

This guide also assumes you are using version 1.0.10dev or higher of the psiTurk command line tool. Type `psiturk --version` in your command shell/terminal program to verify your version number.

19.1 Background

The Stroop effect is the finding that people show interference from reading while naming the font color of words. The task is used to suggest that reading has become a highly “automatic” cognitive skill. You can read more about the Stroop task [here](#). This guide won't comment much on the psychology of it, rather focusing on the technical aspect of running such an experiment online that consists of a sequence of trials and which records response time and key presses.

19.2 Initialize the demo code

The first step is to obtain the archive of code and resources specific to the Stroop demo. Additional experiments are shared on the psiTurk experiment exchange. However, the Stroop demo comes bundled within the psiturk command line tool.

First use the `psiturk-setup-example` command to place fresh copies of the files into a new folder:

```
$ psiturk-setup-example
Creating new folder `psiturk-example` in the current working directory
Copying /Users/gureckis/Library/Enthought/Canopy_64bit/User/lib/python2.7/site-
→packages/PsiTurk-1.0.10dev-py2.7.egg/psiturk/example to ./psiturk-example
Creating default configuration file (config.txt)
```

afterward you should have a new folder in the current directory named “psiturk-example” with the following listing of files:

```
$ cd psiturk-example
$ ls -la
```

(continues on next page)

(continued from previous page)

```
total 16
drwxrwxr-x   6 gureckis  staff   204 Mar 31 12:18 .
drwx----- 23 gureckis  staff   782 Mar 31 12:18 ..
-rw-r--r--   1 gureckis  staff   796 Mar 31 11:55 config.txt
-rw-r--r--   1 gureckis  staff  3226 Mar 31 11:55 custom.py
drwxrwxr-x   9 gureckis  staff   306 Mar 31 12:18 static
drwxrwxr-x  19 gureckis  staff   646 Mar 31 12:18 templates
```

See also:

A full description of the individual files is provided [here](#). A few of the files described on the full documentation will not appear until the first time you start `psiturk` and launch the `psiTurk` server.

19.3 Set Your AWS Credentials

To access Amazon Mechanical Turk and other Amazon Web Services features, you need to set your AWS Credentials and also your default AWS region (see [these instructions](#) for details).

19.4 Configure the option for the demo experiment

Another of the files generated by `psiturk-setup-example` is the `config.txt` file, which contains a variety of experiment and server parameters. These values can be changed by altering the file in any text editor.

The default `config.txt` file is already mostly configured to help you test the Stoop demo. Three options you might want to adjust to begin with are:

1. In the **[Server Parameters]** section ensure that the port listed is one that is available on your computer (answer is usually yes unless you have particular firewall software running).
2. In the **[Server Parameters]** section ensure that the host is either `localhost` (if just testing/debugging locally) or set to `0.0.0.0` (if planning to test live on the AMT site).

See also:

A full description of the local configuration file and the meaning of the various option is available [here](#).

19.5 Launch the psiTurk shell

All user commands to `psiTurk`, such as creating a HIT, launching the experiment server, or approving workers, are issued through the `psiTurk` command. To open the shell, run `psiturk` in a valid experiment folder. You should see something like this (though probably colored on your display):

```
$ psiturk

http://psiturk.org

  /\  ==  /\  _/\  /\  _/\  /\  _/\  /\  ==  /\  _/\  /\
 \/\  _/\  \/\  _/\  \/\  _/\  \/\  _/\  \/\  _/\  \/\  _/\
 \/\  _/\  \/\  _/\  \/\  _/\  \/\  _/\  \/\  _/\  \/\  _/\
 \/\  _/\  \/\  _/\  \/\  _/\  \/\  _/\  \/\  _/\  \/\  _/\
```

(continues on next page)

(continued from previous page)

```

an open platform for science on Amazon Mechanical Turk

-----
System status:
Hi all, You need to be running psiTurk version >= 1.0.5dev to use the
Ad Server feature!

Check https://github.com/NYUCCL/psiTurk or http://psiturk.org for
latest info.
psiTurk version 1.0.10dev
Type "help" for more information.
[psiTurk server:off mode:sdbx #HITS:0]$

```

The psiTurk shell prompt displays several useful pieces of information: whether the experiment server is on, whether you are in sandbox or live mode, and how many hits are online in your current mode (more on all of these below). While in the psiTurk shell, all commands entered will be executed by psiTurk. To exit the shell, type `quit`.

See also:

`command-line-overview`

19.6 Start/stop the experiment server

The psiTurk experiment server is a separate process that acts as a custom, local web server. To launch the server type `server on` in the command line interface:

```

[psiTurk server:off mode:sdbx #HITS:0]$ server on
Experiment server launching...
Now serving on http://localhost:
[psiTurk server:on mode:sdbx #HITS:0]$

```

Note that the command prompt has changed from showing `server:off` to `server:on` in this example (and also changed from red to green on colorized terminals). You can start or stop the server at any time using the `server on` and `server off` commands. Typically you want to have the server running when you are testing locally, testing on the AMT “sandbox”, or running your actual experiment. If the server stops when running your actual experiment, Internet users will no longer be able to participate in your experiment even if you still have HITS posted on AMT’s website. Thus, you should think of the experiment server as meaning your experiment is current “live.”

19.7 Debug/test the experiment locally

Frequently you would like to test your experiment in your browser locally without involving Amazon’s servers at all. To do so, ensure that the experiment server is running (the prompt should show `server:on`). Then enter the command `debug`. A new browser tab will open with the first screen of the experiment. The URL string for this will look something like this:

```

http://localhost:22362/ad?assignmentId=debug7FIXMF&hitId=debugI3XW1P&
↪workerId=debugY3UNQY

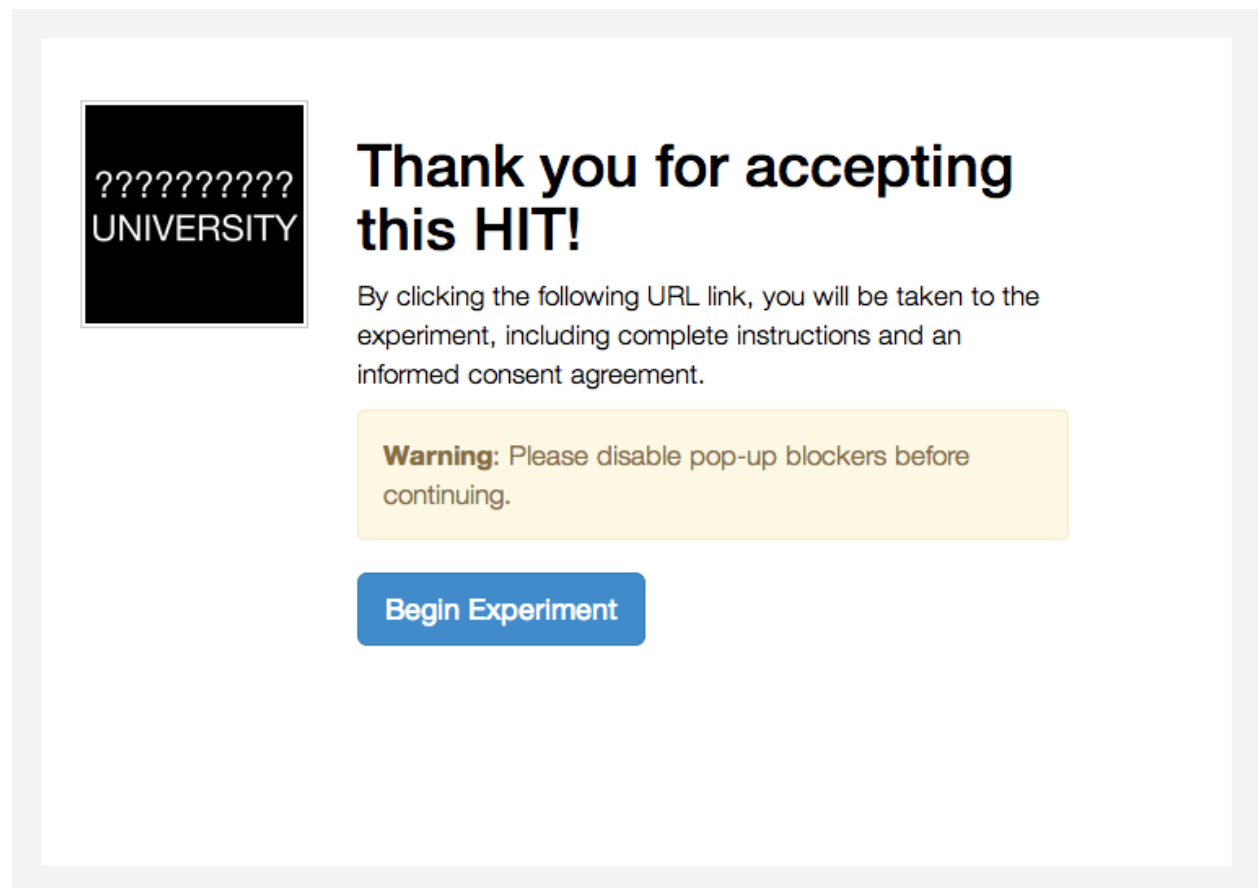
```

The `http://localhost:22362/` part is set in the configuration options under `Server Parameters` in the fields “host” and “port”. The default value, `http://localhost:22362/` is a special term that refers to your own computer. As mentioned above, if you wanted to run this experiment publically you would want to change the host option to `0.0.0.0`.

The remaining part of the URL created random (i.e., fake) identifiers which stand-in for the values that Amazon provides identifying the user, hit, etc... Since by default psiTurk does not allow individuals to take the same experiment more than once (it checks for you to see if the worker has already completed the task or read too far into the instructions) these random values are helpful during debugging.

Important: When running in debug mode (i.e., when the `assignmentId`, `hitId`, and `workerId` variables are prefixed with the word “debug”) everything proceeds as usual. However, the server will not block the same user from restarting the experiment after finishing the instructions (as is true normally). This helps debugging since you don’t have to keep inventing new fake `workerId`. However, good to keep in mind this difference.

The first page that you see in the experiment looks something like this:



This is the page the AMT worker would see when they first accept the hit. When you click the link, a full screen window will open up which will run the experiment. You can test it now if you like just to get a sense of things. If you want to stop midway through that is no problem. Just close that browser window. Running debug again will open a new browser window and let you repeat the process.

Important: In the typical development cycle you would make changes to the javascript, CSS, or HTML files in your project locally and use `debug` to see those changes and test them. This way the development environment is the same as the one in which you will eventually deploy your experiment on Mechanical Turk.

19.8 Experiment Structure

The basic stroop demo lays out a pretty standard experiment sequence. It is perhaps most helpful to step through this sequence yourself, but conceptually:

First the users view an “ad” for the study (that is what is displayed above).

Then they view a consent form and are asked to verify that they read and understood the consent.

Next they are given a sequence of instruction screens. The experiment logs how long they look at the each instruction screen as well as if they shift back and forth using the next/previous buttons.

Then the main experiment begins which dynamically re-draws the browser window using Javascript. The `psiturk.js` API records the data and synchronizes it with your server from time to time.

After the experiment finishes the user is given a simple questionnaire about their experiences in the task. Finally control is returned to Amazon (or if debugging a stand-in message is displayed).

While all this is going on the `psiturk.js` API records if the user is changing windows and prevent them from reloading the browser mid-way into the task to start over.

19.9 Launch in AMT sandbox

Now that you’ve tested the experiment locally, you may want to see how it would appear on mturk before running it live with paid workers. Amazon offers this ability through the worker sandbox – a simulated environment that allows developers to test their HITs.

To create a hit in the worker sandbox, first check that the server is on and that you are in sandbox mode; the psiTurk prompt should say on next to server and `sdbx` next to mode. If you are in `live` mode, enter the command `mode` to switch to sandbox mode. If you are in `live` mode it will post your task to the live, paid AMT website instead of the free demo site.

When you are in sandbox mode if you type `amt_balance` you will see you have a never ending account with \$10,000.00 of fake money to spend on sandbox HITs.

```
[psiTurk server:on mode:sdbx #HITs:0]$ amt_balance
$10,000.00
```

To create a hit, enter the command `hit create`, and then answer the prompts to set up the HIT. Your choices for the prompt answers are arbitrary for now, since the HIT won’t be completed by real workers. If the `host` variable in the `config.txt` file for this project is set to `localhost` (default) or `127.0.0.1` you will get an error reminding you that you server is no accessible to the general Internet. Please change this option before trying to post your task on AMT.

```
[psiTurk server:on mode:sdbx #HITs:0]$ hit create
number of participants? 5
reward per HIT? 1.00
duration of hit (in hours)? 1
*****
Creating sandbox HIT
  HITid: 3SA4EMRVJV2ALPN29ZGP6BDPNBS0P0
  Max workers: 5
  Reward: $1.00
  Duration: 1 hours
  Fee: $0.50
  _____
```

(continues on next page)

(continued from previous page)

```
Total: $5.50
Ad for this HIT now hosted at: https://ad.psiturk.org/view/oyG8sMCn9ySLTTrumsYgHe?
↪assignmentId=debugFOFTCL&hitId=debugTSXLIB
```

This example create a hit with 5 “slots” for participants (or 5 assignments). The reward is \$1.00 and the participant has 1 hour to complete the task after accepting the HIT before it will be returned. Finally the unique “ad” for this experiment/HIT is displayed at the bottom.

You can also run `create_hit` non-interactively by providing arguments when you run the command, for example `create_hit 10 1.00 4`.

You should now see the number “1” next to “#HITs:” in the psiTurk prompt, denoting that you have one active HIT in the worker sandbox. If you type the command `hit list active`, you should see a description of your HIT including the HIT id:

```
[psiTurk server:on mode:sdbx #HITs:1]$ hit list active
Stroop task
  Status: Assignable
  HITid: 3SA4EMRVJV2ALPN29ZGP6BDPNBSOP0
  max: 5/pending: 0/complete: 0/remain: 5
  Created: 2014-03-31T21:32:27Z
  Expires: 2014-04-01T21:32:27Z
```

To test your HIT, go to the worker sandbox and search for your HIT by entering the name of your requester account in the search bar. You should see something like this:

The screenshot shows the Amazon Mechanical Turk Developer Sandbox interface. At the top, a banner states: "You are using the Mechanical Turk Developer Sandbox. This site is for test and development only. [Learn more](#)". Below this, the Amazon Mechanical Turk logo is visible. The main navigation bar includes "Your Account", "HITS", and "Qualifications". A notification indicates "361,162 HITs available now". The search bar shows "Find HITs" with a dropdown menu, and the search criteria are "containing NYU computational cognition lab" and "that pay at least \$ 0.00". The results section shows "HITS containing 'NYU computational cognition lab'" with "1-1 of 1 Results". The results are sorted by "HITS Available (most first)". A table displays the details of the HIT:

Stroop task		View a HIT in this group	
Requester: NYU Computational Cognition Lab	HIT Expiration Date: Nov 25, 2013 (23 hours 59 minutes)	Reward: \$1.00	
	Time Allotted: 4 hours	HITS Available: 1	

At the bottom, there is a footer with links for "FAQ", "Contact Us", "Careers at Amazon", "Developers", "Press", "Policies", and "Blog". The copyright notice is "©2005-2013 Amazon.com, Inc. or its Affiliates". The Amazon logo is also present.

Click “view a HIT in this group” to open a hit. You should see an ad for your HIT appear on the screen. Click “accept HIT”, then click the link in the HIT ad to open the experiment in a full-screen window. If you complete the HIT in this manner you it should go through all the steps of the AMT process. Afterwards you will have some data in your database.

19.10 Accessing your data

The simplest way to retrieve data is using the `download_datafiles` command. This creates three csv files containing the three kinds of data: `trial data`, `question data`, and `event data`.

If you are using the default SQLite database (see [configuring databases](#)) then another option is to use a GUI tool like `Base` to access the data in the `participants.db` file in your project folder.

If you set your database to use MySQL then you may be able to connect and export the data using `Sequel Pro`.

19.11 Automatically computing a bonus

See *Example: Automatically computing performance bonus*.

19.12 Approve/Reject Workers

19.13 Assigning bonuses

19.14 Launch “live” experiment

To launch an experiment “live” you follow the same steps as launching in the sandbox but first set the “mode” of the command line to “live”:

```
[psiTurk server:on mode:sdbx #HITS:1]$ mode
Switching modes requires the server to restart. Really switch modes? y or n: y
Entered live mode
Shutting down experiment server at pid 55158...
Please wait. This could take a few seconds.
Experiment server launching...
Now serving on http://0.0.0.0:22362
[psiTurk server:on mode:live #HITS:0]$
```

Now if you run `hit create` it will post a hit on the live website. You must have enough money in your AMT account to pay for the HITs you are requesting, otherwise an error message will be displayed. The `amt_balance` command will let you check your current balance:

```
[psiTurk server:on mode:live #HITS:0]$ amt_balance
$178.70
```

Danger: Remember to switch back to “sandbox” mode when you are finished collecting data so that the command you type will not accidentally create tasks that will charge you account money!

19.15 Conclusion

This concludes the conceptual overview of the Stroop example that ships with psiTurk.

USING JSPSYCH+PSITURK

Some hints about the integration.

HOSTING YOUR OWN HTTPS EXPERIMENT

With support for the psiTurk Secure Ad server dropped from psiturk 3, you may wonder how you will now serve ads securely in order to run your experiments on platforms such as AWS Mturk which still require that your experiment “ads” be served over HTTPS.

If you have been hosting your own experiments on your own server using a static IP address, one option is to buy a domain name, and to let a DNS provider such as cloudflare proxy HTTPS for you. With this approach, Cloudflare sits in between your participants and your psiturk server. Cloudflare provides a valid HTTPS certificate to participants’ browsers, enabling your ads to be served over HTTPS. Then, Cloudflare can connect to your psiturk server either over HTTP (unencrypted), or over HTTPS (encrypted). For the latter, you would configure your psiturk server to either use a self-signed certificate, or to use a certificate provided by free by Cloudflare. Regardless, participants’ browsers will view the connection as secure.

Regardless of whether a proxied connection between Cloudflare and your psiturk server is HTTP or HTTPS, Cloudflare has plaintext access to all of the data being sent between the participants’ browsers and your psiturk server.

The first option – Cloudflare <- HTTP -> Psiturk server – is called “Flexible” encryption. With this option, your psiturk server **must run on port 80**. This means that, unless you are using a reverse proxy on your server such as nginx or apache in front of your psiturk servers, you can only have one psiturk experiment running on a given server at a time. The second – Cloudflare <- HTTPS -> Psiturk server – is called “Full” encryption. Read [Cloudflare’s page on proxied HTTPS](#) for more information.

Note: It is good practice to configure a reverse proxy server such as nginx or apache on your servers to (1) handle serving static files, taking that load off of your psiturk server and letting it handle running python routes, and (2) to more easily serve multiple experiments on one server.

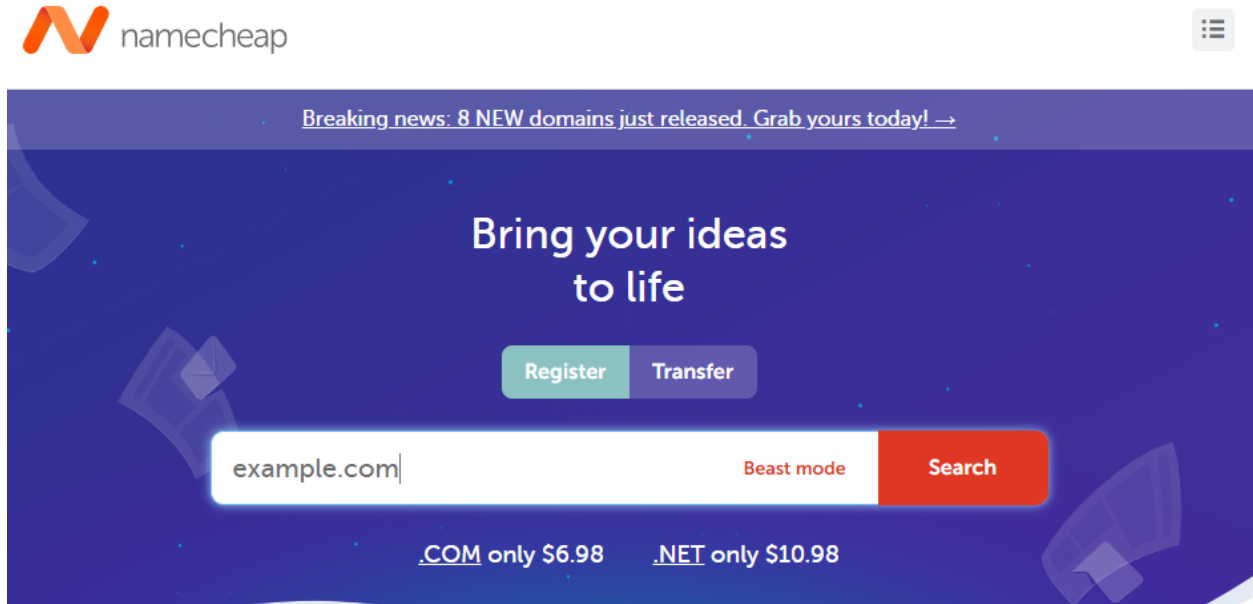
For the latter, reverse proxy servers can route to different psiturk servers depending on the domain name requested. For running multiple psiturk servers off of the same static IP, you might choose to set several subdomains in your DNS settings, each pointing to your same single IP address, but each associated with a different psiturk experiment.

Note: For any experiment that has used the psiturk secure ad server in the past without configuring SSL support for the psiturk server (unicorn), it has always been the case that participant network traffic travels between the psiturk server and participants’ browsers over HTTP (in plaintext). Only the *ad* traffic was ever encrypted, to satisfy AWS Mturk requirements. By design, the Psiturk Ad Server handed off participants to psiturk server’s static ip addresses over HTTP when participants clicked “Begin the study.”

Setting up Cloudflare Flexible encryption is done as follows. The example below uses the imaginary domain name **example.com** and imaginary static IP address **192.0.2.1**.



21.1 Purchase a domain name from a domain name registrar

You must first buy a domain name. These are inexpensive – usually less than \$10 per year per domain. Use a domain name registrar such as [namecheap](), and buy any domain you like.




Note that you cannot purchase domains directly from Cloudflare, although you *can* transfer domain names to Cloudflare after a waiting period.

21.2 Create an account on Cloudflare if necessary



Get started with Cloudflare



Email

Password  Show

By clicking Create Account, I agree to Cloudflare's [terms](#), [privacy policy](#), and [cookie policy](#).

Create Account

21.3 Add a new “site” to your Cloudflare account with the name of your purchased domain name

 Menu ▼

Home Members Audit Log Billing Configurations Notifications Registrar

+ Add a Site



21.4 Set Cloudflare to be your DNS provider

Cloudflare will tell you that to complete the process, you must visit the site of your domain name registrar and set Cloudflare to be your DNS provider. Once you have done so, this transfer process may take a few hours to complete. The change has to propagate throughout the DNS servers of the world.

Cloudflare generally recognizes who your domain name registrar is based on the DNS servers your registrar auto-configured for your domain. Cloudflare will generally point you to registrar-specific instructions for how to configure your domain to use their DNS servers. If they don't, try a general internet search for "cloudflare [name of registrar] set dns"

21.5 Create a DNS A record



On the Cloudflare page for your site, in the DNS tab, set an A record pointing your domain name to your static ip address.

 example.com ▼

◀ Overview Analytics **DNS** SSL/TLS Firewall Access Speed Caching Wc ▶

DNS management for **example.com**

+ Add record

 Search DNS Records  Advanced

example.com points to **192.0.2.1** and has its traffic proxied through Cloudflare.

Type

A ▼

Name

example.com


IPv4 address

192.0.2.1

TTL

Auto

Proxy status

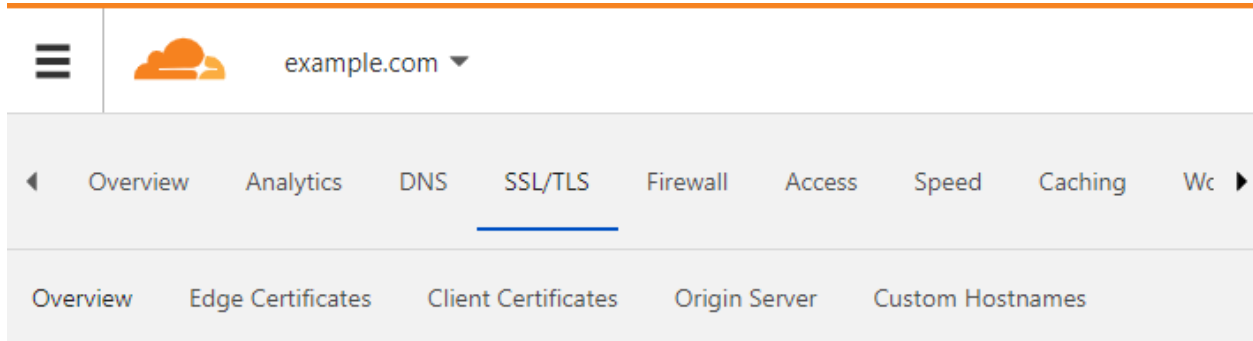
 Proxied

Cancel Save

Make sure that this record is cloudflare-proxied. (It should have an orange cloud, not a grey cloud.)

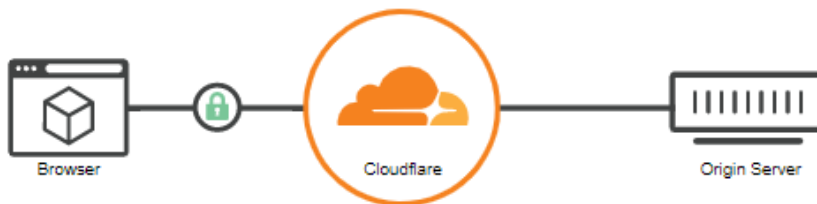
21.6 Enable Flexible SSL

In cloudflare, enable [flexible SSL](#) for your domain.



✓ Your SSL/TLS encryption mode is Flexible

This setting was last changed 4 days ago



- ☐ Off (not secure) ⓘ
No encryption applied
- ☒ **Flexible**
Encrypts traffic between the browser and Cloudflare
- ☐ Full
Encrypts end-to-end, using a self signed certificate on the server
- ☐ Full (strict)
Encrypts end-to-end, but requires a trusted CA or Cloudflare Origin CA certificate on the server

Learn more about [End-to-end encryption with Cloudflare](#)

21.7 Start your psiTurk server

Set your psiturk server to run on `host=0.0.0.0` and `port=80`. Then, turn on your psiturk server:

```
/psiturk-experiment-directory$ psiturk server on  
Now serving on http://0.0.0.0:80
```

Visit your domain name in your browser, using your psiturk server's port. You should see a secure connection.

If posting a HIT to mturk using the above settings, the `hit_configuration_ad_url_ad_url_domain` would be set to `example.com`, and the defaults of `hit_configuration_ad_url_ad_url_port`, `hit_configuration_ad_url_ad_url_protocol`, and `hit_configuration_ad_url_ad_url_route`.

USING COMMERCIAL SURVEY TOOLS

With the magic of `iframes` and javascript window messaging, you can integrate external survey tools into your psiTurk experiment. This is possible as long as the survey tool allows custom javascript to be triggered.

Window messaging allows cross-domain messaging via javascript, without having to configure security settings on the server. [MDN](#) says it best:

“The `window.postMessage` method safely enables cross-origin communication. Normally, scripts on different pages are allowed to access each other if and only if the pages that executed them are at locations with the same protocol (usually both `https`), port number (443 being the default for `https`), and host (modulo `document.domain` being set by both pages to the same value). `window.postMessage` provides a controlled mechanism to circumvent this restriction in a way which is secure when properly used.”

Three special steps to hook up your survey to psiTurk:

1. Embed your survey as an `iframe` within one of your psiTurk pages or views.
2. Add a message event listener to your psiTurk window
3. Post a message from the survey tool to the `window.top` when the survey is complete. `window.top` will be your psiTurk window. Do whatever you want via javascript once you receive the expected message.

To tie the psiTurk data and the external survey data together, embed a unique id into the `iframe` url you load, and then record that unique url into your survey data. *Don't forget to do this. If you forget, you won't know to who to connect your survey data.* If you want to tie things both ways, post back your survey session id as part of the survey-complete post-back.

22.1 An example with Qualtrics

As of the time this documentation page was written, Qualtrics has an undocumented “feature”. Qualtrics automatically posts a window message to `window.top` when the Qualtrics “end of the survey event” is triggered. For Qualtrics surveys embedded as `iframes` in psiTurk experiments, we can take advantage of this behavior. The Qualtrics-posted message contains your `survey_id` and the participant's Qualtrics-created unique `session_id`. You should already know the `survey_id` (because you just embedded a link containing this id), but the `session_id` is Qualtrics's unique id for whoever just finished your survey. You can record that with psiTurk as unstructured data (see [Recording unstructured data](#)) if you desire.

Don't forget to explicitly log the psiTurk unique id as embedded data within Qualtrics. See [here](#) for more about embedding data into Qualtrics surveys.

The posted message when they finish a qualtrics survey is a string that looks like this:

```
QualtricsEOS|<survey_id>|<qualtrics_session_id>
```

So you can do something like this on your psiTurk page:

```
// load your iframe with a url specific to your participant
$('#iframe').attr('src', '<your qualtrics url>&UID=' + uniqueId);

// add the all-important message event listener
window.addEventListener('message', function(event){

    // normally there would be a security check here on event.origin (see the MDN_
    ↪link above), but meh.
    if (event.data) {
        if (typeof event.data === 'string') {
            q_message_array = event.data.split('|');
            if (q_message_array[0] == 'QualtricsEOS') {
                psiTurk.recordTrialData({'phase':'postquestionnaire', 'status':'back_
                ↪from_qualtrics'}));
                psiTurk.recordUnstructuredData('qualtrics_session_id', q_message_
                ↪array[2]);
            }
        }
    }
    // display the 'continue' button, which takes them to the next page
    $('#next').show();
})
```

This code can be put on a page that has a link with id #next default-hidden via css which advances the participant to the next experimental page. Note that this code checks that the event is QualtricsEOS before continuing on. That's because Qualtrics posts other events to window.top, too. This code is only interested in the EndOfSurvey event.

Also notice that this code doesn't implement any security precautions. Normally it's good practice to check to see where a message is coming from before you act on it. For instance, it might check to verify that the message is coming from a qualtrics.com domain. But in this code, the worst-case scenario is that a tech-savvy participant somehow triggers that they completed the survey before they actually did. In that case, their survey data would be blank, and after visual inspection their assignment could be rejected.

22.2 What about not-Qualtrics?

If your survey tool isn't posting messages to window.top for you, just window.top.postMessage(<message>, <targetOrigin>) yourself. For instance, you might have javascript in your survey tool that does:

```
window.top.postMessage("all_done|<survey_session_id>", "*")
```

Then just listen for that event back on your psiTurk page, as in the Qualtrics example above.

AUTOMATICALLY COMPUTE BONUSES

Options are 1. in real time or 2. offline ()

PSITURK.JS API

Everything in the **psiturk.js** API is scoped under the `psiturk` namespace.

24.1 Creating the psiTurk object

To use the psiTurk library, a `psiturk` object must be created at the beginning of your experiment. It takes two key arguments `uniqueId` and `adServerLoc`. These two variables are first created in *exp.html* `<file_desc/exp_html.html>`. They tell psiTurk which unique number/code corresponds to the current participant (allowing updating of data as the task progresses) and the location of the `ad` where users should be sent when the task is complete.

```
// Create the psiturk object
var psiTurk = PsiTurk(uniqueId, adServerLoc);

// Add some data and save
psiturk.addUnstructuredData('age', 24)
psiturk.saveData();
```

The following documents the javascript API.

24.2 `psiturk.taskdata`

`taskdata` is a `Backbone model` used to store all data generated by a participant and to sync it to the database.

`taskdata` has the following fields with these default values:

```
condition: 0
counterbalance: 0
assignmentId: 0
workerId: 0
hitId: 0,
useragent: ""
currenttrial: 0
data: ""
questiondata: {}
eventdata: []
```

These variables are either set during initialization or using the methods of the `psiturk` object. However, since `taskdata` is a `Backbone model`, you can always access their values directly using the Backbone ``set`` <https://backbonejs.org/#Model-set> ``__`` and ``get`` <https://backbonejs.org/#Model-get> ``__`` methods, which may be useful for debugging. For example:

```
psiturk.taskdata.set('condition', 2);
psiturk.taskdata.get('condition');
```

24.3 psiturk.preloadPages (pagelist)

For each path in `pagelist`, this will request the html and store in the `psiturk` object. A given page can then be loaded later using `psiturk.getPage (pagename)`.

Returns a `Promise`. See the `example task.js` for a full usage example.

Example:

```
// Preload a set of HTML files
async function example() {
  await psiturk.preloadPages(['instructions.html', 'block1.html', 'block2.html']);

  // Set the content of the body tag to one of the pages
  $('body').html(psiturk.getPage('block1.html'));
}

example()
```

24.4 psiturk.getPage (pagename)

Retrieve a stored HTML object that has been preloaded using `psiturk.preloadPages`.

24.5 psiturk.showPage (pagename)

Set the BODY content using an HTML object that has been preloaded using `psiturk.preloadPages`.

Example:

```
async function example() {
  psiturk.preloadPages(['instructions.html', 'block1.html', 'block2.html']);
  psiturk.showPage('instructions.html');
}

example()
```

24.6 psiturk.preloadImages (imagelist)

Cache each image in `imagelist` for use later.

24.7 `psiturk.recordTrialData(datalist)`

Add a single line of data (a list with any number of entries and any type) to the `psiturk` object. Using this will *not* save this data to the server, for that you must still call `psiturk.saveData()`.

Example:

```
// data comprised of some list of variables of varying types
data = ['output', condition, trialnumber, response, rt];
psiturk.recordTrialData(data);
```

24.8 `psiturk.recordUnstructuredData(field, value)`

Add a (field, value) pair to the list of unstructured data in the task data object.

Example:

```
psiturk.recordUnstructuredData('age', 24);
```

24.9 `psiturk.saveData([callbacks])`

Sync the current psiTurk task data to the database.

An optional argument `callbacks` can provide functions to run upon success or failure of the saving.

```
psiturk.saveData({
  success: function() {
    // function to run if the data is saved
  },
  error: function() {
    // function to run if there was an error
  }
});
```

24.10 `psiturk.completeHIT()`

This finishes the task by passing control of the experiment back to the *Secure Ad Server* `<secure_ad_server.html>`. When in debug mode this just cleans up the task. When running live on the sandbox or live site this passes control of the browser back to the Ad Server so that the subject can be marked as complete and the user's browser will correctly finish the HIT on Amazon's site.

24.11 psiturk.doInstructions(pages, callback)

psiTurk includes a basic method for showing a sequence of instructions. You are always free to write your own instructions code (and may need to). However, this provides a basic template for a pretty simple typical type of instructions composed of a sequence of multiple pages of text and graphics along with a “next” and (optionally) “previous” button.

The `doInstructions()` method takes two arguments. The first is a list of HTML pages that you would like to display. These should appear in the order you would like them to be displayed to participants. The instructions method uses the `showPage()` method to display the HTML of the page.

Prior to calling `doInstructions()` all the instruction pages you plan to display should be preloaded using the `preloadPages()` method.

Within each HTML page there should be a button or other HTML element with class equal to `continue` which the user can click to move to the next screen.

An [Bootstrap](#) example is:

```
<button type="button" id="next" value="next" class="btn btn-primary btn-lg continue">
  Next <span class="glyphicon glyphicon-arrow-right"></span>
</button>
```

In addition, if the HTML document includes an element with class `previous` it will, when clicked, go to the previous page. As a result you should not include a previous button on the first HTML page.

An example previous button using [Bootstrap](#) is:

```
<button type="button" id="next" value="next" class="btn btn-primary btn-lg previous">
  <span class="glyphicon glyphicon-arrow-left"></span> Previous
</button>
```

The final argument to the instructions object is the method to be called when the “continue” button on the last page of the instructions is called.

Example

```
async function start_experiment() {
  psiturk = new PsiTurk(uniqueId, adServerLoc);

  var pages = [
    "instructions/instruct-1.html",
    "instructions/instruct-2.html",
    "instructions/instruct-3.html"];
  await psiturk.preloadPages(pages); // preload the pages

  var instructionPages = [ // any file here should be preloaded first
    "instructions/instruct-1.html",
    "instructions/instruct-2.html",
    "instructions/instruct-3.html"]; // however, you can have as many as you like
  psiturk.doInstructions(instructionPages,
    function() { currentview = new StroopExperiment(); });
}
start_experiment()
```

The last line in this example uses an anonymous function to launch the [Stroop Experiment](#).

24.12 psiturk.finishInstructions()

`finishInstructions` is used to change the participant's status code to 2 in the database, indicating that they have begun the actual task.

In addition, this removes the `beforeunload` handler such that if people attempt to close (or reload) the page, they will get an alert asking them to confirm that they want to leave the experiment.

You do not have to use `doInstructions()` in order to call `finishInstructions()`. In the example above you would want to call `psiturk.finishInstructions()` in the `StroopExperiment()` class.

Example

```
psiturk = new PsiTurk(uniqueId, adServerLoc);  
...  
psiturk.finishInstructions();
```


PSITURK COMMANDS

Each of these commands can be run either from an interactive shell, or as arguments to the `psiturk` command (e.g., `psiturk amt_balance` or `psiturk hit create 1 0.01 1` from a bash prompt).

See also:

`command-line-overview`

Commands

- `amt_balance`
- `config`
- `debug`
- `download_datafiles`
- `help`
- `hit`
- `worker`
- `quit`
- `server`
- `status`
- `mode`

25.1 amt_balance

Displays your current AMT balance, or your worker sandbox balance (always \$10,000.00) if you are in sandbox mode.

An example of checking your balance in sandbox mode:

```
[psiTurk server:off mode:sdbx #HITs:1]$ amt_balance
$10,000.00
```

25.2 config

Used with a variety of subcommands to control the current configuration context.

Commands

- `config print`
- `config reload`
- `config help`

25.2.1 config print

Prints the current configuration context.

Example:

```
[psiTurk server:off mode:sdbx #HITs:0]$ config print
[Server parameters]
threads=auto
...
[Shell parameters]
launch_in_sandbox_mode=true
[psiTurk server:on mode:sdbx #HITs:0]$
```

25.2.2 config reload

Reloads the current config context (both local and global files). This will cause the server to restart.

Example:

```
[psiTurk server:on mode:sdbx #HITs:0]$ config reload
  Reloading configuration requires the server to restart. Really reload? y or n: y
  Shutting down experiment server at pid 82701...
  Please wait. This could take a few seconds.
  Experiment server launching...
  Now serving on http://localhost:22362
[psiTurk server:off mode:sdbx #HITs:0]$
```

25.2.3 config help

Display a help message concerning the config subcommand.

25.3 debug

Makes it possible to locally test your experiment without contacting Mechanical Turk servers. Type `debug` to automatically launch your experiment in a browser window. The server must be [running](#) to debug your experiment. When debugging, the server feature that prevents participants from reloading the experiment is disabled, allowing you to make changes to the experiment on the fly and reload the debugging window to see the results.

- `debug -p, --print-only`

Use the `-p` flag to print a URL to use for debugging the experiment, without attempting to automatically launch a browser. This is particularly useful if your experiment server is running remotely.

Example using the `-p` flag to request a debug link:

```
[psiTurk server:on mode:sdbx #HITS:0]$ debug -p
Here's your randomized debug link, feel free to request another:
http://localhost:22362/ad?assignmentId=debugDKSAAE&hitId=debug2YW8RI&
↪workerId=debugM1QUH4
[psiTurk server:on mode:sdbx #HITS:0]$
```

25.4 download_datafiles

Accesses the current experiment database table (defined in `config.txt`) and creates a copy of the experiment data in a csv format. `download_datafiles` creates three files in your current folder:

- `eventdata.csv`

Contains events such as window-resizing, and is formatted as follows:

column 1	column 2	column 3	column 4	column 5
unique user ID	event type	interval	value	time

- `questiondata.csv`

Contains data recorded with `psiturk.recordUnstructuredData()`, and is formatted as follows:

column 1	column 2	column 3
unique user ID	question name	response

- `trialdata.csv`

Contains data recorded with `psiturk.recordTrialData()`, and is formatted as follows:

column 1	column 2	column 3	column 4
unique user ID	trial #	time	trial data

Note: More information about how to record different types of data in an experiment can be found [here](#).

25.5 help

Usage:

```
help
help <command>
```

The `help` command displays a list of valid `psiturk` shell commands. Entering `help` followed by the name of a command brings up information about that command.

Examples:

1. List all commands:

```
[psiTurk server:on mode:sdbx #HITS:0]$ help
```

psiTurk command help:

\=

amt_balance	debug		mode		server	
config	download_datafiles	open		setup_example	version	
db	hit		psiturk_status	status	worker	

basic CMD command help:

\=

EOF	ed	help	li	py	run	shortcuts
_load	edit	hi	list	q	save	show
_relative_load	eof	history	load	quit	set	
cmdenvironment	exit	l	pause	r	shell	

psiTurk commands are listed first, followed by commands inherited from the python *cmd2* module. More information about *cmd2* commands can be found [here](#).

2. View the help menu for a command and its subcommands

```
[psiTurk server:on mode:sdbx #HITS:0]$ help server
```

Usage:

- server on
- server off
- server restart
- server log
- server help

'server' is used with the following subcommands:

- on Start server. Will not work if server is already running.
- off Stop server. May take several seconds.
- restart Run 'server off', followed by 'server on'.
- log Open live server log in a separate window.
- help Display this screen.

Note: With commands with subcommands such as `server`, you can also view the help screen by entering

<command> help. For example, `server help` has the same effect at `help server`.

25.6 hit

The `hit` command is used to create, view, delete, and modify Human Intelligence Tasks (“HITs”) on Amazon Mechanical Turk.

Commands

- `hit create`
- `hit extend`
- `hit expire`

25.6.1 hit create

Usage:

```
hit create [<numWorkers> <reward> <duration>]
```

Create a HIT with the specified number of assignments, reward amount, and duration. Will be posted either live to AMT or to the Worker Sandbox depending upon your current mode. `hit create` can also be run interactively by entering the command without parameters.

The `duration` specifies how long a worker can “hold on” to your HIT (in hours or hours.<fraction_of_hour>). This should be long enough for workers to actually complete your HIT, but sometimes workers will “accept” a HIT which is worth a lot of money but come back and do the work later in the day. You can specify a shorter duration if you want workers to complete your HIT immediately.

Example of creating a HIT in the sandbox with three assignments that pays \$2.00 and has a 1.5 hour time limit:

```
[psiTurk server:on mode:sdbx #HITS:0]$ hit create 3 2.00 1.5
*****
Creating sandbox HIT
  HITid:  2XE40SPW1INMXUF9OJUNDB6BT8W2F4
  Max workers: 3
  Reward: $2.00
  Duration: 1.5 hours
  Fee: $0.60

  Total: $6.60
  Ad for this HIT now hosted at: https://ad.psiturk.org/view/Q3HWnfqzg3MP9VDbu3kFyn?
  ↳assignmentId=debugJCI80S&hitId=debug9AWC90
[psiTurk server:on mode:sdbx #HITS:1]$
```

25.6.2 hit extend

Usage:

```
hit extend <HITid> [--assignments <number>] [--expiration <time>]
```

Extend an existing HIT by increasing the amount of time before the HIT expires (and is no longer available to workers) or by increasing the number of workers who can complete the HIT.

Example adding both time and assignments to a HIT:

```
psiTurk server:on mode:sdbx #HITs:1]$ hit list --active
Stroop task
  Status: Assignable
  HITid: 2776AUC26DG6NRIGNVRFN0COYO0B4R
  max:3/pending:0/complete:0/remain:3
  Created:2014-03-07T21:36:33Z
  Expires:2014-03-08T21:36:33Z

[psiTurk server:on mode:sdbx #HITs:1]$ hit extend 2776AUC26DG6NRIGNVRFN0COYO0B4R --
↪assignments 10 --expiration 12
HIT extended.
[psiTurk server:on mode:sdbx #HITs:1]$ hit list --active
Stroop task
  Status: Assignable
  HITid: 2776AUC26DG6NRIGNVRFN0COYO0B4R
  max:13/pending:0/complete:0/remain:13
  Created:2014-03-07T21:36:33Z
  Expires:2014-03-08T21:48:33Z
```

Note that both the remaining number of assignments and the expiration time of the HIT have increased. One can also increase the number of assignments or the expiration independently.

25.6.3 hit expire

Usage:

```
hit expire (--all | <HITid> ...)
```

Expire one or more existing HITs, or expire all HITs using the `--all` flag.

Examples:

1. Expiring two HITs at once:

```
[psiTurk server:on mode:sdbx #HITs:4]$ hit expire 2Y0T3HVWAVKIMG42A2S75Z9943NNFG
↪2RVZXR24SMEZFG314ME9X8P9CPPH0X
expiring sandbox HIT 2Y0T3HVWAVKIMG42A2S75Z9943NNFG
expiring sandbox HIT 2RVZXR24SMEZFG314ME9X8P9CPPH0X
[psiTurk server:on mode:sdbx #HITs:2]$
```

2. Expiring all active HITs:

```
[psiTurk server:on mode:sdbx #HITs:2]$ hit expire --all
expiring sandbox HIT 2776AUC26DG6NRIGNVRFN0COYO0B4R
expiring sandbox HIT 2VUWA6X3YOCCVET8PKOPWINIWJFPO0
[psiTurk server:on mode:sdbx #HITs:0]$
```

25.7 worker

The `worker` command is used to list, approve and reject, and bonus worker assignments on Amazon mechanical Turk.

Commands

- `worker approve`
- `worker reject`
- `worker unreject`
- `worker bonus`
- `worker list`
- `psiturk_status`

25.7.1 worker approve

Usage:

```
worker approve (--hit <hit_id> | <assignment_id> ...)
```

Approve worker assignments for one or more assignment ID's, or use the `--hit` flag to approve all workers for a specific HIT.

Examples:

1. Approve a single assignment:

```
[psiTurk server:on mode:sdbx #HITS:0]$ worker approve_
↪21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
approved 21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
```

2. Approve all assignments for a given hit:

```
[psiTurk server:on mode:sdbx #HITS:0]$ worker approve --hit_
↪2QKHECWA6X3Y4QTYKCG5NXPTWYGMLF
approving workers for HIT 2QKHECWA6X3Y4QTYKCG5NXPTWYGMLF
approved 2MB011K274J7PY7FQ1ZN76UXH0ECED
approved 2UO4ZMAZHRR1T7J8NEVUH1KJCAKBY
```

25.7.2 worker reject

Usage:

```
worker reject (--hit <hit_id> | <assignment_id> ...)
```

Reject worker assignments for one or more assignment ID's, or use the `--hit` flag to reject all workers for a specific HIT.

Example rejecting a single assignment:

```
[psiTurk server:on mode:sdbx #HITS:0]$ worker reject 2Y9OVR14IXKOIZQL1E3WD6X30CD98U
rejected 2Y9OVR14IXKOIZQL1E3WD6X30CD98U
```

25.7.3 worker unreject

Usage:

```
worker unreject (--hit <hit_id> | <assignment_id> ...)
```

Unreject worker assignments for one or more assignment ID's, or use the `--hit` flag to unreject all workers for a specific HIT.

Note: Unrejecting an assignment automatically approves that assignment.

Example of unrejecting a single assignment:

```
[psiTurk server:on mode:sdbx #HITS:0]$ worker unreject 2Y9OVR14IXKOIZQL1E3WD6X30CD98U
unrejected 2Y9OVR14IXKOIZQL1E3WD6X30CD98U
```

25.7.4 worker bonus

Usage:

```
worker bonus (--amount <amount> | --auto) (--hit <hit_id> | <assignment_id> ...)
```

Grant bonuses to workers for one or more assignment ID's, or use the `--hit` flag to bonus all workers for a specific HIT.

Enter the bonus `--amount <amount>` in an X.XX format, or use the `--auto` flag to bonus each worker according to the 'bonus' field of hte database (requires a [custom bonus route](#) in the experiment's *custom.py* file).

Upon running `worker bonus`, you will be asked to input a reason for the bonus. This message will be displayed to workers who receive the bonus.

Note: You must approve the worker assignment *before* you grant a bonus.

Warning: While it isn't possible to approve an assignment more than once, it is possible to grant a bonus repeatedly. When running `worker bonus` with the `--hit` flag, only workers who have not yet received a bonus for the assignment will be bonused. However, when running `worker bonus` on individual assignments the worker will be bonused regardless of whether they have already received one.

Examples:

1. Bonusing an individual assignment. The bonus can be granted repeatedly, making this risky:

```
[psiTurk server:on mode:sdbx #HITS:0]$ worker bonus --amount 2.00
↪21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
Type the reason for the bonus. Workers will see this message: Here's a bonus!
gave bonus of $2.00 to 21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
```

(continues on next page)

(continued from previous page)

```
[psiTurk server:on mode:sdbx #HITs:0]$ worker bonus --amount 2.00
↪21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
Type the reason for the bonus. Workers will see this message: Here's another one!
gave bonus of $2.00 to 21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
```

2. Say there are approved assignments for a HIT, one already bonused, one not yet bonused. Bonusing by HIT prevents repeated bonuses:

```
[psiTurk server:on mode:sdbx #HITs:0]$ worker bonus --amount 2.00 --hit
↪2ECYT3DHJHP4RRU304P8USX9BCXU10
Type the reason for the bonus. Workers will see this message: you haven't been
↪bonused yet. Here's a bonus!
bonusing workers for HIT 2ECYT3DHJHP4RRU304P8USX9BCXU10
gave a bonus of $2.00 to 2MB011K274J7PY7FQ1ZN76UXH0ECED
bonus already awarded to 21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
```

3. If a compute-bonus route exists in the experiment *custom.py*, we can also use the `--auto` flag to automatically give each worker the correct bonus:

```
[psiTurk server:on mode:sdbx #HITs:0]$ worker bonus --auto --hit
↪2ECYT3DHJHP4RRU304P8USX9BCXU10
Type the reason for the bonus. Workers will see this message: Thanks for moving
↪science forward!
bonusing workers for HIT 2ZQIUB2YU98JX6A4V3C0IBJ9W0HL9P
gave a bonus of $3.00 to 27UQ45UUKQOYW1ZFLNJ8RG012VYDVP
gave a bonus of $2.50 to 24I1HPCGJ2D2H2KFPX80MPPSKQM933
```

Note: Unlike the commands to approve, reject, or unreject workers, the `worker bonus` command requires the psiturk shell to be launched in the same project as the HIT for which workers are being bonused, since the information about which workers have been bonused is stored in the experiment database.

25.7.5 worker list

Usage:

```
worker list [--submitted | --approved | --rejected] [--hit <hit_id>]
```

List all worker assignments, or list worker assignments fitting a given condition using the provided flags. Use the `--hit` flag to list workers for a specific HIT.

Examples:

1. Listing all submitted workers:

```
[psiTurk server:on mode:sdbx #HITs:0]$ worker list --submitted
[
  {
    "status": "Submitted",
    "assignmentId": "2VQHVI44OS2K18PW7EQSEAP5DPV5ZY",
    "workerId": "A2O6BB9HXEUX1",
    "submit_time": "2014-03-04T16:14:32Z",
    "hitId": "2ZRNZW6HEZ6OUI7FRTZ6CGUMGIQPZ0",
    "accept_time": "2014-03-04T16:14:05Z"
```

(continues on next page)

(continued from previous page)

```

    },
    {
      "status": "Submitted",
      "assignmentId": "2XB92NJKM05B2XAD1YN2JTP9TYXAFG",
      "workerId": "A2O6BB9HXEUXX1",
      "submit_time": "2014-03-03T23:35:17Z",
      "hitId": "2RWSCWY2AO02W03X0OFGTSCMKZZ22I",
      "accept_time": "2014-03-03T23:34:19Z"
    }
  ]
}
```

2. Listing approved workers for a specific HIT:

```
[psiTurk server:~$ mode:sdbx #HITs:0]$ worker list --approved --hit_
↪ 2ECYT3DHP4RRU304P8USX9BCXU10
listing workers for HIT 2ECYT3DHP4RRU304P8USX9BCXU10
[
  {
    "status": "Approved",
    "assignmentId": "21A8IUB2YU98ZV9C5BUL3FBJB5B8K7",
    "workerId": "A2O6BB9HXEUXX1",
    "submit_time": "2014-02-26T03:26:55Z",
    "hitId": "2ECYT3DHP4RRU304P8USX9BCXU10",
    "accept_time": "2014-02-26T03:26:36Z"
  }
]
```

25.7.6 psiturk_status

Usage:

psiturk_status

Display startup screen with message from psiturk.org.

Example:

```
[psiTurk server:off mode:sdbx #HITS:1]$ psiturk_status
```

http://psiturk.org

an open platform for science on Amazon Mechanical Turk

System status:

Hi all, You need to be running psiTurk version >= 1.0.5dev to use the Ad Server feature!

Check <https://github.com/NYUCCL/psiTurk> or <http://psiturk.org> for latest info.

(continues on next page)

(continued from previous page)

```
psiTurk version 1.0.8dev
Type "help" for more information.
[psiTurk server:off mode:sdbx #HITS:1]$
```

25.8 quit

Usage:

```
quit
```

Quits the psiTurk shell. If you have a server running, psiTurk will confirm that you want to quit before exiting, since quitting psiTurk turns off the server.

Example of quitting psiTurk with the server running:

```
[psiTurk server:on mode:sdbx #HITS:0]$ quit
Quitting shell will shut down experiment server. Really quit? y or n: y
Shutting down experiment server at pid 40182...
Please wait. This could take a few seconds.
$
```

25.9 server

The server command is used with a variety of subcommands to control the experiment server.

- *server on*
- *server off*
- *server restart*
- *server log*

25.9.1 server on

Start the experiment server.

Example:

```
[psiTurk server:off mode:sdbx #HITS:0]$ server on
Experiment server launching...
Now serving on http://localhost:22362
[psiTurk server:on mode:sdbx #HITS:0]$
```

25.9.2 server off

Shut down the experiment server.

Example:

```
[psiTurk server:on mode:sdbx #HITS:0]$ server off
Shutting down experiment server at pid 32911...
Please wait. This could take a few seconds.
[psiTurk server:off mode:sdbx #HITS:0]$
```

25.9.3 server restart

Runs `server off`, followed by `server on`.

25.9.4 server log

Opens the server log in a separate window. Uses Console.app on Max OS X and xterm on other systems.

25.10 status

Usage:

```
status
```

The `status` command updates and displays the server status and number of HITs available on AMT or in the worker sandbox.

Note: This information is also displayed in the psiTurk shell prompt, but `#HITS` is not updated after every command (as every update requires contacting the AMT server). `status` provides a way to make sure the prompt is up-to-date.

Example of using the `status` command in sandbox mode:

```
[psiTurk server:off mode:sdbx #HITS:1]$ status
Server: currently offline
AMT worker site - sandbox: 1 HITs available
```

25.11 mode

Usage:

```
mode
mode <which>
```

The `mode` command controls the current mode of the psiTurk shell. Type `mode live` or `mode sandbox` to switch to either mode, or simply `mode` to switch to the opposite mode. The current mode affects almost every psiturk shell command. For example, running `hit create` while in sandbox mode will create a HIT in the sandbox, while running it in live mode will create a HIT on the live AMT site. Similarly, commands like `worker list all` or `hit list all` will list assignments and HITs from either the live site or the sandbox, depending on your mode.

Note: Switching the psiturk shell mode while the server is running requires the server to restart, since at the end of the experiment participants need to be correctly redirected back to either the live AMT site or the sandbox. Therefore, **you should not change modes while you are serving a live HIT to workers.**

Examples:

1. Switching mode, with and without <which> specifier:

```
[psiTurk server:off mode:sdbx #HITs:0]$ mode
Entered live mode
[psiTurk server:off mode:live #HITs:0]$ mode sandbox
Entered sandbox mode
[psiTurk server:off mode:sdbx #HITs:0]$
```

2. Switching mode with the server running:

```
[psiTurk server:on mode:sdbx #HITs:0]$ mode
Switching modes requires the server to restart. Really switch modes? y or n
↩n: y
Entered live mode
Shutting down experiment server at pid 33447...
Please wait. This could take a few seconds.
Experiment server launching...
Now serving on http://localhost:22362
[psiTurk server:on mode:live #HITs:0]$
```

Type n instead to abort the mode switch harmlessly.

MIGRATING FROM PSITURK 2 TO 3

26.1 Announcing psiTurk 3.0

A message from project founder Todd Gureckis.

psiTurk 2.0 launched on April 28, 2014 and there has been (according to github) 786 commits since then to the project from a wide variety of contributors. The evolution and longevity of the project really has exceeded anything the original authors thought would be possible. In the seven years since 2.0 was first tagged so many things have changed about web experimentation, web application development, the Amazon turk API, and even the Python ecosystem.

One part of psiTurk that has always been both a blessing and a curse is the reliance on several services provided by xxx.psiturk.org. This includes the “Ad server” and several other api elements that were envisioned to distribute timely information to users of the system. Generally this seemed like a good engineering solution to a problem, but centralization is generally bad because if something happened to psiturk.org (annual SSL certs renew on time) then the system goes down for everyone. Every year during conference deadlines the original project creator would lose sleep.

As a result, a major change in psiTurk 3.0 is to make the system decoupled from the services on psiturk.org. It is possible now to easily get a SSL signed connection to a cloud-based server (e.g., heroku) and to run the server in a “headless” mode. This gets around the need for the psiturk.org “Ad server” which was written in 2014 and has basically never been updated since. In a way this means psiTurk acts more like a traditional Flask “web application” rather than an interactive command line tool, although the command line interface remains for interacting with Amazon and for development.

In addition to the hard decoupling work (led really by Dave Eargle), version 3.0 offers several new features including a “campaign mode” which allows you to run a certain number of subject (say 100) by repeatedly posting lower cost 9 assignment HITs, and a dashboard for managing hits. In addition, a major change since 2.0 is support for Python 3.0 and the changes that entailed to work with more recent versions of boto Mturk python api.

There may be some growing pains as people adjust to the new workflow so a goal is to update the documentation to reflect the new changes as soon as possible. As always, assistance from the psiturk community to make this documentation is always appreciated!

26.2 Migration technical considerations

Below are some notable technical differences between psiTurk 2 and psiTurk 3.

26.2.1 No More Secure Ad Server

Psiturk 3 drops all psiturk.org-hosted services, including most notably the Secure Ad Server. MTurk still requires that ads be hosted over https, but in the years since the secure ad server was launched, it has become easier to obtain an SSL certificate. psiTurk supports hosting ads on [Heroku](#). Hosting an experiment on heroku provides free SSL. See [Deploy to Heroku](#) for more information.

If your lab has a static IP address and somewhat technical prowess, you might also choose to obtain your own certificate for free by first buying a domain name and then using [letsencrypt](#).

Because the secure ad server has gone away, you will need to specify your `ad_url` in your config file. This setting is passed to mturk when you run `hit_create`. See the `hit_configuration_ad_url` section of the documentation.

26.2.2 More flexible configuration approach

psiTurk 3 also is more flexible in how it handles configuration variables, respecting environment variables over psiturk defaults. This enables having different config settings locally versus on hosted platforms such as Heroku. See the [configuration-overview](#) page for more information.

Note that the location of a few config variables has changed – specifically, `contact_email_on_error` and `cutoff_time` have moved under the `Task Parameters` section.

Note: If you are migrating a current experiment, it is recommended that you copy [the example config file from github](#) and fill in your experiment's values.

26.2.3 No More Python 2

psiTurk 3 drops support for python 2, for various reasons. See the [changelog](#) for more details.

26.2.4 Optimized psiturk.js preloadPages()

`psiturk.js`'s `preloadPages()` now returns a javascript [Promise](#). It does this so that it can simultaneously start to load *all* of the pages, rather than one at a time. The function will `resolve` when *all* pages have finished preloading. Look in the [example's task.js](#) for more detailed comments on how you need to refactor your `task.js` to use the new `preloadPages()`. Also see [the api page for preloadPages](#) for other examples.

26.2.5 Detailed changelog

For a more detailed listing of changes between psiturk 2 and 3, see the [changelog on github](#).

DISCLAIMER

psiturk is free, open source software provided to scientists to aid in research. Because it helps you run paid experiments online using crowdworking websites where you transfer money, errors in the software, or in your use of the software, can lead to **loss of money**. This is the very nature of online research (errors may mean someone will actually do your task and you need to pay them as a result).

Our belief is that these types of errors can be best limited by having open, peer-reviewable software and sharing bug reports between labs and research groups. In other words, even if you wrote this software yourself it is possible that some bug could cost you money when getting started.

Danger: We take no responsibility for your use of the software. We make no claims that it is entirely bug-free and any errors are not our responsibility. This is a community-run, community-supported system and not a company selling a product. We use the software in our lab and, when used correctly, has never caused us to lose money due to mistakes. However, it is **always possible to mis-use the software in a costly way**.

In addition, while we strive to keep the services and dependencies related to this project running and up-to-date, certain things can happen that, in the short-term, affect your ability to collect data. Again, using the system you must understand what the risks are. The good news is that because the system is open source if there is a problem everyone can read the code themselves and make suggestions on how to fix things.

Some suggestion to avoid costly mistakes from happening:

1. Test your code a lot in the sandbox to make sure every stage is working and you understand what psiTurk is doing.
2. Run small batches at a time to verify everything is working
3. Keep your payment account balance reasonably low at any point in time. It is impossible to spend more money than is in your account at any point in time.
4. Exit the psiturk server or shut down remote servers when you are not using it to collect data. This ensures that no one will be able to actually perform your task and then claim they are owed payment. This also limits the ability of bots and other scammers to reverse engineer your task.
5. When testing “live”, explain in the text of your Ad that this is a test and you are looking for feedback. Workers get frustrated when you put bad or broken experiments online, but are often very helpful if you explain that you are hoping to get feedback on an unfinished project.

GETTING HELP

There are a number of ways to get help with psiTurk.

1. The <https://psiturk.org> has meta-information about the system.
2. Usage questions should go to the psiTurk Google Group [here](#). Search for answers to common questions or post your own. Chances are if you run into a problem someone else will as well.
3. Potential bugs in the code, or feature requests, should be posted as a [github issue](#). This is an open discussion of possible issues, bugs, feature requests, etc. Browse the open and closed issues first before posting. See the [guide for contributors](#) for more information about using the issues tracker.
4. `Todd Gureckis taught a class covering online data collection and psiTurk at NYU Spring 2014. All lectures were videotaped and are available [here](#)
5. Follow [@psiturk](#) on Twitter for helpful tips and breaking news.
6. If all else fails and you feel you simply cannot get help, you can consider emailing authors@psiturk.org, the benevolent dictators of the project and system architects. However, if you haven't first pursued the above options, you may not get a quick response.

CODE OF CONDUCT

29.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

29.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

29.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

29.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

29.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at authors@psiturk.org. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

29.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

29.6.1 1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

29.6.2 2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

29.6.3 3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

29.6.4 4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

29.7 Attribution

This Code of Conduct is adapted from the [Contributor Covenant](https://www.contributor-covenant.org/version/2/0/code_of_conduct.html), version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by [Mozilla's code of conduct enforcement ladder](#).

For answers to common questions about this code of conduct, see the FAQ at <https://www.contributor-covenant.org/faq>. Translations are available at <https://www.contributor-covenant.org/translations>.

PROJECT ROADMAP

psiTurk is always looking to improve and to increase the number of contributors. We thought it would be helpful to lay out a basic roadmap of where we would like to see the project go in the future. This roadmap may inspire you to implement a new feature!

30.1 General priorities

30.1.1 Documentation

The documentation is greatly lagging behind progress on the psiTurk platform. We need help with people debugging documentation, improving it, and making additions! Notice how all documentation pages (including this one!) include a link to “Edit on GitHub”. Make a pull request and help us improve these docs!

30.1.2 Automated testing

The version 2.0 release introduced a number of new features which are fairly complex because they require communication over the Internet, RESTful APIs, etc. . . While there are automated unit tests for many of these features, it is important to have better tests of these features. Testing isn’t glamorous but writing tests improves your health, looks, and chances of getting in heaven.

30.1.3 Alternative database solutions

Currently psiTurk offers a variety of database solutions including local SQLite files, self-administered MySQL servers, and MySQL processes hosted on Amazon’s Web Services (RDS) platform. However, all of these are a little clunky and require users to know quite a bit about data management. The demands placed on these databases by a single experiment are not excessive, and thus there might be a more robust solution (e.g., NoSQL). One possibility is to host a robust cloud-based data API off psiturk.org.

30.1.4 psiturk.js

All projects currently should use **psiturk.js** to save data to the server and update the user status as they progress. It might be nice if these included additional features including easily displaying instructions, providing simple quizzes, etc... In theory many parts of the psiturk command shell could be moved into the psiturk.js library (e.g., one could even create hits and ads via javascript calls). This might eventually allow the power of the psiturk platform to be leveraged even on simple, standard web server platforms (i.e., not relying on Flask).

30.1.5 Ad Server

The Ad Server has the potential to gather valuable data about participants in studies, how naive they are, etc... Currently only a limited number of statistics are gathered, and much of this data is not publically accessible via an API or interface. Future versions of the psiturk.org dashboard could provide users with more interesting statistics about participants in their experiments, their geographic location, etc...

30.1.6 Unique IP issues

A major issue with psiTurk is that it requires a unique, Internet addressable IP address. This is a hurdle at some universities or companies. This is a bug and a feature at some level. The feature side is that for many users the ability to serve experiments off their local computer obviates the need for a dedicated server and simplifies some web security issues. For other users thought this is a frustrating hurdle to overcome in order to use psiturk. We are interesting in the community's thoughts about this and suggestions about best practices include cloud based hosting systems like Salesforce's Heroku and Amazon's AWS.

30.2 Version 3.0

We envision that eventually psiturk could move entirely into the cloud (i.e., no need for user to install command line tool). This may be supported by changes and extensions to the psiturk.org API and the psiturk.js library. The emphasis in our initial development has been on advanced users/programmers comfortable in a unix environment, but future version could emphasize novice web programmers who are new to online experiments (e.g., undergrads).

If you have ideas about future directions for the project the Github [issues tracker](#) is a great place to share them.

CONTRIBUTING TO PSITURK

Note: This guide is copied more or less from the [contributors guidelines](#) of the [gunicorn](#) project. Alternations were made for the nature of this particular project. An up to date copy of this guide always resides [here](#).

Want to contribute to psiTurk? Awesome! Here are instructions to get you started. We want to improve these as we go, so please provide feedback.

31.1 Contribution guidelines

31.1.1 Pull requests are always welcome

We are always thrilled to receive pull requests, and do our best to process them as fast as possible. Not sure if that typo is worth a pull request? Do it! We will appreciate it.

If your pull request is not accepted on the first try, don't be discouraged! If there's a problem with the implementation, hopefully you received feedback on what to improve.

We're trying very hard to keep psiTurk lean, focused, and useable. We don't want it to do everything for everybody. This means that we might decide against incorporating a new feature. However, there might be a way to implement that feature *on top of* psiTurk.

31.1.2 Discuss your design on the mailing list

We recommend discussing your plans in our [Google group](#) before starting to code - especially for more ambitious contributions. This gives other contributors a chance to point you in the right direction, give feedback on your design, and maybe point out if someone else is working on the same thing.

31.1.3 Create issues...

Any significant improvement should be documented as a [github issue](#) before anybody starts working on it.

31.1.4 ...but check for existing issues first!

Please take a moment to check that an issue doesn't already exist documenting your bug report or improvement proposal. If it does, it never hurts to add a quick "+1" or "I have this problem too". This will help prioritize the most common problems and requests.

31.1.5 Conventions

Fork the repo and make changes on your fork in a new feature branch:

- If it's a bugfix branch, name it XXX-something where XXX is the number of the issue
- If it's a feature branch, create an enhancement issue to announce your intentions, and name it XXX-*something* where XXX is the number of the issue.

Make sure you include relevant updates or additions to documentation when creating or modifying features.

Write clean code.

Pull requests descriptions should be as clear as possible and include a reference to all the issues that they address.

Code review comments may be added to your pull request. Discuss, then make the suggested modifications and push additional commits to your feature branch. Be sure to post a comment after pushing. The new commits will show up in the pull request automatically, but the reviewers will not be notified unless you comment.

Commits that fix or close an issue should include a reference like *Closes #XXX* or *Fixes #XXX*, which will automatically close the issue when merged.

Add your name to the THANKS file, but make sure the list is sorted and your name and email address match your git configuration.

31.1.6 Contributing to the docs

Our docs are currently hosted at [readthedocs](#). Readthedocs uses [Sphinx](#) as the backend for their documentation so in order to update the docs you will first have to install Sphinx simply by typing:

```
easy_install -U Sphinx
```

on the command line.

There's a Makefile in the docs directory, so you can generate the docs by running *make* on the command line, for example:

```
make html
```

will generate the html docs in *_build/html*. Running make with no arguments will show you the available subcommands.

All documentation files are in the docs folder and are formatted as reStructured Text. A good, detailed manual for the reStructured Text syntax can be found [here](#).

Some essentials:

The index page is the main page that users see will see when they open the docs. It is also how readthedocs generates the sidebar that contains all the names of individual pages in the documentary so it is important that this is formatted correctly.

The main important feature is the [toctree](#).

The toctree just looks like this:

```
.. toctree::
    forward
    install
    quickstart
    recording
```

Sphinx will go through the pages listed in the toctree, search for subject headers and create both links for the index page and the sidebar in the correct format in the order that the pages are listed. For this reason, it is also very important that subjected headers be used correctly on the individual pages. For example, the forward page has a title that looks like this:

```
Forward
=====
```

and subtitles that look like this:

```
What is psiTurk?
~~~~~
```

It actually doesn't matter what character you use for the underline, it can be any of

```
= - ` ' " : ~ ^ _ * + # < >
```

but it must be consistent since all headers with the same character will be at the same level. For convenience, we are using ===== to mean title and ~~~~~ to mean sub header. Some other basic things in rST:

Links look like this:

```
``Getting psiTurk installed on your computer <install.html>``_
```

with the actual page in angle brackets. If the link is to another page within the docs, you only need to include the name of the page. Whenever you include a code example, put this line before:

```
.. code:: javascript
```

All pages on readthedocs.org (including this one) have a link to “Edit on Github.” This can be a great way to “steal” formatting ideas for your documentation edits.

31.2 Decision process

31.2.1 How are decisions made?

In general, all decisions affecting psiTurk, big and small, follow the same 3 steps:

- Step 1: Open a pull request. Anyone can do this.
- Step 2: Discuss the pull request. Anyone can do this.
- Step 3: Accept or refuse a pull request. The little dictators do this (see below “Who decides what?”)

31.2.2 Who decides what?

psiTurk, like gunicorn, follows the timeless, highly efficient and totally unfair system known as [Benevolent dictator for life](#). In the case of psiTurk, there are multiple little dictators which are the core members of the [gureckislab](#) research group and alumni. The dictators can be emailed at authors@psiturk.org.

For new features from outside contributors, the hope is that friendly consensus can be reached in the discussion on a pull request. In cases where it isn't the original project creators [John McDonnell](#) and/or [Todd Gureckis](#) will intervene to decide.

The little dictators are not required to create pull requests when proposing changes to the project.

31.2.3 Is it possible to become a little dictator if I'm not in the Gureckis lab?

Yes, we will accept new dictators from people esp. engaged and helpful in improving the project.

31.2.4 How is this process changed?

Just like everything else: by making a pull request :)

WELCOME TO PSITURK

`psiturk` is an open-source Python library that makes it easy to create high-quality behavioral experiments that are delivered over the Internet using a web browser.

`psiturk` is not, *by itself*, used for creating surveys or interfaces in the browser (for that we recommend tools like `jspsych` or `d3.js`). Instead, `psiturk` solves the *other* myriad of problems that come up in web experimentation including:

- Reliably and securely serving webpages to participants over the internet
- Blocking repeat participation (when desired)
- Counterbalancing conditions
- Incrementally saving data to databases
- Simplifying the process of soliciting and approving work on crowdworking sites

`psiturk` is used in [many research fields](#) ranging from cognitive science, psychology, neuroscience, bioinformatics, marketing, computer security, user interface testing, computer science, and machine learning in both academia and industry.

32.1 Video introduction

Still unsure if `psiturk` is for you? Try this quick five minute video introduction!

32.2 How to use our docs

The docs are broken up into several sections:

- **First Steps:** include *What is psiturk?* as an overview. Then there is a quick-start guide in two main parts: *Getting started developing* and *Collecting data*.
- **Topic guides:** give you background on specific aspects of `psiturk` including how to *install*, how to *record* and *retrieve* data, how to *set up a Mechanical Turk account*, and how to use `psiturk` with other recruitment systems such as *in the lab*.
- **Tutorials:** provides high level overview of specific tasks such as *using psiturk with jsPsych* or *automatically computing performance-based bonus payment amounts*. Make sure to check out the sections on the *sample project walk-through*.
- **Reference guides:** are the bread and butter of how our *APIs* and *configuration files* work and will give you short, actionable explanations of specific functions and features.

- **Support:** gives you more options for when you're *stuck* or want to talk about an idea with other people.
-

32.3 Open source, community-built

psiturk is built using an open source ([MIT License](#)) model that draws from the community to share best online experiment practices. Version 1.0 was launched in November 2013 and since then psiturk has maintained a diverse and supportive community that includes over 40 contributors providing 1800 commits. We happily accept bug reports, feature requests, and – even better – pull requests!

32.4 Join the community

Please come join us on the [community forum](#) or follow us on [twitter](#). We love to hear your questions, ideas, and help you work through your bugs on our github [issues tracker](#)! The project leaders are [Dave Eargle](#) and [Todd Gureckis](#).

Note: Citing this project in your papers:

To credit *psiturk* in your work, please cite both the original journal paper and a version of the Zenodo archive. The former provides a high level description of the package, and the latter points to a permanent record of all *psiturk* versions (we encourage you to cite the specific version you used). Example citations (for *psiturk* 3.0.6):

Zenodo Archive:

Eargle, David, Gureckis, Todd, Rich, Alexander S., McDonnell, John, & Martin, Jay B. (2021, March 28). psiTurk: An open platform for science on Amazon Mechanical Turk (Version v3.0.6). Zenodo. <http://doi.org/10.5281/zenodo.3598652>

Journal Paper:

Gureckis, T.M., Martin, J., McDonnell, J., Rich, A.S., Markant, D., Coenen, A., Halpern, D., Hamrick, J.B., Chan, P. (2016) psiTurk: An open-source framework for conducting replicable behavioral experiments online. *Behavioral Research Methods*, 48 (3), 829-842. DOI: <http://doi.org/10.3758/s13428-015-0642-8>
